

Содержание

Введение.....	4
1 Спецификация языка программирования.....	5
1.1 Характеристика языка программирования.....	5
1.2 Определение алфавита языка программирования.....	5
2 Структура транслятора.....	15
2.1 Компоненты транслятора, их назначение и принципы взаимодействия.....	15
2.2 Перечень входных параметров транслятора.....	16
2.3 Перечень протоколов, формируемых транслятором и их содержимое.....	17
3 Разработка лексического анализатора.....	18
3.1 Структура лексического анализатора.....	18
3.2 Контроль входных символов.....	18
3.3 Удаление избыточных символов.....	19
3.4 Перечень ключевых слов, сепараторов, символов операций, соответствующим им лексемам и конечных автоматов.....	19
3.5 Основные структуры данных.....	20
3.6 Принцип обработки ошибок.....	21
3.7 Структура и перечень сообщений лексического анализатора.....	21
3.8 Параметры лексического анализатора и режимы его работы.....	22
3.9 Алгоритм лексического анализа.....	22
3.10 Контрольный пример.....	23
4 Разработка синтаксического анализатора.....	24
4.1 Структура синтаксического анализатора.....	24
4.2 Контекстно-свободная грамматика, описывающая синтаксис языка.....	24
4.3 Построение конечного магазинного автомата.....	25
4.4 Основные структуры данных.....	26
4.5 Описание алгоритма синтаксического разбора.....	26
4.6 Структура и перечень сообщений синтаксического анализатора.....	27
4.7 Параметры синтаксического анализатора и режимы его работы.....	27
4.8 Принцип обработки ошибок.....	27
4.9 Контрольный пример.....	28

5	Разработка семантического анализатора.....	30
5.1	Структура семантического анализатора.....	30
5.2	Функции семантического анализа.....	30
5.3	Структура и перечень сообщений семантического анализатора.....	30
5.4	Принцип обработки ошибок.....	31
5.5	Контрольный пример.....	31
6	Вычисление выражений.....	33
6.1	Выражения, допускаемые языком.....	33
7	Генерация кода.....	34
7.1	Структура генератора кода.....	34
7.2	Представление типов данных в оперативной памяти.....	34
7.3	Библиотека.....	34
7.4	Особенности алгоритма генерации кода.....	35
7.5	Входные параметры генератора кода.....	37
7.6	Контрольный пример.....	37
8	Тестирование транслятора.....	38
8.1	Общие положения.....	38
8.2	Результаты тестирования.....	38
	Заключение.....	39
	Список использованных источников.....	40
	Приложение А.....	41
	Приложение Б.....	42
	Приложение В.....	45
	Приложение Г.....	46
	Приложение Д.....	48
	Приложение Е.....	51
	Приложение Ж.....	60
	Приложение И.....	61

Введение

Целью курсового проекта является разработка собственного языка программирования ZEI-2020 и транслятора для него. Написание транслятора будет осуществляться на языке C++.

Транслятор — программа, которая преобразует исходный код на одном языке в исходный код на другом языке программирования. Его компонентами являются лексический, синтаксический и семантический анализаторы, а также генератор кода.

Исходя из цели, задачи на курсовой проект можно сформировать следующим образом:

- разработка языка программирования ZEI-2020, создание спецификации к нему (глава 1);
- разработка лексического анализатора (глава 3);
- разработка синтаксический анализатор (глава 4);
- разработка семантический анализатор (глава 5);
- разработка генератора кода, или интерпретатора в язык C# (глава 7);
- тестирование транслятора (глава 8).

1 Спецификация языка программирования

1.1 Характеристика языка программирования

Язык ZEI-2020 — это универсальный язык высокого уровня. Он является строго типизированным, процедурным, интерпретируемым.

1.2 Определение алфавита языка программирования

Базовый алфавит языка ZEI-2020 программирования использует символы восьмибитной кодировки ASCII, представленной на рисунке 1.1.

Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ
0	0	спец. NOP	32	20	спец. SP (Пробел)	64	40	@	96	60	`	128	80	Ђ	160	A0	Ѐ	192	C0	А	224	E0	а
1	1	спец. SOH	33	21	!	65	41	A	97	61	a	129	81	Г	161	A1	Ђ	193	C1	Б	225	E1	б
2	2	спец. STX	34	22	"	66	42	B	98	62	b	130	82	,	162	A2	Ѓ	194	C2	В	226	E2	в
3	3	спец. ETX	35	23	#	67	43	C	99	63	c	131	83	і	163	A3	Ј	195	C3	Г	227	E3	г
4	4	спец. EOT	36	24	\$	68	44	D	100	64	d	132	84	..	164	A4	□	196	C4	Д	228	E4	д
5	5	спец. ENQ	37	25	%	69	45	E	101	65	e	133	85	...	165	A5	Г	197	C5	Е	229	E5	е
6	6	спец. ACK	38	26	&	70	46	F	102	66	f	134	86	†	166	A6	і	198	C6	Ж	230	E6	ж
7	7	спец. BEL	39	27	'	71	47	G	103	67	g	135	87	‡	167	A7	§	199	C7	З	231	E7	з
8	8	спец. BS	40	28	(72	48	H	104	68	h	136	88	€	168	A8	Е	200	C8	И	232	E8	и
9	9	спец. Tabуляция	41	29)	73	49	I	105	69	i	137	89	‰	169	A9	Є	201	C9	Й	233	E9	й
10	0A	спец. LF (Возв. каретки)	42	2A	*	74	4A	J	106	6A	j	138	8A	Љ	170	AA	Є	202	CA	К	234	EA	к
11	0B	спец. VT	43	2B	+	75	4B	K	107	6B	k	139	8B	«	171	AB	»	203	CB	Л	235	EB	л
12	0C	спец. FF	44	2C	,	76	4C	L	108	6C	l	140	8C	Ђ	172	AC	—	204	CC	М	236	EC	м
13	0D	спец. CR (Новая строка)	45	2D	-	77	4D	M	109	6D	m	141	8D	Њ	173	AD	-	205	CD	Н	237	ED	н
14	0E	спец. SO	46	2E	.	78	4E	N	110	6E	n	142	8E	Ћ	174	AE	Ѓ	206	CE	О	238	EE	о
15	0F	спец. SI	47	2F	/	79	4F	O	111	6F	o	143	8F	Ќ	175	AF	Ѐ	207	CF	П	239	EF	п
16	10	спец. DLE	48	30	0	80	50	P	112	70	p	144	90	ђ	176	B0	°	208	D0	Р	240	F0	р
17	11	спец. DC1	49	31	1	81	51	Q	113	71	q	145	91	‘	177	B1	±	209	D1	С	241	F1	с
18	12	спец. DC2	50	32	2	82	52	R	114	72	r	146	92	’	178	B2	І	210	D2	Т	242	F2	т
19	13	спец. DC3	51	33	3	83	53	S	115	73	s	147	93	“	179	B3	і	211	D3	У	243	F3	у
20	14	спец. DC4	52	34	4	84	54	T	116	74	t	148	94	”	180	B4	г	212	D4	Ф	244	F4	ф
21	15	спец. NAK	53	35	5	85	55	U	117	75	u	149	95	•	181	B5	μ	213	D5	Х	245	F5	х
22	16	спец. SYN	54	36	6	86	56	V	118	76	v	150	96	—	182	B6	¶	214	D6	Ц	246	F6	ц
23	17	спец. ETB	55	37	7	87	57	W	119	77	w	151	97	—	183	B7	·	215	D7	Ч	247	F7	ч
24	18	спец. CAN	56	38	8	88	58	X	120	78	x	152	98	◆	184	B8	ё	216	D8	Ш	248	F8	ш
25	19	спец. EM	57	39	9	89	59	Y	121	79	y	153	99	™	185	B9	№	217	D9	Щ	249	F9	щ
26	1A	спец. SUB	58	3A	:	90	5A	Z	122	7A	z	154	9A	љ	186	BA	є	218	DA	Ъ	250	FA	ъ
27	1B	спец. ESC	59	3B	;	91	5B	[123	7B	{	155	9B	»	187	BB	»	219	DB	Ы	251	FB	ы
28	1C	спец. FS	60	3C	<	92	5C	\	124	7C		156	9C	њ	188	BC	ј	220	DC	Ь	252	FC	ь
29	1D	спец. GS	61	3D	=	93	5D]	125	7D	}	157	9D	ќ	189	BD	ѕ	221	DD	Э	253	FD	э
30	1E	спец. RS	62	3E	>	94	5E	^	126	7E	~	158	9E	ћ	190	BE	ѕ	222	DE	Ю	254	FE	ю
31	1F	спец. US	63	3F	?	95	5F	_	127	7F	~	159	9F	и	191	BF	і	223	DF	Я	255	FF	я

Рисунок 1.1 ASCII таблица кодов сиволов (Windows - 1251).

Для записи ключевых слов и конструкций языка используются строчные буквы латинского алфавита [a..z], а также символы-сепараторы, описанные в таблице 1.1.

Запрещённые символами языка являются все символы с десятичными кодами 0-8, 11-19, 127-255. При этом символы с кодами 224-255 (строчная кириллица) могут быть допущены в строковом литерале.

Допущенный в исходном коде запрещённый символ, при условии написания его вне строкового литерала, генерирует исключение, прекращая работу транслятора. Исключение также фиксируется в протоколе работы транслятора.

1.3 Применяемые сепараторы

Символы, входящие в число сепараторов, представлены в таблице 1.1.

Таблица 1.1 Сепараторы языка ZEI-2020.

Код символа	Символ	Наименование	Описание
32		пробел	допускается везде, кроме идентификаторов и ключевых слов;
59	!	восклицательный знак	разделитель инструкций;
40, 41	()	открывающая круглая скобка; закрывающая круглая скобка	используется для записи параметров функций (как фактические, так и формальные); условие в циклах или условных конструкциях; изменяют приоритетность операций;
123, 125	[]	открывающая квадратная скобка; закрывающая квадратная скобка	ограничение программного блока (тело функции, условной конструкции или цикла);
44	,	запятая	разделитель параметров функции;
126	~	тильда	однострочный комментарий (вся строка после знака игнорируется);

1.4 Применяемые кодировки

Язык ZEI-2020 использует расширенную кодировку ACSII, описание которой представлено в подразделе 1.1.

1.5 Типы данных

В языке ZEI-2020 существуют типы данных, представленные в таблице 1.2.

Таблица 1.2 Типы данных языка ZEI-2020.

Тип	Описание
tiny	целочисленный (1 байт), знаковый тип данных; диапазон значений: от -128 до 127; по умолчанию инициализируется нулем;
symbolic	строковый тип данных; может содержать строчные символы латиницы и кириллицы, а также пробелы, символы табуляции и символы знаков препинания (максимально 255 символов); по умолчанию инициализируется пустой строкой “”;
logical	логический тип данных, значения «true» или «false», по умолчанию инициализируется значением «false»

Допустимые операции для каждого типа данных определены в пункте 1.12.

1.6 Преобразование типов данных

Язык ZEI-2020 не поддерживает преобразование типов данных.

1.7 Идентификаторы

Для именования функций, их параметров, а также переменных используются идентификаторы.

Идентификаторы в языке ZEI-2020 могут содержать лишь строчные символы латинского алфавита. Максимальное количество символов в идентификаторе — 15 (при превышении значения генерируется исключение). Идентификатор не должен совпадать с ключевыми словами языка.

<буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<идентификатор> ::= <буква> { <буква> }.

Для стандартной библиотеки резервируются следующие идентификаторы:

- symbtotiny;
- symblen;
- sconcat.

Таким образом, примеры правильных идентификаторов языка ZEI-2020 — var, x, а неправильных — tiny (совпадает с ключевым словом), xxxxxxxxxxxxxxxxx (превышено максимально допустимое количество символов).

1.8 Литералы

Литерал — элемент программы, который непосредственно представляет значение.

В языке ZEI-2020 предусмотрены типы литералов, описанные в таблице 1.3.

Для целочисленных литералов в разных представлениях предусмотрены постфиксы и префиксы, также указанные в таблице 1.3. до или после названия представления. Так, десятичное представление определяется как представление по умолчанию и постфикса не имеет, но оно может иметь префикс «m», определяющий, что число является отрицательным.

Таблица 1.3 Литералы языка ZEI-2020.

Тип	Представление, постфикс	Примечание
целочисленный	«m»десятичное, двоичное «b», восьмеричное «q», шестнадцатеричное «h»	число в диапазоне от -128 до 127, интерпретируется как tiny; при выходе из диапазона генерируется исключение;
строковый	Последовательность или символов или символ; заключен в кавычки	строковый литерал, должен быть заключён в кавычки; интерпретируется как symbolic; литералы «true» и «false» зарезервированы для типа данных logical

Формально литералы языка ZEI-2020 можно определить следующим образом:

<буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | б | в | г | д | е | ж | з | и | ё | к | л | м | н | о | п | р | с | т | у | ф | х | ц | ч | ш | щ | ь | ы | ё | ю | я

<знак> ::= пробел | , | ? | : | ; | ! | - | табуляция

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<строковый литерал> ::= "(<буква> | <цифра> | <знак>) { (<буква> | <цифра> | <знак>) } "

<целочисленный десятичный> := [m] { / <цифра> / }

<целочисленный двоичный> := { / (0 | 1) / } b

<целочисленный восьмеричный> := { / (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7) / } q

<целочисленный шестнадцатеричный> := { / (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F) / } h

Примеры правильных литералов языка ZEI-2020 — 10, 39h, 15q, 11b, “abc”, а неправильных — 9123, 0x14, m12q, “ABC”.

1.9 Объявление данных

Объявление идентификаторов в языке происходит с помощью ключевого слова `set`. Обязательно указание типа данных. Не обязательно присваивать значение при объявлении. Запрещено использование необъявленных идентификаторов или переобъявление. Возможно объявление одинаковых идентификаторов в разных областях видимости. Объявление глобальных переменных не предусмотрено. Объявление идентификаторов внутри условных конструкций, вне блока функции, в теле условных конструкций, а также в теле цикла не допускается.

Формально объявление идентификатора можно описать следующим образом:

```
set<тип      данных>    <идентификатор>[:(<литерал>|<идентификатор>|
<выражение>)]!
```

1.10 Инициализация данных

Присвоение значение идентификатора допускается при его объявлении. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака присваивания. Объектами-инициализаторами могут быть идентификаторы, литералы, выражения и вызовы функций. Предусмотрены значения по умолчанию, если переменные не инициализированы: значение 0 для целочисленного типа данных и строка нулевой длины ("") для строкового.

1.11 Инструкции языка

Язык ZEI-2020 предусматривает инструкции, описанные в таблице 1.4. Инструкции языка разделяются символом «!».

Таблица 1.4 Инструкции языка ZEI-2020.

Инструкция	Описание
<code>set<типа данных><идентификатор>:<значение>!</code>	объявление переменной
<code><тип данных> func <идентификатор>(<формальные параметры>)[]</code>	объявление функций
<code>show(<идентификатор> <литерал>)!</code>	вывод данных
<code><идентификатор>:<литерал>!</code> <code><идентификатор>:<выражение>!</code> <code><идентификатор1>:<идентификатор2>!</code>	присвоить значение
<code><идентификатор функции>(<фактические параметры>)!</code>	вызов функции
<code>giveback <идентификатор> <литерал>!</code>	возврат из функции

1.12 Операции языка

В языке ZEI-2020 предусмотрены операции, описанные в таблице 1.5.

Среди арифметических операций наибольшую приоритетность операций сложения и деления. При одинаковом приоритете первой выполнится операция, расположенная левее. Изменить приоритетность можно с помощью круглых скобок.

Таблица 1.5 Операции языка ZEI-2020.

Группа	Операторы	Типы данных
Логические	1. > (бинарный) – оператор «больше»; 2. < (бинарный) – оператор «меньше»; 3. =(бинарный) – оператор проверки на равенство; 4. ^ (бинарный) – оператор проверки на неравенство;	tiny; = и ^ можно применять и к типу logical;
Бинарные (сдвиги)	1. \ – сдвиг влево; <идентификатор литерал>:<идентификатор литерал>\<литерал>! 2. / – сдвиг вправо; <идентификатор литерал>:<идентификатор литерал>/<литерал>!	tiny;
Арифметические	1. + (бинарный) – оператор сложения; 2. - (бинарный) – оператор вычитания; 3. * (бинарный) – оператор произведения; 4. # (бинарный) – оператор деления;	tiny.

1.13 Выражения и их вычисление

В выражениях языком ZEI-2020 предусмотрено использование операций, описанных в таблице 1.5.

Предусмотрены следующие правила составления выражений:

- выражения читаются слева направо и записываются в одну строку;
- для изменения приоритета операция используются круглые скобки;
- допустимо использование арифметических выражений с использованием вызова функции.

1.14 Конструкции языка

В языке ZEI-2020 можно использовать конструкции языка, описанные в таблице 1.6.

Таблица 1.6 Конструкции языка ZEI-2020.

Конструкция	Реализация, описание
1	2
Главная функция, точка входа в программу	<pre>perform [инструкции]</pre>
Функция	<pre><тип данных> func <идентификатор>(<формальные параметры>) [инструкции giveback <идентификатор/литерал>!]</pre>
Оператор цикла	<pre>loop(условие) [инструкции]</pre> <p>Цикл (операторы внутри блока loop) выполняется, пока истинно условие.</p>

1	2
Условная конструкция	<pre> when(условие) [инструкции1] otherwise [инструкции2] </pre> <p>Если условие истинно, выполняются инструкции, определенные в блоке when, если ложно — определенные в блоке otherwise. Также блок otherwise может быть опущен.</p>

1.15 Область видимости идентификаторов

В языке ZEI-2020 область видимости сверху вниз (по принципу C++). Передаваемые в функцию параметры видны только внутри функции; идентификаторов, объявленные внутри функций, доступны лишь внутри той же функции. Запрещено объявление одинаковых идентификаторов в одной области видимости.

Идентификаторы как библиотечных, так и определенных пользователем функций имеют глобальную видимость (GLB). Наличие переменных с глобальной областью видимости не предусмотрено.

1.16 Семантические проверки

В языке ZEI-2020 предусмотрены следующие семантические проверки:

- наличие функции `perform` – точки входа в программу;
- единственность точки входа в программу `perform`;
- переопределение идентификаторов;
- использование идентификаторов без их объявления;
- объявление функций вне других функций (в том числе `perform`);
- проверка соответствия типа функции и возвращаемого значения;
- ограничение количества максимально возможных параметров определяемой функции;
- корректность передаваемых в функцию параметров: количество, типы;
- выход за пределы диапазона числовых литералов или превышение размера строковых литералов;

- корректность составленного условия цикла или условного оператора;
- корректность выражений и арифметических операций (типы данных, допустимые операторы).

1.17 Стандартная библиотека и её состав

В языке ZEI-2020 существует возможность использовать стандартную библиотеку, которая автоматически подключается на этапе генерации кода. Её функции описаны в таблице 1.7.

Вызов стандартных функций доступен там же, где и вызов пользовательских функций.

Таблица 1.7 Стандартная библиотека и её состав.

Функция стандартной библиотеки	Описание	Тип возвращаемого значения
symbtotiny (<symbolic идентификатор / литерал >)	преобразование строки в число	tiny
symlen (<symbolic идентификатор>)	вычисление длины строки	tiny
gettime (<symbolic идентификатор / литерал >)	добавление к строке текущего времени	symbolic
genertiny()	получение случайного числа из диапазона [-128; 127]	tiny
generlogical()	получение случайного значения true или false	logical

1.18 Ввод и вывод данных

В языке ZEI-2020 поток ввода не предусмотрен.

Вывод данных осуществляется с помощью ключевого слова **show**, использование которого допускается с литералами и идентификаторами.

show(<идентификатор>|<литерал>)!

Примеры применения: show(x)! show("str")!

1.19 Точка входа

В языке ZEI-2020 может быть только одна точка входа, определяющаяся функцией perform. В случае её отсутствия или наличия более одного её

экземпляра будет сгенерировано и записано в протокол исключение. Работа транслятора будет прервана.

Синтаксис определения точки входа представлен в таблице 1.6.

1.20 Препроцессор

Препроцессоры в языке ZEИ-2020 не предусмотрены.

1.21 Объектный код

Язык ZEИ-2020 транслируется в язык С#.

1.22 Классификация сообщений транслятора

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Сообщения транслятора приведены в таблице 1.8.

Таблица 1.8 Классификация ошибок.

Номера ошибок	Характеристика	Префикс
0 – 199	Системные ошибки	[!]
300 – 399	Ошибки лексического анализа	[SA]
500 – 599	Ошибки синтаксического анализа	[SX]
600 – 699	Ошибки семантического анализа	[SM]
200-299, 400-499	Зарезервированные коды ошибок	

1.23 Контрольный пример

Контрольный пример, демонстрирующий главные особенности языка ZEИ-2020, представлен в приложении А.

2 Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

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

Основными компонентами транслятора ZEI-2020 являются лексический анализатор, синтаксический анализатор, семантический анализатор и генератор кода в язык C#. Они приведены на рисунке 2.1.

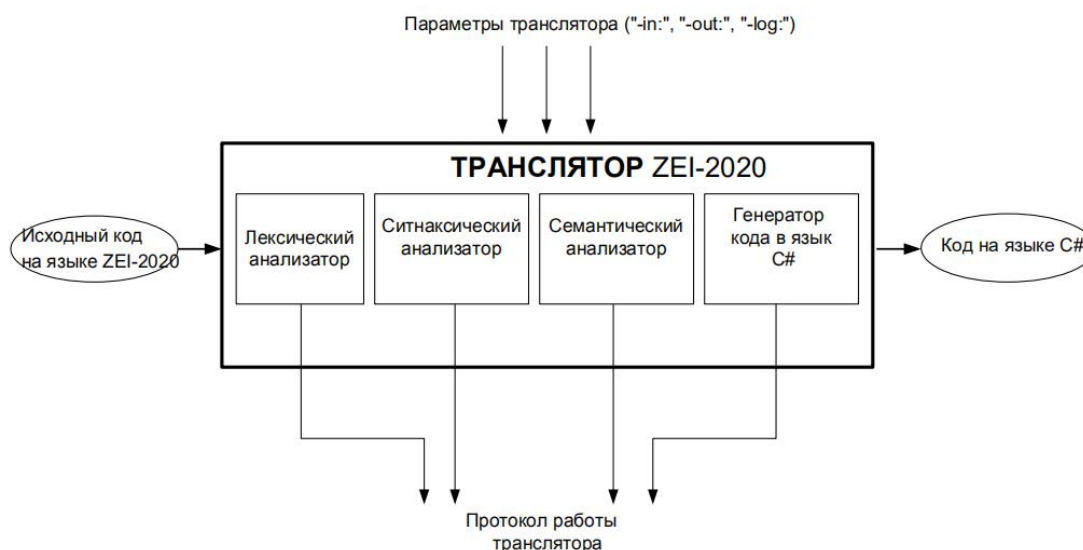


Рисунок 2.1 Схема структуры транслятора.

Таблица 2.1 Структура транслятора.

Компонент	Предназначение
1	2
Лексический анализатор	Обрабатывает входной файл исходного кода, проверяя допустимость его символов. Удаляет избыточные символы (пробелы, символы табуляции). Выделяет простейшие конструкции языка (лексические единицы). Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением — лексемами, для создания промежуточного представления исходной программы. Результатом работы лексического анализатора являются таблица лексем и таблица идентификаторов.

1	2
Синтаксический анализатор	Взаимодействует с лексическим анализатором последовательно (рисунок 2.2). Входом для синтаксического анализа является таблица лексем и таблица идентификаторов. Он проверяет правильность написанных конструкций, выявляет синтаксические ошибки. Результатом работы синтаксического анализатора является дерево разбора выражения.
Семантический анализатор	Семантический анализатор выявляет ошибки семантики (смысловые). Его входом являются таблица лексем и идентификаторов.
Генератор кода	Входом генератора кода является таблица лексем и таблица идентификаторов. На их основе он выполняет генерацию кода на языке C#.

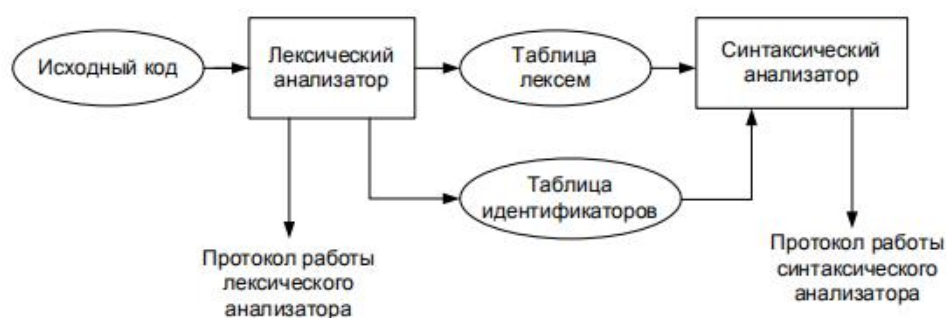


Рисунок 2.1 Последовательное взаимодействие лексического и синтаксического анализаторов.

2.2 Перечень входных параметров транслятора

Перечень входных параметров транслятора языка ZEI-2020 представлен в таблице 2.2.

Таблица 2.2 Входные параметры транслятора.

Входной параметр	Описание	Тип параметра
1	2	3
-in:	указывает транслятору путь к исходному коду, находящемуся в файле с расширением «.txt»;	обязательный;

1	2	3
-log:	указывает транслятору путь к файлу протокола;	не обязательный (при отсутствии явного указания путь к файлу протокола формируется конкатенацией имени файла исходного кода с постфиксом «.log»);
-out:	указывает транслятору выходной файл;	не обязательный (при отсутствии явного указания конкатенацией имени файла исходного кода с постфиксом «.cs»).

2.3 Перечень протоколов, формируемых транслятором и их содержимое

По итогам своей работы транслятор формирует один протокол, согласно заданным входным параметрам. -log: <путь к файлу>.

Информация, записываемая в протокол, представлена в таблице 2.3.

Таблица 2.3 Протокол транслятора.

Тип информации	Описание информации
дата и время	выводится дата и время создания протокола.
параметры командой строки	выводится информация об указанных параметрах командной строки.
информация об исходном коде	общее количество символов, количество проигнорированных символов, количество строк исходного файла.
таблица лексем	выводится таблица лексем.
таблица идентификаторов	выводится таблица идентификаторов, указывающая номер строки в таблице лексем, тип идентификатора, тип данных идентификатора, его имя, видимость, значение.
трассировочная информация синтаксического анализа	выводится полная информация о разборе таблицы лексем синтаксическим анализатором.
дерево разбора	в случае успешного разбора выводятся правила, по которым осуществился разбор исходного кода.

3 Разработка лексического анализатора

3.1 Структура лексического анализатора

Определение и назначение лексического анализатора было описано в пункте 2.1.

Структура лексического анализатора ZEI-2020 представлена на рисунке 3.1.



Рисунок 3.1 Структура лексического анализатора.

Исходный код на языке ZEI-2020 является входными данными лексического анализатор, выходными же являются таблицы лексем и идентификаторов.

3.2 Контроль входных символов

Таблица для контроля входных символов представлена на рисунке 3.2.

F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	/t*/I,	/r*/T,	F,	F,	F,	F,	F,	\
F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	\
/s */T,	/*!*/T,	/***/T,	/s#/T,	/*\$*/F,	/%*/F,	/*&*/F,	/**/T,	/*(*/T,	/*)*/T,	/***/T,	/*+*/T,	/*,**/T,	/*-*/T,	/*.*/T,	/*'*/T,	/*"/T,	\
/*0*/T,	/*1*/T,	/*2*/T,	/*3*/T,	/*4*/T,	/*5*/T,	/*6*/T,	/*7*/T,	/*8*/T,	/*9*/T,	/*:>/T,	/*;*/F,	/*<*/T,	/*=*/T,	/*>*/T,	/*?*/T,	/*%*/F,	\
/*@*/F,	/*A*/T,	/*B*/T,	/*C*/T,	/*D*/T,	/*E*/T,	/*F*/T,	/*G*/T,	/*H*/T,	/*I*/T,	/*J*/T,	/*K*/T,	/*L*/T,	/*M*/T,	/*N*/T,	/*O*/T,	/*P*/T,	\
/*Q*/T,	/*R*/T,	/*S*/T,	/*T*/T,	/*U*/T,	/*V*/T,	/*W*/T,	/*X*/T,	/*Y*/T,	/*Z*/T,	/*[*/T,	/**/F,	/*]*/T,	/*^*/T,	/*_*/F,	/*`*/T,	/*~*/F,	\
/**/F,	/*a*/T,	/*b*/T,	/*c*/T,	/*d*/T,	/*e*/T,	/*f*/T,	/*g*/T,	/*h*/T,	/*i*/T,	/*j*/T,	/*k*/T,	/*l*/T,	/*m*/T,	/*n*/T,	/*o*/T,	/*p*/T,	\
/*q*/T,	/*r*/T,	/*s*/T,	/*t*/T,	/*u*/T,	/*v*/T,	/*w*/T,	/*x*/T,	/*y*/T,	/*z*/T,	/*{*/T,	/* */F,	/*}*/T,	/*~*/F,				\
\																	
F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	\
F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	\
F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	\
F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	F,	\
/a*/F,	/*b*/F,	/*B*/F,	/*G*/F,	/*d*/F,	/*E*/F,	/*J*/F,	/*3*/F,	/*H*/F,	/*Й*/F,	/*K*/F,	/*Л*/F,	/*И*/F,	/*Н*/F,	/*О*/F,	/*П*/F,	/*Q*/F,	\
/*P*/F,	/*C*/F,	/*T*/F,	/*Y*/F,	/*o*/F,	/*X*/F,	/*Ц*/F,	/*ч*/F,	/*ш*/F,	/*Щ*/F,	/*Ъ*/F,	/*Ы*/F,	/*Ь*/F,	/*Э*/F,	/*Ю*/F,	/*Я*/F,	/*Z*/F,	\
/*a*/F,	/*6*/F,	/*b*/F,	/*r*/F,	/*d*/F,	/*e*/F,	/*ж*/F,	/*з*/F,	/*и*/F,	/*й*/F,	/*к*/F,	/*л*/F,	/*м*/F,	/*н*/F,	/*о*/F,	/*п*/F,	/*q*/F,	\
/*p*/F,	/*c*/F,	/*t*/F,	/*y*/F,	/*o*/F,	/*x*/F,	/*ц*/F,	/*ч*/F,	/*ш*/F,	/*щ*/F,	/*ъ*/F,	/*ы*/F,	/*ь*/F,	/*э*/F,	/*ю*/F,	/*я*/F,	/*z*/F,	\

Рисунок 3.2 Таблица допустимости входных символов.

Каждому элементу таблицы соответствует тот же код, что и в таблице кодировки Windows-1251 (рисунок 1.1).

В представленной таблице F — запрещённый символ, T — разрешённый символ, I — игнорируемый символ. Каждому символу из файла с исходным кодом

ставится в соответствие одно из этих значений. В случае запрещённого символа, при условии, что он встречается не в комментарии и не в строковом литерале, работа транслятора завершается, генерируется и записывается в протокол исключение. Символ, отмеченный как I, будет проигнорирован. Разрешенный символ будет допущен.

3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы. Они удаляются сразу после получения исходного текста программы.

Перед разбиением кода на лексемы запускается функция, которая удаляет повторяющиеся пробельные символы, а также все пробельные символы вокруг знаков операций (арифметических, логических, сдвигов) и вокруг символов сепараторов. При этом, если избыточный символ находится в строковом литерале, его удаление не происходит.

Примеры одного фрагмента кода до и после удаления избыточных символов представлены на рисунках 3.3 и 3.4.

```
set tiny a : x+y  + 22h!
show(a)!
set symbolic      str : "sp ace"!
show(str)!
set symbolic mystr : "а бв?гдежзикла  мопростуфхцч"!
```

Рисунок 3.3 Фрагмент кода до удаления избыточных символов.

```
set tiny a:x+y+22h!
show(a)!
set symbolic str:"sp ace"!
show(str)!
set symbolic mystr:"а бв?гдежзикла  мопростуфхцч"!
```

Рисунок 1.4 Фрагмент кода после удаления избыточных символов.

3.4 Перечень ключевых слов, сепараторов, символов операций, соответствующим им лексемам и конечных автоматов

Перечень ключевых слов и соответствующих им лексем представлен в таблице 3.1.

Таблица 3.1 Перечень ключевых слов и соответствующих им лексем.

Слово	Лексема	Пояснение
1	2	3
set	s	объявление переменной
tiny, symbolic, logical	t	типы данных

1	2	3
when	w	условный оператор, блок, выполняющийся при истинности условия
otherwise	o	условный оператор, блок, выполняющийся при ложности условия
show	h	вывод данных
function	f	определение функции
giveback	g	возвращение значения из функции
perform	p	точка входа в программу
loop	y	цикл
	i	идентификатор
	l	литерал
	b	библиотечная функция

Лексемы символов + , - * # = ^ < > [] () < > , : \ / соответствуют самим же символам.

Также присутствуют лексемы false и true для литерала типа logical.

Для всех лексем определены конечные автоматы, по которым происходит разбор выражения: на каждый автомат в массиве подаётся фраза и с помощью графа переходов происходит разбор. Если разбор выполнен, происходит заполнение таблицы лексем и, при необходимости, таблицы идентификаторов.

Благодаря замене цепочек, написанных на языке ZEI-202, лексемами, упрощается дальнейшая обработка исходного кода программы. Перечень выражений представлен в приложении Б.

3.5 Основные структуры данных

Описание основных структур данных, используемых для хранения таблиц лексем, представлено на рисунке 3.5.

```

struct Entry//строка таблицы лексем
{
    char lexema;
    int sn;//номер строки в тексте
    int indxTI;//индекс в таблице идентификаторов или TI_NULLIDX
    char info[30]; //информация о лексеме
};

struct LexTable//экземпляр таблицы лексем
{
    int maxsize;//емкость таблицы лексем( < LT_MAXSIZE)
    int size;//текущий размер таблицы лексем ( < LT_MAXSIZE)
    Entry* table;//массив строк таблицы лексем
};

```

Рисунок 3.5 Структуры для таблицы лексем ZEI-2020.

Описание основных структур данных, используемых для хранения таблиц идентификаторов, представлено в приложении В.

3.6 Принцип обработки ошибок

При обнаружении ошибки на этапе лексического анализа немедленно генерируется исключение, содержащее следующую информацию: код ошибки, номер строки в коде номер столбца в коде. При возникновении ошибки работа транслятора прекращается, а в протокол заносится информация об ошибке.

3.7 Структура и перечень сообщений лексического анализатора

Сообщения лексического анализа помечены префиксом [LA]. Их перечень представлен в таблице 3.2.

Таблица 3.2 Перечень сообщений лексического анализатора.

Код	Сообщение
1	2
300	Превышен максимальный размер таблицы лексем
301	Таблица лексем переполнена
302	Таблица лексем, вероятно, ещё не создана
303	Недопустимый номер строки таблицы лексем
304	Превышен максимальный размер таблицы идентификаторов
305	Таблица идентификаторов переполнена
306	Таблица идентификаторов, вероятно, ещё не создана

1	2
307	Недопустимый номер строки таблицы идентификаторов
308	Превышена максимальная длина имени идентификатора
309	Значение вне диапазона для литерала типа tiny [-128; +127]
310	Превышена максимальная длина литерала типа symbolic
311	Ошибка лексического разбора

3.8 Параметры лексического анализатора и режимы его работы

Исходный текст на языке ZEI-2020 подается на вход. Параметры, определяющие режим работы лексического анализатора, не предусмотрены.

3.9 Алгоритм лексического анализа

Лексический анализ является первой и наиболее простой фазой трансляции. Алгоритм лексического анализатора заключается в следующем: после разбиения текста из файла с исходным кодом на слова, для каждого слова подбирается конечный автомат, способный его разобрать, в случае, если такой автомат существует, цепочка будет разобрана, иначе будет сгенерировано исключение. Далее сканер анализирует лексему, соответствующую данному слову, и выполняет действия, описанные для данной лексемы. Лексический анализатор продолжает работать, пока не будет разобрано последнее слово.

Работу конечных автоматов можно представить в виде графа. Пример графа представлен на рисунке 3.5. В виде кода представлен на рисунке 3.6. На рисунке 3.5 осуществляется разбор цепочки “func”, где S0 — начальное состояние, а S4 — конечное.

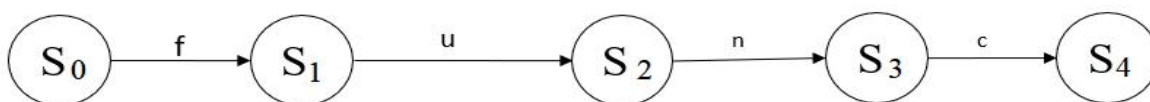


Рисунок 3.5 Граф переходов для цепочки “func”, графическое представление.

```

#define FST_FUNC 5, \
    FST::NODE(1, FST::RELATION('f', 1)), \
    FST::NODE(1, FST::RELATION('u', 2)), \
    FST::NODE(1, FST::RELATION('n', 3)), \
    FST::NODE(1, FST::RELATION('c', 4)), \
    FST::NODE()
  
```

Рисунок 3.6 Граф переходов для цепочки “func”.

3.10 Контрольный пример

Результатом работы лексического анализатора являются таблица лексем и таблица идентификаторов. Содержимое таблиц на основе исходного кода из приложения А представлено в приложении Г.

4 Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 Структура синтаксического анализатора ZEI-2020.

Таблицы лексем и идентификаторов являются входными данными, а дерево разбора, выводимое в файл протокола, является выходными данными синтаксического анализатора.

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

Грамматика для синтаксического разбора языка ZEI-2020 представляется четверкой $G = \langle T, N, P, S \rangle$, где T — множество терминальных символов, N — множество нетерминальных символов, P — множество правил языка, S — начальный символ грамматики, являющийся нетерминалом.

В грамматике языка ZEI-2020 множество нетерминальных символов представлено следующим образом:

- S порождает правила, описывающие общую структуру программы;
- N порождает правила, описывающие основные конструкции языка;
- E порождает правила, описывающие выражения;
- F порождает правила, описывающие список формальных параметров функций;
- P порождает правила, описывающие формальные параметры функции;
- M порождает правила, описывающие арифметические действия;
- H порождает правила, описывающие сдвиговые операции;
- K порождает правила, описывающие вызов функции;
- W порождает правила, описывающие фактические параметры функции;
- B порождает правила, описывающие структуру тела функции;
- I порождает правила, описывающие литерал или идентификатор;
- R порождает правила, описывающие конструкции, допустимые условия циклов или условных конструкций;
- X порождает правила, описывающие конструкции, допустимые в теле циклов и условных операторов;

Перечень нетерминалов и порождаемых ими цепочек правил представлен в таблице 4.1.

Таблица 4.1 Правила грамматики ZEI-2020.

Нетерминальный символ	Цепочки правил
S	tfiFBS p[N]
F	() (P)
P	ti, P ti
B	[gI!] [NgI!]
I	l i
N	sti:E!N sti!N i:E!N i:H!N y(R)[X]N h(I)!N w(R)[X]N w(R)[X]o[X]N bK!N iK!N sti:E! sti! i:E! i:H! y(R)[X] h(I)! w(R)[X] w(R)[X]o[X] bK! iK!
R	i<I l<I i>I l>I i=I l=I i^I l^I
H	i/I i\I l/I l\I
E	bK i l (E) iK iM lM (E)M iKM bKM
K	() (W)
W	i l i, W l, W
M	+EM -EM *EM #EM +E *E #E -E
X	i:I!X i:E!X i:H!X h(I)!X bK!X iK!X i:I! i:E! i:H!X h(I)! bK! iK!

4.3 Построение конечного магазинного автомата

Принцип действия конечного магазинного автомата представлен на рисунке 4.2.

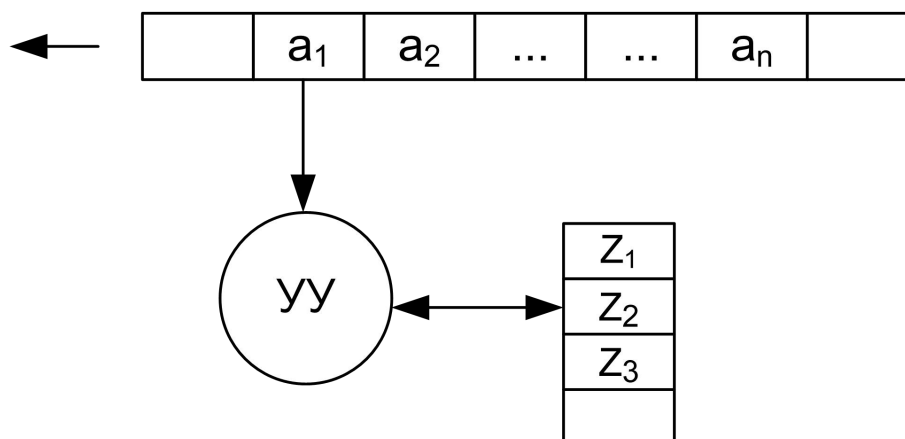


Рисунок 4.2 Схема автомата с магазинной памятью.

Формальное описание МП-автомата:

$$M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$$

Q - множество состояний;

V - алфавит входных символов;

Z - специальный алфавит магазинных символов;

δ - функция переходов автомата $Q \times (V \cup \{\lambda\}) \times Z \rightarrow P(Q \times Z^*)$, где

$P(Q \times Z^*)$ - множество подмножеств $Q \times Z^*$;

$q_0 \in Q$ - начальное состояние автомата;

$z_0 \in Z$ - начальное состояние магазина (маркер дна);

$F \subseteq Q$ - множество конечных состояний.

Конфигурация (текущее состояние автомата) описывается тройкой (q, α, ω) , где q - текущее состояние автомата, α - остаток цепочки, ω - цепочка-содержимое магазина.

Начальное состояние (q_0, α, z_0) , q_0 - начальное состояние автомата, α - входная цепочка, z_0 - маркер дна магазина.

Цепочка α является допустимой (распознается) автоматом $M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$, если $(q_0, \alpha, z_0) \succ^* (q', \lambda, \lambda)$ и $q' \in F$.

Работа автомата $M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$

- 1) состояние автомата $(q, a\alpha, z\beta)$
- 2) читает символ a находящийся под головкой (сдвигает ленту);
- 3) не читает ничего (читает λ , не сдвигает ленту);
- 4) из δ определяет новое состояние q' , если $(q', \gamma) \in \delta(q, a, z)$ или $(q', \gamma) \in \delta(q, \lambda, z)$.
- 5) читает верхний (в стеке) символ z и записывает цепочку γ т.к. $(q', \gamma) \in \delta(q, a, z)$, при этом, если $\gamma = \lambda$, то верхний символ магазина просто удаляется.

работа автомата заканчивается (q, λ, λ) .

4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающую правила языка ZEI-2020. Правила грамматики описаны в приложении Д.

4.5 Описание алгоритма синтаксического разбора

Входные символы и лексемы в форме Грейбах находятся в ленте на входе конечного автомата.

- 1) Если лента не пустая, переходим далее следующему пункту, иначе переходим к пункту 5.
- 2) Если на верхушке магазина нетерминальный символ.
 - 2.1) Если есть такое правило, то переходим к следующему пункту.
 - 2.1.1) Если цепочка есть, возвращаем NS_OK. Переходим к пункту 4.
 - 2.1.2) Иначе восстанавливаем состояние. Переходим к пункту 4.
 - 2.2) Иначе возвращаем ошибку. Переход к пункту 4.
- 3) Если на верхушке терминал и он совпадает с символом на ленте, то удаляем его из стека и продвигаем ленту. Переход к пункту 4.
- 4) Повторяем шаг, переходим к пункту 1.
- 5) Конец работы.

4.6 Структура и перечень сообщений синтаксического анализатора

Сообщения синтаксического анализатора отмечены префиксом [SX], и их перечень приведен в таблице 4.2.

Таблица 4.2 Перечень сообщений синтаксического анализатора.

Код ошибки	Сообщение
500	Неверная структура программы
501	Неверный список параметров функции
502	Неверный список параметров функции при её объявлении
503	Отсутствует тело функции
504	Неверное выражение. Ожидаются только идентификаторы и литералы
506	Неверная конструкция в теле функции
507	Ошибка в условном выражении
508	Ошибка в вызове функции
509	Ошибка в параметрах вызываемой функции
510	Ошибка в арифметических или сдвиговых операциях
511	Неверная конструкция в теле цикла / условного выражения

4.7 Параметры синтаксического анализатора и режимы его работы

Входных параметров для синтаксического анализатора, определяющих режим его работы, не предусмотрено.

4.8 Принцип обработки ошибок

Принцип обработки ошибок заключается в том, что синтаксический анализатор перебирает все возможные правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем. В

случае, если не была найдена ни одна подходящая цепочка, формируется соответствующая ошибка из таблицы 4.2. Все ошибки записываются в общую структуру ошибок (в которой запоминается до 3 ошибок), выводятся в файл протокола и отображаются на консоли.

4.9 Контрольный пример

Результатом работы синтаксического анализатора для контрольного примера, представленного на рисунке 4.3, являются трассировка, представленная в приложении Е и дерево разбора выражения, представленное на рисунке 4.4 и в графическом материале.

```

tiny func boo()
[
  show("boo function")!           ~комментарий
  giveback 19!
]
perform
[
  set tiny x : m100!
  set symbolic      str : "sp ace"!
  set logical mb : true!
  set tiny convert : 5 + symbtotiny(str)!
  when (mb ^ false)
  [
    show(x)!
  ]
]

```

Рисунок 4.3 Контрольный пример для синтаксического анализа.

Всего строк: 60, разбор завершен без ошибок

0	S->tfiFBS
3	F->O
5	B->[NgI!]
6	N->h(I)!
8	I->l
12	I->l
15	S->p[N]
17	N->sti:E!N
21	E->l
23	N->sti:E!N
27	E->l
29	N->sti:E!N
33	E->l
35	N->sti:E!N
39	E->IM
40	M->+E
41	E->bK
42	K->(W)
43	W->i
46	N->w(R)[X]
48	R->i^I
50	I->l
53	X->h(i)!

Рисунок 4.4 Дерево разбора для контрольного примера с рисунка 4.3.

5 Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализ в трансляторе языка ZEI-2020 выделен в отдельную фазу, а также частично реализован на этапе лексического анализа. Состоит из набора инструкций для проверки смысловой корректности исходной кода.

Структура семантического анализатора представлена на рисунке 2.1.

5.2 Функции семантического анализа

Семантические проверки языка ZEI-2020 представлены в главе 1, пункте 1.16. Из них на этапе лексического анализа выполняются следующие:

- проверка наличия и единственности точки входа в программу perform;
- проверка на необъявленный идентификатор;
- проверка переобъявления идентификатора;

Остальные семантические проверки реализованы функцией Analysis из пространства имен Sem.

5.3 Структура и перечень сообщений семантического анализатора

Сообщения синтаксического анализатора отмечены префиксом [SM].

Сообщения, формируемые семантическим анализатором представлены в таблице 5.1.

Таблица 5.1 Перечень сообщений семантического анализатора.

Код ошибки	Сообщение
1	2
602	Отсутствует точка входа в программу perform
603	Присутствует более одной точки входа в программу perform
604	Идентификатор не объявлен
605	Невозможно вызвать функцию с объявляемым идентификатором
606	Идентификатор уже определен
607	Неверное место для объявления функции! Должна быть определена до точки входа в программу perform
608	Несовпадение типа функции и типа возвращаемого значения
609	Превышено максимальное значение в параметрах определяемой функции (3)
610	Неверные параметры вызываемой функции
611	Слишком много аргументов в вызове функции
612	Слишком мало аргументов в вызове функции

1	2
613	Типы данных в выражении не совпадают
614	Арифметические операции можно производить только над типом tiny
615	Логические операции < > можно производить только над типом tiny
616	Логические операции = ^ можно производить только над типами tiny или logical
617	Неверное выражение. Несовпадение типов слева и справа от знака присваивания

5.4 Принцип обработки ошибок

Все семантические ошибки являются критическими, и при генерации одной из них генерируется исключение, транслятор прекращает работу, и в протокол работы выводится соответствующее сообщение об ошибке.

5.5 Контрольный пример

Контрольный пример заключается в тестировании семантического анализатора при наличии соответствующих ошибок в исходном коде. Тестирование представлено в таблице 5.3.

Таблица 5.3 Тестирование.

Исходный код с ошибкой	Генерируемое сообщение об ошибке
1	2
perform [] perform [Ошибка 603: [SM] Присутствует более одной точки входа в программу perform
perform [show(x)!	Ошибка 604: [SM] Идентификатор не объявлен строка 15 позиция 6
perform [set tiny x : m100! set tiny x : 100!	Ошибка 606: [SM] Идентификатор уже определен строка 16 позиция 10

1	2
perform [set symbolic x : m100!	Ошибка 613: [SM] Типы данных в выражении не совпадают строка 15 позиция 0
tiny func boo() [show("boo function")! giveback 19!] perform [set tiny x : m100! boo(x)!	Ошибка 611: [SM] Слишком много аргументов в вызове функции строка 16 позиция 0
set logical mb : true! set tiny convert: sybmtotiny(mb)!	Ошибка 610: [SM] Неверные параметры вызываемой функции строка 29 позиция 0

6 Вычисление выражений

6.1 Выражения, допускаемые языком

В языке ZEI-2020 допускаются выражения с использованием числовых идентификаторов и литералов, а также вызовом функций. Предусмотрены операции, описанные в пункте 1.12.

Примеры допускаемых языком выражений:

- set tiny fef : (x -12)# (a +y)!
- set tiny a : x+y + 22h!
- convert : 2 + x + symbtotiny(str)!
- mb ^ false!
- y : x \ 2!

7 Генерация кода

7.1 Структура генератора кода

Генерация кода — это перевод транслятором представления исходной программы на языке ZEI-2020 в цепочку символов выходного языка C#. На вход генератора подаются таблицы лексем и идентификаторов на основе которых генерируется файл с кодом на языке C#.

Схематично генерация кода показана на рисунке 7.1.

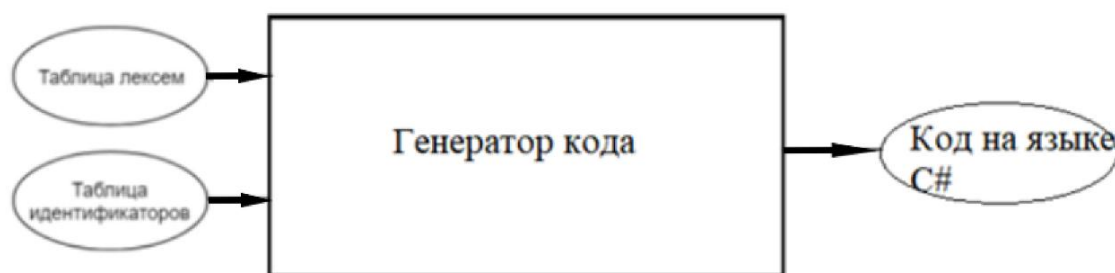


Рисунок 7.1 Структура генератора кода.

7.2 Представление типов данных в оперативной памяти

Язык ZEI-2020 требует указывать тип данных при объявлении идентификатора с помощью ключевого слова `set`. Соответствия между типами данных идентификаторов языка ZEI-2020 и языка C# представлены в таблице 7.1.

Таблица 7.1 Соответствие типов данных.

Тип данных на языке ZEI-2020	Тип данных на языке C#
<code>tiny</code> (целочисленный, 1 байт)	<code>sbyte</code> (целочисленный, 1 байт)
<code>symbolic</code> (строковый)	<code>string</code> (строковый)
<code>logical</code> (логический)	<code>bool</code> (логический)

7.3 Библиотека

В языке ZEI-2020 предусмотрена стандартная библиотека, представленная файлом `StandartLibrary.cs`. Она содержит функции, написанные на языке C#, которые описаны в таблице 1.9.

Объявление функций библиотеки происходит автоматически.

Вызовы стандартных функций доступны там же, где и вызов пользовательских функций.

Фрагмент генератора кода, обеспечивающий работу библиотечных функций, приведен на рисунке 7.2.

```

case LEX_LIBFUNC: {
    *(out.stream) << "ZEI2020stdlib.StandardLibrary.";
    if (!strcmp(idtable.table[lextable.table[i].indxTI].id, "symlen"))
    {
        *(out.stream) << "SymbLen";
    }
    if (!strcmp(idtable.table[lextable.table[i].indxTI].id, "symltotiny"))
    {
        *(out.stream) << "SymbToTiny";
    }
    if (!strcmp(idtable.table[lextable.table[i].indxTI].id, "gettime"))
    {
        *(out.stream) << "GetTime";
    }
    if (!strcmp(idtable.table[lextable.table[i].indxTI].id, "genertiny"))
    {
        *(out.stream) << "GenerTiny";
    }
    if (!strcmp(idtable.table[lextable.table[i].indxTI].id, "generlgcl"))
    {
        *(out.stream) << "GenerLogical";
    }
    break;
}

```

Рисунок 7.2 Фрагмент генератора кода, обеспечивающий работу библиотечных функций.

7.4 Особенности алгоритма генерации кода

Одной из особенностей генерации кода является добавление явного приведения к типу `sbyte` в коде на языке `C#` в те инструкции, где находятся выражения типа `tiny`, применяющие арифметические операции, операции сдвига или в места возвращения из функции значения типа `tiny`.

```

case LEX_ASSIGN: {
    //в присваивании литерала не надо приведение типов
    *(out.stream) << " = ";
    if (idtable.table[lextable.table[i - 1].indxTI].iddatatype == IT::TINY && lextable.table[i+2].lexema != LEX_EXCLAMATION) {
        typeflag = true;
        *(out.stream) << "(sbyte)(";
    }
    break;
}
case LEX_GIVEBACK: {
    *(out.stream) << "return ";
    if (idtable.table[lextable.table[i + 1].indxTI].iddatatype == IT::TINY) {
        typeflag = true;
        *(out.stream) << "(sbyte)(";
    }
    break;
}

```

Рисунок 7.3 Фрагмент генератора кода, добавляющего явное приведение типов.

Такое приведение типов необходимо потому, что в языке `C#` нельзя применять многие операции (в том числе арифметические) к `sbyte` типу. Операции выполняются над типом `System.Int32`, а затем приводятся к необходимому однобайтовому типу.

Примеры подобных приведений представлены в таблице 7.2.

Таблица 7.2 Соответствие выражений при приведении типов.

Выражение на языке ZEI-2020	Выражение на языке C#
giveback 19!	return (sbyte)(19);
y : x \ 2!	y = (sbyte)(x << 2);
set tiny a : x+y + 22h!	sbyte a = (sbyte)(x + y + 34);
set tiny convert : 2 + x + symbtotiny(str)!	sbyte convert = (sbyte)(2 + x + ZEI2020stdlib.StandardLibrary.SymbToTiny(str));
set tiny fef : (x -12) # (a +y)!	sbyte fef = (sbyte)((x - 12) / (a + y));

Также на этапе генерации кода осуществляется инициализация переменных по умолчанию, описанная в подразделе 1.5.

Еще одной особенностью является генерация отступов в исходном коде на языке C# для лучшего представления структуры программы.

Сразу после успешной генерации кода есть возможность вызвать компилятор C# (csc.exe) непосредственно из транслятора ZEI-2020. В передаваемые ему параметры входит путь к файлу, заданный ключом -out при вызове транслятора. Компилятор csc.exe формирует исполнимый файл, а затем он также вызывается непосредственно из транслятора ZEI-2020.

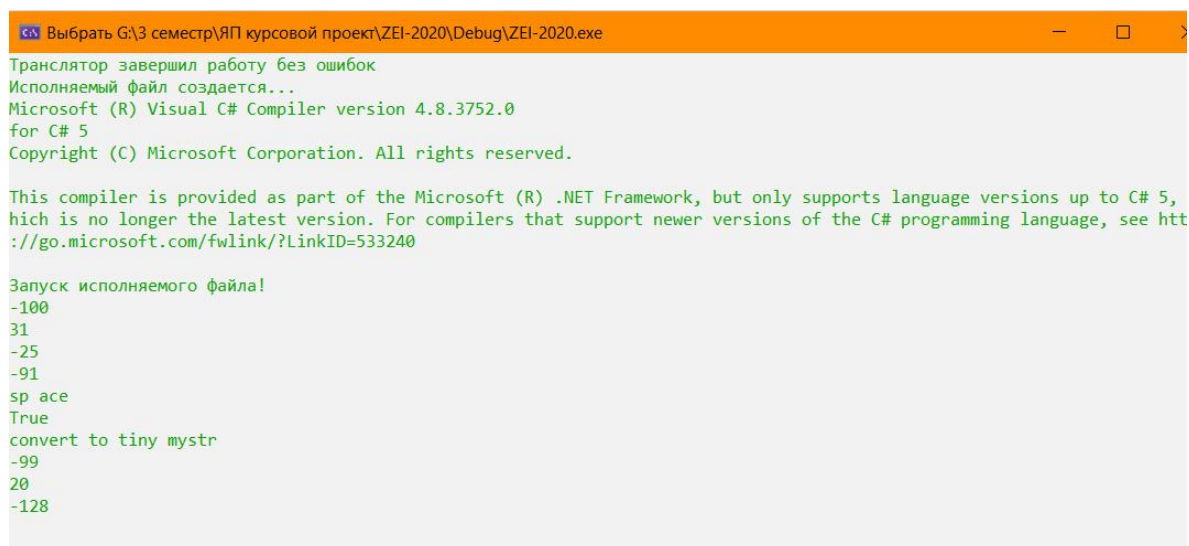
```

#ifdef AUTO
    size_t n = 0;
    char path[PARM_MAX_SIZE];
    char cs[PARM_MAX_SIZE];
    char csc[] = "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\csc.exe";
    char exe[] = "..\\Debug\\OutputFile.exe";
    char stdlib[] = "..\\ZEI2020stdlib\\StandartLibrary.cs";
    wcstombs_s(&n, cs, parm.out, PARM_MAX_SIZE);

    sprintf_s(path, PARM_MAX_SIZE, "%s -out:\"%s\" \"%s\" \"%s\" -warn:0", csc, exe, cs, stdlib);
    std::cout << "Исполняемый файл создается..." << std::endl;
    system(path);
    std::cout << "Запуск исполняемого файла!" << std::endl;
    system(exe);
#endif

```

Рисунок 7.2 Реализация запуска программы на языке C# из транслятора ZEI-2020.



```

Выбрать G:\3 семестр\ЯП курсовой проект\ZEI-2020\Debug\ZEI-2020.exe
Транслятор завершил работу без ошибок
Исполняемый файл создается...
Microsoft (R) Visual C# Compiler version 4.8.3752.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports language versions up to C# 5,
which is no longer the latest version. For compilers that support newer versions of the C# programming language, see http
://go.microsoft.com/fwlink/?LinkID=533240

Запуск исполняемого файла!
-100
31
-25
-91
sp ace
True
convert to tiny mystr
-99
20
-128

```

Рисунок 7.3 Демонстрация запуска программы на языке C# из транслятора ZEI-2020.

7.5 Входные параметры генератора кода

На этапе генерации кода входные параметры транслятора не предусмотрены.

7.6 Контрольный пример

Результат генерации кода на основе контрольного примера из приложения А приведен в приложении Ж.

8 Тестирование транслятора

8.1 Общие положения

В результате обработки исходного кода программы, представленного в приложении А, транслятор языка ZEI-2020 генерирует общий протокол работы, куда записываются все возникшие ошибки. Также они выводятся и на консоль.

8.2 Результаты тестирования

Транслятор языка ZEI-2020 представляет диагностику и выявление ошибок на разных этапах трансляции.

Тестирование ошибок транслятора представлено в приложении И.

Заключение

В данном курсовом проекте были выполнены поставленные минимальные требования. Были успешно достигнуты поставленные цели, представленные во введении.

В итоге был разработан язык программирования ZEI-2020 и транслятор к нему.

Окончательная версия языка ZEI-2020 включает:

- 3 типа данных;
- поддержка операции вывода;
- возможность вызова функций стандартной библиотеки;
- 4 арифметических оператора для вычисления выражений;
- 4 логических оператора;
- 2 оператора сдвига;
- структурированная система для обработки ошибок пользователя.

Основные характеристики транслятора ZEI-2020:

- Возможность обработки 3 входных параметров;
- Возможность обработки 49 ошибок;
- Реализация 39 конечных автоматов;
- Реализация 78 цепочек правил грамматики;
- Наличие порядка 2900 строк кода.

Список использованных источников

1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – М.: Вильямс, 2003. – 768с.
2. Смелов, В.В. Курс лекций по предмету языки программирования – 2016
3. Прата, С. Язык программирования C++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 с.
4. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с.

Приложение А

Контрольный пример на языке ZEI-2020

```

symbolic func      foo(symbolic str, tiny k, tiny m)
[
  show("symlen")!
  set tiny p : symlen(str)!
  show(str)!
  giveback str!
]
tiny func  boo()
[
  show("boo function")!           ~комментарий
  giveback 29!
]
perform
[
  set tiny x : m100!
  show(x)!
  set tiny y!
  y : x / 2!
  y: genertiny() + 2!
  show(y)!
  set tiny a : x+y  + 22h!
  show(a)!
  set symbolic      str: "sp ace"!
  show(str)!
  set symbolic mystr : "а бв?гдежзикла  мопростуфхцч"!
  set symbolic w : gettime(str)!
  show(w)!
  set logical mb : true!
  set logical k!
  set logical mbd : generlgcl()!
  show(mb)!
  set tiny convert : symbtotiny(str) + 1!
  show(convert)!
  when (mb ^ false)
  [
    a: 19+1!
    show(a)!
  ]
  otherwise
  [
    show(str)!
  ]
  set tiny fef : (x - 12)  # (a + y)!
  loop(fef > 5)
  [
    fef : fef + 1!
  ]
  show(fef)!
]

```

Рисунок 1 Контрольный пример.

Приложение Б

Графы переходов конечных автоматов

```
#pragma region TYPESOFDATA
#define FST_TINY 5, \
    FST::NODE(1, FST::RELATION('t', 1)), \
    FST::NODE(1, FST::RELATION('i', 2)), \
    FST::NODE(1, FST::RELATION('n', 3)), \
    FST::NODE(1, FST::RELATION('y', 4)), \
    FST::NODE()

#define FST_SYMBOLIC 9, \
    FST::NODE(1, FST::RELATION('s', 1)), \
    FST::NODE(1, FST::RELATION('y', 2)), \
    FST::NODE(1, FST::RELATION('m', 3)), \
    FST::NODE(1, FST::RELATION('b', 4)), \
    FST::NODE(1, FST::RELATION('o', 5)), \
    FST::NODE(1, FST::RELATION('l', 6)), \
    FST::NODE(1, FST::RELATION('i', 7)), \
    FST::NODE(1, FST::RELATION('c', 8)), \
    FST::NODE()

#define FST_LOGICAL 8, \
    FST::NODE(1, FST::RELATION('l', 1)), \
    FST::NODE(1, FST::RELATION('o', 2)), \
    FST::NODE(1, FST::RELATION('g', 3)), \
    FST::NODE(1, FST::RELATION('i', 4)), \
    FST::NODE(1, FST::RELATION('c', 5)), \
    FST::NODE(1, FST::RELATION('a', 6)), \
    FST::NODE(1, FST::RELATION('l', 7)), \
    FST::NODE()
#pragma endregion

#define FST_FUNC 5, \
    FST::NODE(1, FST::RELATION('f', 1)), \
    FST::NODE(1, FST::RELATION('u', 2)), \
    FST::NODE(1, FST::RELATION('n', 3)), \
    FST::NODE(1, FST::RELATION('c', 4)), \
    FST::NODE()

#define FST_PERFORM 8, \
    FST::NODE(1, FST::RELATION('p', 1)), \
    FST::NODE(1, FST::RELATION('e', 2)), \
    FST::NODE(1, FST::RELATION('r', 3)), \
    FST::NODE(1, FST::RELATION('f', 4)), \
    FST::NODE(1, FST::RELATION('o', 5)), \
    FST::NODE(1, FST::RELATION('r', 6)), \
    FST::NODE(1, FST::RELATION('m', 7)), \
    FST::NODE()
```

Рисунок 2 Графы для типов данных, функции и точки входа.

Продолжение приложения Б

```
#pragma region LITERAL

#define FST_TINYLITERAL16 3, \
    FST::NODE(32, \
        FST::RELATION('1', 0), FST::RELATION('2', 0), FST::RELATION('3', 0), FST::RELATION('4', 0), FST::RELATION('5', 0), FST::RELATION('6', 0), \
        FST::RELATION('7', 0), FST::RELATION('8', 0), FST::RELATION('9', 0), FST::RELATION('A', 0), FST::RELATION('B', 0), FST::RELATION('C', 0), \
        FST::RELATION('D', 0), FST::RELATION('E', 0), FST::RELATION('F', 0), FST::RELATION('0', 0), \
        \
        FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1), FST::RELATION('6', 1), \
        FST::RELATION('7', 1), FST::RELATION('8', 1), FST::RELATION('9', 1), FST::RELATION('A', 1), FST::RELATION('B', 1), FST::RELATION('C', 1), \
        FST::RELATION('D', 1), FST::RELATION('E', 1), FST::RELATION('F', 1), FST::RELATION('0', 1)), \
        FST::NODE(1, FST::RELATION('h', 2)), \
        FST::NODE())

#define FST_TINYLITERAL2 3, \
    FST::NODE(4, \
        FST::RELATION('0', 0), FST::RELATION('1', 0), \
        \
        FST::RELATION('0', 1), FST::RELATION('1', 1)), \
        FST::NODE(1, FST::RELATION('b', 2)), \
        FST::NODE())

#define FST_TINYLITERAL10 2, \
    FST::NODE(21, \
        FST::RELATION('m', 0), FST::RELATION('1', 0), FST::RELATION('2', 0), FST::RELATION('3', 0), FST::RELATION('4', 0), FST::RELATION('5', 0), FST::RELATION('6', 0), \
        FST::RELATION('7', 0), FST::RELATION('8', 0), FST::RELATION('9', 0), FST::RELATION('0', 0), \
        \
        FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1), FST::RELATION('6', 1), \
        FST::RELATION('7', 1), FST::RELATION('8', 1), FST::RELATION('9', 1), FST::RELATION('0', 1)), \
        FST::NODE())

#define FST_TINYLITERAL8 3, \
    FST::NODE(16, \
        FST::RELATION('1', 0), FST::RELATION('2', 0), FST::RELATION('3', 0), FST::RELATION('4', 0), FST::RELATION('5', 0), FST::RELATION('6', 0), \
        FST::RELATION('7', 0), FST::RELATION('0', 0), \
        \
        FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1), FST::RELATION('6', 1), \
        FST::RELATION('7', 1), FST::RELATION('0', 1)), \
        FST::NODE(1, FST::RELATION('q', 2)), \
        FST::NODE())

#define FST_SYMBOLICLITERAL 4, \
    FST::NODE(1, FST::RELATION(' ', 1)), \
    FST::NODE(156, \
        FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1), \
        FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1), \
        FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1), \
        FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1), \
        FST::RELATION('y', 1), FST::RELATION('z', 1), FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), FST::RELATION('4', 1), \
        FST::RELATION('5', 1), FST::RELATION('6', 1), FST::RELATION('7', 1), FST::RELATION('8', 1), FST::RELATION('9', 1), FST::RELATION('0', 1), \
        \
        FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1), \
        FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1), \
        FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1), \
        FST::RELATION('c', 1), FST::RELATION('t', 1), FST::RELATION('y', 1), FST::RELATION('f', 1), FST::RELATION('x', 1), FST::RELATION('u', 1), \
        FST::RELATION('u', 1), FST::RELATION('w', 1), FST::RELATION('b', 1), FST::RELATION('w', 1), FST::RELATION('b', 1), \
        FST::RELATION('a', 1), FST::RELATION('u', 1), FST::RELATION('j', 1), FST::RELATION(' ', 1), FST::RELATION('.', 1), FST::RELATION(',', 1), \
        FST::RELATION('?', 1), FST::RELATION('!', 1), FST::RELATION(';'), FST::RELATION(':', 1), FST::RELATION('-', 1), FST::RELATION('\t', 1), \
        \
        FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2), \
        FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2), \
        FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2), \
        FST::RELATION('s', 2), FST::RELATION('t', 2), FST::RELATION('u', 2), FST::RELATION('v', 2), FST::RELATION('w', 2), FST::RELATION('x', 2), \
        FST::RELATION('y', 2), FST::RELATION('z', 2), FST::RELATION('1', 2), FST::RELATION('2', 2), FST::RELATION('3', 2), FST::RELATION('4', 2), \
        FST::RELATION('5', 2), FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2), FST::RELATION('9', 2), FST::RELATION('0', 2), \
        \
        FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2), \
        FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2), \
        FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2), \
        FST::RELATION('c', 2), FST::RELATION('t', 2), FST::RELATION('y', 2), FST::RELATION('f', 2), FST::RELATION('x', 2), FST::RELATION('u', 2), \
        FST::RELATION('u', 2), FST::RELATION('w', 2), FST::RELATION('b', 2), FST::RELATION('w', 2), FST::RELATION('b', 2), \
        FST::RELATION('a', 2), FST::RELATION('u', 2), FST::RELATION('j', 2), FST::RELATION(' ', 2), FST::RELATION('.', 2), FST::RELATION(',', 2), \
        FST::RELATION('?', 2), FST::RELATION('!', 2), FST::RELATION(';'), FST::RELATION(':', 2), FST::RELATION('-', 2), FST::RELATION('\t', 2)), \
        FST::NODE(1, FST::RELATION(' ', 3)), \
        FST::NODE())
```

Рисунок 3 Графы для литералов.

Продолжение приложения Б

```

#pragma region ARIFM
#define FST_PLUS 2, \
    FST::NODE(1, FST::RELATION('+', 1)),\
    FST::NODE()

#define FST_MINUS 2, \
    FST::NODE(1, FST::RELATION('-', 1)),\
    FST::NODE()

#define FST_STAR 2, \
    FST::NODE(1, FST::RELATION('*', 1)),\
    FST::NODE()

#define FST_DIVISION 2, \
    FST::NODE(1, FST::RELATION('/', 1)),\
    FST::NODE()
#pragma endregion

#pragma region SHIFTS
#define FST_LEFTSHIFT 2, \
    FST::NODE(1, FST::RELATION('<<', 1)),\
    FST::NODE()

#define FST_RIGHTSHIFT 2, \
    FST::NODE(1, FST::RELATION('>>', 1)),\
    FST::NODE()
#pragma endregion

#pragma region LOGICAL
#define FST_EQUALITY 2, \
    FST::NODE(1, FST::RELATION('=', 1)),\
    FST::NODE()

#define FST_LESS 2, \
    FST::NODE(1, FST::RELATION('<', 1)),\
    FST::NODE()

#define FST_MORE 2, \
    FST::NODE(1, FST::RELATION('>', 1)),\
    FST::NODE()

#define FST_INEQUALITY 2, \
    FST::NODE(1, FST::RELATION('^=', 1)),\
    FST::NODE()
#pragma endregion

```

Рисунок 4 Графы арифметических и логических операторов, операторов сдвига.

Приложение В

Структуры для представления таблицы идентификаторов

```
enum IDDATATYPE
{
    TINY = 1, SYMB, LGCL, UNDEF //типы данных идентификаторов
};

enum IDTYPE
{
    V = 1, L, F, B, P, U //тип идентификатора(переменная, литерал), функция,
    //библиотечная функция, параметр, не определено
};

struct Entry
{
    short idxfirstLE; //индекс первой строки в таблице лексем
    char id[ID_MAXSIZE * 2] = ""; //идентификатор(автоматически усекается до ID_MAXSIZE)
    char postfix[ID_MAXSIZE]; //постфикс для области видимости идентификатора
    IDDATATYPE iddatatype = IDDATATYPE::UNDEF; //тип данных
    IDTYPE idtype = IDTYPE::U; //тип идентификатора.
    union
    {
        {
            char vlogical[6]; //значение logical
            int vtiny; //значение tiny
            struct
            {
                int len; //длина строки стр
                char str[TI_STR_MAXSIZE - 1]; //строка
            } vsymb[TI_STR_MAXSIZE]; //значение стр
        } value; //значение идентификатора
        struct
        {
            int count = 0;
            IDDATATYPE typeofparameter[MAXPARMCOUNT];
        } parameters; //параметры (если идентификатор - функция)
    };
};

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

Рисунок 5 Структуры для представления таблицы идентификаторов.

Приложение Г

Таблица лексем

----- Таблица лексем -----	
1	tfi(ti,ti,ti)
2	[
3	h(l)!
4	sti:b(i)!
5	h(i)!
6	gl!
7]
8	tfi()
9	[
10	h(l)!
11	gl!
12]
13	p
14	[
15	sti:!!
16	h(i)!
17	sti:!!
18	h(i)!
19	i:i/l!
20	h(i)!
21	sti:i+i+!!
22	h(i)!
23	sti:!!
24	h(i)!
25	sti:!!
26	sti:!!
27	sti:!!
28	h(i)!
29	h(l)!
30	sti:b(i)!
31	h(i)!
32	w(i^l)
33	[
34	i:l+!!
35	h(i)!
36]
37	o
38	[
39	h(i)!
40]
41	sti:(i-l)#(i+i)!
42	y(i>l)
43	[
44	i:i+l!
45]
46	h(i)!
47]

Рисунок 6 Таблица лексем.

Продолжение приложения Г

Таблица идентификаторов

-----Таблица идентификаторов-----					
Строка	Тип ID	Тип данных	Имя	Видимость	Значение
1	function	symbolic	foo	GLB	
1	param	symbolic	str	foo	
1	param	tiny	k	foo	0
1	param	tiny	m	foo	0
3	libfunc		show		
3	literal	symbolic	N1		"symlen"
4	variable	tiny	p	foo	0
4	libfunc	tiny	symlen	GLB	0
8	function	tiny	boo	GLB	0
10	literal	symbolic	N2		"boo function"
11	literal	tiny	N3		19
15	variable	tiny	x	perform	0
15	literal	tiny	N4		-100
17	variable	tiny	y	perform	0
17	literal	tiny	N5		31
19	literal	tiny	N6		2
21	variable	tiny	a	perform	0
21	literal	tiny	N7		34
23	variable	symbolic	str	perform	
23	literal	symbolic	N8		"sp ace"
25	variable	symbolic	mystr	perform	
25	literal	symbolic	N9		"а б в ? г д ж з к л а м о п р о с т у ф х ц ч"
26	variable	logical	mb	perform	
26	literal	logical	N10		true
27	variable	logical	mbd	perform	
27	literal	logical	N11		true
29	literal	symbolic	N12		"convert to tiny mystr"
30	variable	tiny	convert	perform	0
30	libfunc	tiny	symltotiny	GLB	0
32	literal	logical	N13		false
34	literal	tiny	N14		19
34	literal	tiny	N15		1
41	variable	tiny	fef	perform	0
41	literal	tiny	N16		12
42	literal	tiny	N17		5
44	literal	tiny	N18		1

Рисунок 7 Таблица идентификаторов.

Приложение Д

Цепочки правил для синтаксического анализа

```

Greibach greibach(
    NS('S'), TS('$'),           // стартовый символ, дно стека
    13,                          // количество правил
    Rule(
        NS('S'), GRB_ERROR_SERIES + 0, // неверная структура программы
        2,
        Rule::Chain(6, TS('t'), TS('f'), TS('i'), NS('F'), NS('B'), NS('S')),
        Rule::Chain(4, TS('p'), TS('['), NS('N'), TS(']'))
    ),
    Rule(
        NS('F'), GRB_ERROR_SERIES + 1, // отсутствует список параметров функции
        2,
        Rule::Chain(3, TS('('), NS('P'), TS(')')),
        Rule::Chain(2, TS('('), TS(')'))
    ),
    Rule(
        NS('P'), GRB_ERROR_SERIES + 2, // Ошибка в параметрах функции при её объявлении
        2,
        Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('P')),
        Rule::Chain(2, TS('t'), TS('i'))
    ),
    Rule(
        NS('B'), GRB_ERROR_SERIES + 3, // Отсутствует тело функции
        2,
        Rule::Chain(5, TS('[', TS('g'), NS('I'), TS('!'), TS(']')),
        Rule::Chain(6, TS('[', NS('N'), TS('g'), NS('I'), TS('!'), TS(']'))
    ),
    Rule(
        NS('I'), GRB_ERROR_SERIES + 4, // Недопустимое выражение. Ожидаются только литералы и идентификаторы
        2,
        Rule::Chain(1, TS('l')),
        Rule::Chain(1, TS('i'))
    ),
    Rule(
        NS('N'), GRB_ERROR_SERIES + 6, // Неверная конструкция в теле функции
        20,
        Rule::Chain(7, TS('s'), TS('t'), TS('i'), TS(':'), NS('E'), TS('!'), NS('N')),
        Rule::Chain(5, TS('s'), TS('t'), TS('i'), TS('!'), NS('N')),
        Rule::Chain(5, TS('i'), TS(':'), NS('E'), TS('!'), NS('N')),
        Rule::Chain(5, TS('i'), TS(':'), NS('H'), TS('!'), NS('N')),
        Rule::Chain(8, TS('y'), TS('('), NS('R'), TS(')'), TS('[', NS('X'), TS(']'), NS('N')),

```

Рисунок 8 Грамматика языка, нетерминалы S, F, P, B, I, N.

Продолжение приложения Д

```

Rule(
    NS('N'), GRB_ERROR_SERIES + 6,    // Неверная конструкция в теле функции
    20,
    Rule::Chain(7, TS('s'), TS('t'), TS('i'), TS(':'), NS('E'), TS('!'), NS('N')),
    Rule::Chain(5, TS('s'), TS('t'), TS('i'), TS(':'), NS('N')),
    Rule::Chain(5, TS('i'), TS(':'), NS('E'), TS('!'), NS('N')),
    Rule::Chain(5, TS('i'), TS(':'), NS('H'), TS('!'), NS('N')),
    Rule::Chain(8, TS('y'), TS(':'), NS('R'), TS(')'), TS('['), NS('X'), TS(']'), NS('N')),
    Rule::Chain(6, TS('h'), TS(')'), NS('I'), TS(')'), TS('!'), NS('N')),
    Rule::Chain(8, TS('w'), TS(')'), NS('R'), TS(')'), TS('['), NS('X'), TS(']'), NS('N')),
    Rule::Chain(12, TS('w'), TS(')'), NS('R'), TS(')'), TS('['), NS('X'), TS(']'), TS('o'), TS('['), NS('X'), TS(']'), NS('N')),
    Rule::Chain(4, TS('b'), NS('K'), TS('!'), NS('N')),
    Rule::Chain(4, TS('i'), NS('K'), TS('!'), NS('N')),

    Rule::Chain(6, TS('s'), TS('t'), TS('i'), TS(':'), NS('E'), TS('!')),
    Rule::Chain(4, TS('s'), TS('t'), TS('i'), TS('!')),
    Rule::Chain(4, TS('i'), TS(':'), NS('E'), TS('!')),
    Rule::Chain(4, TS('i'), TS(':'), NS('H'), TS('!')),
    Rule::Chain(7, TS('y'), TS(')'), NS('R'), TS(')'), TS('['), NS('X'), TS(']')),
    Rule::Chain(5, TS('h'), TS(')'), NS('I'), TS(')'), TS('!')),
    Rule::Chain(7, TS('w'), TS(')'), NS('R'), TS(')'), TS('['), NS('X'), TS(']')),
    Rule::Chain(11, TS('w'), TS(')'), NS('R'), TS(')'), TS('['), NS('X'), TS(']'), TS('o'), TS('['), NS('X'), TS(']')),
    Rule::Chain(3, TS('b'), NS('K'), TS('!')),
    Rule::Chain(3, TS('i'), NS('K'), TS('!'))

),

Rule(
    NS('R'), GRB_ERROR_SERIES + 7,    // Ошибка в условном выражении или выражении цикла
    8,
    Rule::Chain(3, TS('i'), TS('<'), NS('I')),
    Rule::Chain(3, TS('l'), TS('<'), NS('I')),
    Rule::Chain(3, TS('i'), TS('>'), NS('I')),
    Rule::Chain(3, TS('l'), TS('>'), NS('I')),
    Rule::Chain(3, TS('i'), TS('='), NS('I')),
    Rule::Chain(3, TS('l'), TS('='), NS('I')),
    Rule::Chain(3, TS('i'), TS('^'), NS('I')),
    Rule::Chain(3, TS('l'), TS('^'), NS('I'))

),

Rule(
    NS('H'), GRB_ERROR_SERIES + 10,    // Ошибка в арифметических или сдвиговых операциях
    4,
    Rule::Chain(3, TS('i'), TS('\'), TS('l')),
    Rule::Chain(3, TS('i'), TS('/'), TS('l')),
    Rule::Chain(3, TS('l'), TS('/'), TS('l')),
    Rule::Chain(3, TS('l'), TS('\'), TS('l'))

),

Rule(
    NS('E'), GRB_ERROR_SERIES + 8,    // Ошибка в вызове функции
    10,
    Rule::Chain(2, TS('b'), NS('K')),
    Rule::Chain(1, TS('i')),
    Rule::Chain(1, TS('l')),
    Rule::Chain(3, TS('(', NS('E'), TS(')')),
    Rule::Chain(2, TS('i'), NS('K')),

    Rule::Chain(2, TS('i'), NS('M')),
    Rule::Chain(2, TS('l'), NS('M')),
    Rule::Chain(4, TS('(', NS('E'), TS(')'), NS('M')),
    Rule::Chain(3, TS('i'), NS('K'), NS('M')),
    Rule::Chain(3, TS('b'), NS('K'), NS('M'))

),

Rule(
    NS('K'), GRB_ERROR_SERIES + 8,    // Ошибка в вызове функции
    2,
    Rule::Chain(3, TS('(', NS('W'), TS(')')),
    Rule::Chain(2, TS('(', TS(')'))

),

Rule(
    NS('W'), GRB_ERROR_SERIES + 9,    // ошибка в параметрах вызываемой функции
    4,
    Rule::Chain(1, TS('i')),
    Rule::Chain(1, TS('l')),
    Rule::Chain(3, TS('i'), TS(','), NS('W')),
    Rule::Chain(3, TS('l'), TS(','), NS('W'))

),

```

Рисунок 9 Грамматика языка, нетерминалы N, R, H, E, K, W.

Продолжение приложения Д

```

Rule(
    NS('M'), GRB_ERROR_SERIES + 10,    //Ошибка в арифметических или сдвиговых операциях
    8,
    Rule::Chain(3, TS('+'), NS('E'), NS('M')),
    Rule::Chain(3, TS('-'), NS('E'), NS('M')),
    Rule::Chain(3, TS('*'), NS('E'), NS('M')),
    Rule::Chain(3, TS('#'), NS('E'), NS('M')),

    Rule::Chain(2, TS('+'), NS('E')),
    Rule::Chain(2, TS('*'), NS('E')),
    Rule::Chain(2, TS('#'), NS('E')),
    Rule::Chain(2, TS('-'), NS('E'))
),
Rule(
    NS('X'), GRB_ERROR_SERIES + 11,    // Неверная конструкция в теле цикла/условного выражения
    12,
    Rule::Chain(5, TS('i'), TS(':'), NS('I'), TS('!'), NS('X')),
    Rule::Chain(5, TS('i'), TS(':'), NS('E'), TS('!'), NS('X')),
    Rule::Chain(5, TS('i'), TS(':'), NS('H'), TS('!'), NS('X')),
    Rule::Chain(6, TS('h'), TS('('), TS('i'), TS(')'), TS('!'), NS('X')),
    Rule::Chain(6, TS('h'), TS('('), TS('l'), TS(')'), TS('!'), NS('X')),
    Rule::Chain(4, TS('b'), NS('K'), TS('!'), NS('X')),
    Rule::Chain(4, TS('i'), NS('K'), TS('!'), NS('X')),

    Rule::Chain(4, TS('i'), TS(':'), NS('I'), TS('!')),
    Rule::Chain(4, TS('i'), TS(':'), NS('E'), TS('!')),
    Rule::Chain(4, TS('i'), TS(':'), NS('H'), TS('!')),
    Rule::Chain(5, TS('h'), TS('('), TS('i'), TS(')'), TS('!')),
    Rule::Chain(5, TS('h'), TS('('), TS('l'), TS(')'), TS('!')),
    Rule::Chain(3, TS('b'), NS('K'), TS('!')),
    Rule::Chain(3, TS('i'), NS('K'), TS('!'))
)

```

Рисунок 10 Грамматика языка, нетерминалы М, Х.

Приложение Е

Трассировка синтаксического разбора

----- Синтаксический анализ -----		
Шаг : Правило	Входная лента	Стек
1 : S->tfiFBS	tfiO[h(l)!gl!]p[sti:!!st	\$\$
1 : SAVESTATE:	1	
1 :	tfiO[h(l)!gl!]p[sti:!!st	tfiFBS\$
2 :	fiO[h(l)!gl!]p[sti:!!sti	fiFBS\$
3 :	iO[h(l)!gl!]p[sti:!!sti:	iFBS\$
4 :	O[h(l)!gl!]p[sti:!!sti:l	FBS\$
5 : F->(P)	O[h(l)!gl!]p[sti:!!sti:l	FBS\$
5 : SAVESTATE:	2	
5 :	O[h(l)!gl!]p[sti:!!sti:l	(P)BS\$
6 :)h(l)!gl!]p[sti:!!sti:!!	P)BS\$
7 : TNS_NORULECHAIN/NS_NORULE		
7 : RESTATE		
7 :	O[h(l)!gl!]p[sti:!!sti:l	FBS\$
8 : F->O	O[h(l)!gl!]p[sti:!!sti:l	FBS\$
8 : SAVESTATE:	2	
8 :	O[h(l)!gl!]p[sti:!!sti:l	OBS\$
9 :)h(l)!gl!]p[sti:!!sti:!!)BS\$
10 :	[h(l)!gl!]p[sti:!!sti:!!s	BS\$
11 : B->[gl!]	[h(l)!gl!]p[sti:!!sti:!!s	BS\$
11 : SAVESTATE:	3	
11 :	[h(l)!gl!]p[sti:!!sti:!!s	[gl!]S\$
12 :	h(l)!gl!]p[sti:!!sti:!!st	gl!]S\$
13 : TS_NOK/NS_NORULECHAIN		
13 : RESTATE		
13 :	[h(l)!gl!]p[sti:!!sti:!!s	BS\$
14 : B->[NgI!]	[h(l)!gl!]p[sti:!!sti:!!s	BS\$
14 : SAVESTATE:	3	
14 :	[h(l)!gl!]p[sti:!!sti:!!s	[NgI!]S\$
15 :	h(l)!gl!]p[sti:!!sti:!!st	NgI!]S\$
16 : N->h(I)!N	h(l)!gl!]p[sti:!!sti:!!st	NgI!]S\$
16 : SAVESTATE:	4	
16 :	h(l)!gl!]p[sti:!!sti:!!st	h(I)!NgI!]S\$
17 :	(l)!gl!]p[sti:!!sti:!!sti	(I)!NgI!]S\$
18 :	l)!gl!]p[sti:!!sti:!!sti:	I)!NgI!]S\$
19 : l->l	l)!gl!]p[sti:!!sti:!!sti:	I)!NgI!]S\$
19 : SAVESTATE:	5	
19 :	l)!gl!]p[sti:!!sti:!!sti:	I)!NgI!]S\$
20 :)!gl!]p[sti:!!sti:!!sti:l)!NgI!]S\$
21 :	!gl!]p[sti:!!sti:!!sti:!!	!NgI!]S\$
22 :	gl!]p[sti:!!sti:!!sti:!!s	NgI!]S\$
23 : TNS_NORULECHAIN/NS_NORULE		
23 : RESTATE		
23 :	l)!gl!]p[sti:!!sti:!!sti:	I)!NgI!]S\$
24 : TNS_NORULECHAIN/NS_NORULE		
24 : RESTATE		
24 :	h(l)!gl!]p[sti:!!sti:!!st	NgI!]S\$
25 : N->h(I)!	h(l)!gl!]p[sti:!!sti:!!st	NgI!]S\$
25 : SAVESTATE:	4	
25 :	h(l)!gl!]p[sti:!!sti:!!st	h(I)!gl!]S\$
26 :	(l)!gl!]p[sti:!!sti:!!sti	(I)!gl!]S\$

Рисунок 11 Трассировка синтаксического разбора.

Продолжение приложения Е

27 :	l)!gl!]p[sti:!!sti:!!sti:	l)!gl!]S\$
28 : I->l	l)!gl!]p[sti:!!sti:!!sti:	l)!gl!]S\$
28 : SAVESTATE:	5	
28 :	l)!gl!]p[sti:!!sti:!!sti:	l)!gl!]S\$
29 :)!gl!]p[sti:!!sti:!!sti:l)!gl!]S\$
30 :	!gl!]p[sti:!!sti:!!sti:!!	!gl!]S\$
31 :	gl!]p[sti:!!sti:!!sti:!!s	gl!]S\$
32 :	l!]p[sti:!!sti:!!sti:!!st	l!]S\$
33 : I->l	l!]p[sti:!!sti:!!sti:!!st	l!]S\$
33 : SAVESTATE:	6	
33 :	l!]p[sti:!!sti:!!sti:!!st	l!]S\$
34 :	!]p[sti:!!sti:!!sti:!!sti	!]S\$
35 :]p[sti:!!sti:!!sti:!!sti:]S\$
36 :	p[sti:!!sti:!!sti:!!sti:l	S\$
37 : S->p[N]	p[sti:!!sti:!!sti:!!sti:l	S\$
37 : SAVESTATE:	7	
37 :	p[sti:!!sti:!!sti:!!sti:l	p[N]\$
38 :	[sti:!!sti:!!sti:!!sti:l+	[N]\$
39 :	sti:!!sti:!!sti:!!sti:l+b	N]\$
40 : N->sti:E!N	sti:!!sti:!!sti:!!sti:l+b	N]\$
40 : SAVESTATE:	8	
40 :	sti:!!sti:!!sti:!!sti:l+b	sti:E!N]\$
41 :	ti:!!sti:!!sti:!!sti:l+b(ti:E!N]\$
42 :	i:!!sti:!!sti:!!sti:l+b(i	i:E!N]\$
43 :	:!!sti:!!sti:!!sti:l+b(i)	:E!N]\$
44 :	!sti:!!sti:!!sti:!!sti:l+b(i)!	E!N]\$
45 : E->l	!sti:!!sti:!!sti:!!sti:l+b(i)!	E!N]\$
45 : SAVESTATE:	9	
45 :	!sti:!!sti:!!sti:!!sti:l+b(i)!	!N]\$
46 :	!sti:!!sti:!!sti:!!sti:l+b(i)!w	!N]\$
47 :	sti:!!sti:!!sti:!!sti:l+b(i)!w(N]\$
48 : N->sti:E!N	sti:!!sti:!!sti:!!sti:l+b(i)!w(N]\$
48 : SAVESTATE:	10	
48 :	sti:!!sti:!!sti:!!sti:l+b(i)!w(sti:E!N]\$
49 :	ti:!!sti:!!sti:!!sti:l+b(i)!w(i	ti:E!N]\$
50 :	i:!!sti:!!sti:!!sti:l+b(i)!w(i^	i:E!N]\$
51 :	:!!sti:!!sti:!!sti:l+b(i)!w(i^l	:E!N]\$
52 :	!sti:!!sti:!!sti:!!sti:l+b(i)!w(i^l)	E!N]\$
53 : E->l	!sti:!!sti:!!sti:!!sti:l+b(i)!w(i^l)	E!N]\$
53 : SAVESTATE:	11	
53 :	!sti:!!sti:!!sti:!!sti:l+b(i)!w(i^l)	!N]\$
54 :	!sti:!!sti:!!sti:!!sti:l+b(i)!w(i^l)[!N]\$
55 :	sti:!!sti:!!sti:!!sti:l+b(i)!w(i^l)[h	N]\$
56 : N->sti:E!N	sti:!!sti:!!sti:!!sti:l+b(i)!w(i^l)[h	N]\$
56 : SAVESTATE:	12	
56 :	sti:!!sti:!!sti:!!sti:l+b(i)!w(i^l)[h	sti:E!N]\$
57 :	ti:!!sti:!!sti:!!sti:l+b(i)!w(i^l)[h(ti:E!N]\$
58 :	i:!!sti:!!sti:!!sti:l+b(i)!w(i^l)[h(i	i:E!N]\$
59 :	:!!sti:!!sti:!!sti:l+b(i)!w(i^l)[h(i	:E!N]\$
60 :	!sti:!!sti:!!sti:!!sti:l+b(i)!w(i^l)[h(i)!	E!N]\$
61 : E->l	!sti:!!sti:!!sti:!!sti:l+b(i)!w(i^l)[h(i)!	E!N]\$
61 : SAVESTATE:	13	

Рисунок 12 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

64 :	sti:l+b(i)!w(i^l)[h(i)!]	sti:E!N]\$
65 :	ti:l+b(i)!w(i^l)[h(i)!]	ti:E!N]\$
66 :	i:l+b(i)!w(i^l)[h(i)!]	i:E!N]\$
67 :	:l+b(i)!w(i^l)[h(i)!]	:E!N]\$
68 :	l+b(i)!w(i^l)[h(i)!]	E!N]\$
69 : E->l	l+b(i)!w(i^l)[h(i)!]	E!N]\$
69 : SAVESTATE:	15	
69 :	l+b(i)!w(i^l)[h(i)!]	l!N]\$
70 :	+b(i)!w(i^l)[h(i)!]	!N]\$
71 : TS_NOK/NS_NORULECHAIN		
71 : RESTATE		
71 :	l+b(i)!w(i^l)[h(i)!]	E!N]\$
72 : E->IM	l+b(i)!w(i^l)[h(i)!]	E!N]\$
72 : SAVESTATE:	15	
72 :	l+b(i)!w(i^l)[h(i)!]	IM!N]\$
73 :	+b(i)!w(i^l)[h(i)!]	M!N]\$
74 : M->+EM	+b(i)!w(i^l)[h(i)!]	M!N]\$
74 : SAVESTATE:	16	
74 :	+b(i)!w(i^l)[h(i)!]	+EM!N]\$
75 :	b(i)!w(i^l)[h(i)!]	EM!N]\$
76 : E->bK	b(i)!w(i^l)[h(i)!]	EM!N]\$
76 : SAVESTATE:	17	
76 :	b(i)!w(i^l)[h(i)!]	bKM!N]\$
77 :	(i)!w(i^l)[h(i)!]	KM!N]\$
78 : K->(W)	(i)!w(i^l)[h(i)!]	KM!N]\$
78 : SAVESTATE:	18	
78 :	(i)!w(i^l)[h(i)!]	(W)M!N]\$
79 :	i)!w(i^l)[h(i)!]	W)M!N]\$
80 : W->i	i)!w(i^l)[h(i)!]	W)M!N]\$
80 : SAVESTATE:	19	
80 :	i)!w(i^l)[h(i)!]	i)M!N]\$
81 :)!w(i^l)[h(i)!])M!N]\$
82 :	!w(i^l)[h(i)!]	M!N]\$
83 : TNS_NORULECHAIN/NS_NORULE		
83 : RESTATE		
83 :	i)!w(i^l)[h(i)!]	W)M!N]\$
84 : W->i,W	i)!w(i^l)[h(i)!]	W)M!N]\$
84 : SAVESTATE:	19	
84 :	i)!w(i^l)[h(i)!]	i,W)M!N]\$
85 :)!w(i^l)[h(i)!]	,W)M!N]\$
86 : TS_NOK/NS_NORULECHAIN		
86 : RESTATE		
86 :	i)!w(i^l)[h(i)!]	W)M!N]\$
87 : TNS_NORULECHAIN/NS_NORULE		
87 : RESTATE		
87 :	(i)!w(i^l)[h(i)!]	KM!N]\$
88 : K->Q	(i)!w(i^l)[h(i)!]	KM!N]\$
88 : SAVESTATE:	18	
88 :	(i)!w(i^l)[h(i)!]	Q)M!N]\$
89 :	i)!w(i^l)[h(i)!])M!N]\$
90 : TS_NOK/NS_NORULECHAIN		
90 : RESTATE		

Рисунок 13 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

90 :	(i)!w(i^l)[h(i)!]	KM!N]\$
91 :	TNS_NORULECHAIN/NS_NORULE	
91 :	RESTATE	
91 :	b(i)!w(i^l)[h(i)!]	EM!N]\$
92 : E->bKM	b(i)!w(i^l)[h(i)!]	EM!N]\$
92 : SAVESTATE:	17	
92 :	b(i)!w(i^l)[h(i)!]	bKMM!N]\$
93 :	(i)!w(i^l)[h(i)!]	KMM!N]\$
94 : K->(W)	(i)!w(i^l)[h(i)!]	KMM!N]\$
94 : SAVESTATE:	18	
94 :	(i)!w(i^l)[h(i)!]	(W)MM!N]\$
95 :	i)!w(i^l)[h(i)!]	W)MM!N]\$
96 : W->i	i)!w(i^l)[h(i)!]	W)MM!N]\$
96 : SAVESTATE:	19	
96 :	i)!w(i^l)[h(i)!]	i)MM!N]\$
97 :)!w(i^l)[h(i)!])MM!N]\$
98 :	!w(i^l)[h(i)!]	MM!N]\$
99 : TNS_NORULECHAIN/NS_NORULE		
99 :	RESTATE	
99 :	i)!w(i^l)[h(i)!]	W)MM!N]\$
100 : W->i,W	i)!w(i^l)[h(i)!]	W)MM!N]\$
100 : SAVESTATE:	19	
100 :	i)!w(i^l)[h(i)!]	i,W)MM!N]\$
101 :)!w(i^l)[h(i)!]	,W)MM!N]\$
102 : TS_NOK/NS_NORULECHAIN		
102 :	RESTATE	
102 :	i)!w(i^l)[h(i)!]	W)MM!N]\$
103 : TNS_NORULECHAIN/NS_NORULE		
103 :	RESTATE	
103 :	(i)!w(i^l)[h(i)!]	KMM!N]\$
104 : K->O	(i)!w(i^l)[h(i)!]	KMM!N]\$
104 : SAVESTATE:	18	
104 :	(i)!w(i^l)[h(i)!]	O)MM!N]\$
105 :	i)!w(i^l)[h(i)!])MM!N]\$
106 : TS_NOK/NS_NORULECHAIN		
106 :	RESTATE	
106 :	(i)!w(i^l)[h(i)!]	KMM!N]\$
107 : TNS_NORULECHAIN/NS_NORULE		
107 :	RESTATE	
107 :	b(i)!w(i^l)[h(i)!]	EM!N]\$
108 : TNS_NORULECHAIN/NS_NORULE		
108 :	RESTATE	
108 :	+b(i)!w(i^l)[h(i)!]	M!N]\$
109 : M->+E	+b(i)!w(i^l)[h(i)!]	M!N]\$
109 : SAVESTATE:	16	
109 :	+b(i)!w(i^l)[h(i)!]	+E!N]\$
110 :	b(i)!w(i^l)[h(i)!]	E!N]\$
111 : E->bK	b(i)!w(i^l)[h(i)!]	E!N]\$
111 : SAVESTATE:	17	
111 :	b(i)!w(i^l)[h(i)!]	bK!N]\$
112 :	(i)!w(i^l)[h(i)!]	K!N]\$

Рисунок 13 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

113 : K->(W)	$\tilde{w}(i^{\wedge}l)[h(i)!]$	$\tilde{K}!N] \$$
113 : SAVESTATE:	18	
113 :	$(i)!w(i^{\wedge}l)[h(i)!]$	$(W)!N] \$$
114 :	$i)!w(i^{\wedge}l)[h(i)!]$	$W)!N] \$$
115 : W->i	$i)!w(i^{\wedge}l)[h(i)!]$	$W)!N] \$$
115 : SAVESTATE:	19	
115 :	$i)!w(i^{\wedge}l)[h(i)!]$	$i)!N] \$$
116 :	$)!w(i^{\wedge}l)[h(i)!]$	$)!N] \$$
117 :	$!w(i^{\wedge}l)[h(i)!]$	$!N] \$$
118 :	$w(i^{\wedge}l)[h(i)!]$	$N] \$$
119 : N->w(R)[X]N	$w(i^{\wedge}l)[h(i)!]$	$N] \$$
119 : SAVESTATE:	20	
119 :	$w(i^{\wedge}l)[h(i)!]$	$w(R)[X]N] \$$
120 :	$(i^{\wedge}l)[h(i)!]$	$(R)[X]N] \$$
121 :	$i^{\wedge}l)[h(i)!]$	$R)[X]N] \$$
122 : R->i<I	$i^{\wedge}l)[h(i)!]$	$R)[X]N] \$$
122 : SAVESTATE:	21	
122 :	$i^{\wedge}l)[h(i)!]$	$i<I)[X]N] \$$
123 :	$^{\wedge}l)[h(i)!]$	$<I)[X]N] \$$
124 : TS_NOK/NS_NORULECHAIN		
124 : RESTATE		
124 :	$i^{\wedge}l)[h(i)!]$	$R)[X]N] \$$
125 : R->i>I	$i^{\wedge}l)[h(i)!]$	$R)[X]N] \$$
125 : SAVESTATE:	21	
125 :	$i^{\wedge}l)[h(i)!]$	$i>I)[X]N] \$$
126 :	$^{\wedge}l)[h(i)!]$	$>I)[X]N] \$$
127 : TS_NOK/NS_NORULECHAIN		
127 : RESTATE		
127 :	$i^{\wedge}l)[h(i)!]$	$R)[X]N] \$$
128 : R->i=I	$i^{\wedge}l)[h(i)!]$	$R)[X]N] \$$
128 : SAVESTATE:	21	
128 :	$i^{\wedge}l)[h(i)!]$	$i=I)[X]N] \$$
129 :	$^{\wedge}l)[h(i)!]$	$=I)[X]N] \$$
130 : TS_NOK/NS_NORULECHAIN		
130 : RESTATE		
130 :	$i^{\wedge}l)[h(i)!]$	$R)[X]N] \$$
131 : R->i^I	$i^{\wedge}l)[h(i)!]$	$R)[X]N] \$$
131 : SAVESTATE:	21	
131 :	$i^{\wedge}l)[h(i)!]$	$i^{\wedge}I)[X]N] \$$
132 :	$^{\wedge}l)[h(i)!]$	$^{\wedge}I)[X]N] \$$
133 :	$l)[h(i)!]$	$l)[X]N] \$$
134 : I->l	$l)[h(i)!]$	$l)[X]N] \$$
134 : SAVESTATE:	22	
134 :	$l)[h(i)!]$	$l)[X]N] \$$
135 :	$)[h(i)!]$	$)[X]N] \$$
136 :	$[h(i)!]$	$[X]N] \$$
137 :	$h(i)!]$	$X]N] \$$
138 : X->h(i)!X	$h(i)!]$	$X]N] \$$
138 : SAVESTATE:	23	
138 :	$h(i)!]$	$h(i)!X]N] \$$
139 :	$(i)!]$	$(i)!X]N] \$$
140 :	$i)!]$	$i)!X]N] \$$

Рисунок 14 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

141 :)!]])!X]N]\$	
142 :	!]]	!X]N]\$	
143 :]]	X]N]\$	
144 :	TNS_NORULECHAIN/NS_NORULE		
144 :	RESTATE		
144 :	h(i)!]]	X]N]\$	
145 :	X->h(l)!X h(i)!]]	X]N]\$	
145 :	SAVESTATE: 23		
145 :	h(i)!]]	h(l)!X]N]\$	
146 :	(i)!]]	(l)!X]N]\$	
147 :	i)!]]	l)!X]N]\$	
148 :	TS_NOK/NS_NORULECHAIN		
148 :	RESTATE		
148 :	h(i)!]]	X]N]\$	
149 :	X->h(i)! h(i)!]]	X]N]\$	
149 :	SAVESTATE: 23		
149 :	h(i)!]]	h(i)!N]\$	
150 :	(i)!]]	(i)!N]\$	
151 :	i)!]]	i)!N]\$	
152 :)!]])!N]\$	
153 :	!]]	!N]\$	
154 :]]]N]\$	
155 :]	N]\$	
156 :	TNS_NORULECHAIN/NS_NORULE		
156 :	RESTATE		
156 :	h(i)!]]	X]N]\$	
157 :	X->h(l)! h(i)!]]	X]N]\$	
157 :	SAVESTATE: 23		
157 :	h(i)!]]	h(l)!N]\$	
158 :	(i)!]]	(l)!N]\$	
159 :	i)!]]	l)!N]\$	
160 :	TS_NOK/NS_NORULECHAIN		
160 :	RESTATE		
160 :	h(i)!]]	X]N]\$	
161 :	TNS_NORULECHAIN/NS_NORULE		
161 :	RESTATE		
161 :	l)[h(i)!]]	l)[X]N]\$	
162 :	TNS_NORULECHAIN/NS_NORULE		
162 :	RESTATE		
162 :	i^l)[h(i)!]]	R)[X]N]\$	
163 :	TNS_NORULECHAIN/NS_NORULE		
163 :	RESTATE		
163 :	w(i^l)[h(i)!]]	N]\$	
164 :	N->w(R)[X]o[X]N w(i^l)[h(i)!]]	N]\$	
164 :	SAVESTATE: 20		
164 :	w(i^l)[h(i)!]]	w(R)[X]o[X]N]\$	
165 :	(i^l)[h(i)!]]	(R)[X]o[X]N]\$	
166 :	i^l)[h(i)!]]	R)[X]o[X]N]\$	
167 :	R->i<I i^l)[h(i)!]]	R)[X]o[X]N]\$	
167 :	SAVESTATE: 21		
167 :	i^l)[h(i)!]]	i<I)[X]o[X]N]\$	
168 :	^l)[h(i)!]]	<I)[X]o[X]N]\$	

Рисунок 15 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

```

169 : TS_NOK/NS_NORULECHAIN
169 : RESTATE
169 :          i^l)[h(i)!]]          R)[X]o[X]N]$
170 : R->i>I          i^l)[h(i)!]]          R)[X]o[X]N]$
170 : SAVESTATE:      21
170 :          i^l)[h(i)!]]          i>I)[X]o[X]N]$
171 :          ^l)[h(i)!]]          >I)[X]o[X]N]$
172 : TS_NOK/NS_NORULECHAIN
172 : RESTATE
172 :          i^l)[h(i)!]]          R)[X]o[X]N]$
173 : R->i=I          i^l)[h(i)!]]          R)[X]o[X]N]$
173 : SAVESTATE:      21
173 :          i^l)[h(i)!]]          i=I)[X]o[X]N]$
174 :          ^l)[h(i)!]]          =I)[X]o[X]N]$
175 : TS_NOK/NS_NORULECHAIN
175 : RESTATE
175 :          i^l)[h(i)!]]          R)[X]o[X]N]$
176 : R->i^I          i^l)[h(i)!]]          R)[X]o[X]N]$
176 : SAVESTATE:      21
176 :          i^l)[h(i)!]]          i^I)[X]o[X]N]$
177 :          ^l)[h(i)!]]          ^I)[X]o[X]N]$
178 :          l)[h(i)!]]          l)[X]o[X]N]$
179 : I->l          l)[h(i)!]]          l)[X]o[X]N]$
179 : SAVESTATE:      22
179 :          l)[h(i)!]]          l)[X]o[X]N]$
180 :          )[h(i)!]]          )[X]o[X]N]$
181 :          [h(i)!]]          [X]o[X]N]$
182 :          h(i)!]]          X]o[X]N]$
183 : X->h(i)!X          h(i)!]]          X]o[X]N]$
183 : SAVESTATE:      23
183 :          h(i)!]]          h(i)!X]o[X]N]$
184 :          (i)!]]          (i)!X]o[X]N]$
185 :          i)!]]          i)!X]o[X]N]$
186 :          )!]]          )!X]o[X]N]$
187 :          !]]          !X]o[X]N]$
188 :          ]]          X]o[X]N]$
189 : TNS_NORULECHAIN/NS_NORULE
189 : RESTATE
189 :          h(i)!]]          X]o[X]N]$
190 : X->h(l)!X          h(i)!]]          X]o[X]N]$
190 : SAVESTATE:      23
190 :          h(i)!]]          h(l)!X]o[X]N]$
191 :          (i)!]]          (l)!X]o[X]N]$
192 :          i)!]]          l)!X]o[X]N]$
193 : TS_NOK/NS_NORULECHAIN
193 : RESTATE
193 :          h(i)!]]          X]o[X]N]$
194 : X->h(i)!          h(i)!]]          X]o[X]N]$
194 : SAVESTATE:      23
194 :          h(i)!]]          h(i)!o[X]N]$
195 :          (i)!]]          (i)!o[X]N]$
196 :          i)!]]          i)!o[X]N]$

```

Рисунок 16 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

197 :)!]])!o[X]N]\$
198 :	!]]	!o[X]N]\$
199 :]]]o[X]N]\$
200 :]	o[X]N]\$
201 :	TS_NOK/NS_NORULECHAIN	
201 :	RESTATE	
201 :	h(i)!]]	X]o[X]N]\$
202 : X->h(I)!	h(i)!]]	X]o[X]N]\$
202 : SAVESTATE:	23	
202 :	h(i)!]]	h(I)!o[X]N]\$
203 :	(i)!]]	(I)!o[X]N]\$
204 :	i)!]]	I)!o[X]N]\$
205 :	TS_NOK/NS_NORULECHAIN	
205 :	RESTATE	
205 :	h(i)!]]	X]o[X]N]\$
206 :	TNS_NORULECHAIN/NS_NORULE	
206 :	RESTATE	
206 :	I][h(i)!]]	I)[X]o[X]N]\$
207 :	TNS_NORULECHAIN/NS_NORULE	
207 :	RESTATE	
207 :	i^I)[h(i)!]]	R)[X]o[X]N]\$
208 :	TNS_NORULECHAIN/NS_NORULE	
208 :	RESTATE	
208 :	w(i^I)[h(i)!]]	N]\$
209 : N->w(R)[X]	w(i^I)[h(i)!]]	N]\$
209 : SAVESTATE:	20	
209 :	w(i^I)[h(i)!]]	w(R)[X]]\$
210 :	(i^I)[h(i)!]]	(R)[X]]\$
211 :	i^I)[h(i)!]]	R)[X]]\$
212 : R->i<I	i^I)[h(i)!]]	R)[X]]\$
212 : SAVESTATE:	21	
212 :	i^I)[h(i)!]]	i<I)[X]]\$
213 :	^I)[h(i)!]]	<I)[X]]\$
214 :	TS_NOK/NS_NORULECHAIN	
214 :	RESTATE	
214 :	i^I)[h(i)!]]	R)[X]]\$
215 : R->i>I	i^I)[h(i)!]]	R)[X]]\$
215 : SAVESTATE:	21	
215 :	i^I)[h(i)!]]	i>I)[X]]\$
216 :	^I)[h(i)!]]	>I)[X]]\$
217 :	TS_NOK/NS_NORULECHAIN	
217 :	RESTATE	
217 :	i^I)[h(i)!]]	R)[X]]\$
218 : R->i=I	i^I)[h(i)!]]	R)[X]]\$
218 : SAVESTATE:	21	
218 :	i^I)[h(i)!]]	i=I)[X]]\$
219 :	^I)[h(i)!]]	=I)[X]]\$
220 :	TS_NOK/NS_NORULECHAIN	
220 :	RESTATE	
220 :	i^I)[h(i)!]]	R)[X]]\$

Рисунок 17 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

```

220 :          i^l)[h(i)!]]          R)[X]]$
221 : R->i^l          i^l)[h(i)!]]          R)[X]]$
221 : SAVESTATE:      21
221 :          i^l)[h(i)!]]          i^l)[X]]$
222 :          ^l)[h(i)!]]          ^l)[X]]$
223 :          l)[h(i)!]]          l)[X]]$
224 : l->l          l)[h(i)!]]          l)[X]]$
224 : SAVESTATE:      22
224 :          l)[h(i)!]]          l)[X]]$
225 :          )[h(i)!]]          )[X]]$
226 :          [h(i)!]]          [X]]$
227 :          h(i)!]]          X]]$
228 : X->h(i)!X          h(i)!]]          X]]$
228 : SAVESTATE:      23
228 :          h(i)!]]          h(i)!X]]$
229 :          (i)!]]          (i)!X]]$
230 :          i)!]]          i)!X]]$
231 :          )!]]          )!X]]$
232 :          !]]          !X]]$
233 :          ]]          X]]$
234 : TNS_NORULECHAIN/NS_NORULE
234 : RESTATE
234 :          h(i)!]]          X]]$
235 : X->h(l)!X          h(i)!]]          X]]$
235 : SAVESTATE:      23
235 :          h(i)!]]          h(l)!X]]$
236 :          (i)!]]          (l)!X]]$
237 :          i)!]]          l)!X]]$
238 : TS_NOK/NS_NORULECHAIN
238 : RESTATE
238 :          h(i)!]]          X]]$
239 : X->h(i)!          h(i)!]]          X]]$
239 : SAVESTATE:      23
239 :          h(i)!]]          h(i)!]]$
240 :          (i)!]]          (i)!]]$
241 :          i)!]]          i)!]]$
242 :          )!]]          )!]]$
243 :          !]]          !]]$
244 :          ]]          ]]$
245 :          ]          ]$
246 :          $
247 : LENTA_END
248 : ----->LENTA_END

```

Рисунок 18 Конец трассировки синтаксического разбора.

Приложение Ж

Результат генерации кода

```

using System;

namespace CourseProject
{
    class ZE12020
    {
        static string foo(string str, sbyte k, sbyte m)
        {
            Console.WriteLine("symlen");
            sbyte p = (sbyte)(ZE12020stdlib.StandartLibrary.Symlen(str));
            Console.WriteLine(str);
            return str;
        }
        static sbyte boo()
        {
            Console.WriteLine("boo function");
            return (sbyte)(29);
        }
        static void Main(string[] args)
        {
            sbyte x = -100;
            Console.WriteLine(x);
            sbyte y = 0;
            y = (sbyte)(x >> 2);
            y = (sbyte)(ZE12020stdlib.StandartLibrary.GenerTiny() + 2);
            Console.WriteLine(y);
            sbyte a = (sbyte)(x + y + 34);
            Console.WriteLine(a);
            string str = "sp ace";
            Console.WriteLine(str);
            string mystr = "а б в г д е ж з и к л м н о п р с т у ф х ц ч";
            string w = ZE12020stdlib.StandartLibrary.GetTime(str);
            Console.WriteLine(w);
            bool mb = true;
            bool k = false;
            bool mbd = ZE12020stdlib.StandartLibrary.GenerLogical();
            Console.WriteLine(mb);
            sbyte convert = (sbyte)(ZE12020stdlib.StandartLibrary.SymbToTiny(str) + 1);
            Console.WriteLine(convert);
            if(mb != false)
            {
                a = (sbyte)(19 + 1);
                Console.WriteLine(a);
            }
            else
            {
                Console.WriteLine(str);
            }
            sbyte fef = (sbyte)((x - 12) / (a + y));
            while(fef > 5)
            {
                fef = (sbyte)(fef + 1);
            }
            Console.WriteLine(fef);
            Console.ReadKey();
        }
    }
}

```

Рисунок 19 Исходный код программы на языке C#.

Приложение И

Тестирование ошибок транслятора

Таблица 1 Тестирование.

Фрагмент исходного кода	Генерируемое исключение
1	2
show(convert)! otherwise [show(str)!]	506: line 33, [SX] Неверная конструкция в теле функции
set symbolic strstrstrstrstrstr : "sp ace"!	Ошибка 308: [LA] Превышена максимальная длина имени идентификатора строка 23 позиция 34
set tiny y : 378!	Ошибка 309: [LA] Значение вне диапазона для литерала типа tiny [-128; +127] строка 17 позиция 14
set tiny y : 18q!	Ошибка 311: [LA] Ошибка лексического разбора строка 17 позиция 14
set tiny y : m12h!	Ошибка 311: [LA] Ошибка лексического разбора строка 17 позиция 15
set symbolic y: 3!	Ошибка 613: [SM] Типы данных в выражении не совпадают строка 17 позиция 0
tiny func boo() [show("boo function")!]	506: line 11, [SX] Неверная конструкция в теле функции 504: line 10, [SX] Неверное выражение. Ожидаются только идентификаторы и литералы 504: line 10, [SX] Неверное выражение. Ожидаются только идентификаторы и литералы

1	2
<pre>tiny func boo() [show("boo function")!~комментарий giveback "wd"!]</pre>	<p>Ошибка 608: [SM] Несовпадение типа функции и типа возвращаемого значения</p> <p>строка 8 позиция 0</p>
<pre>set tiny переменная : m100!</pre>	<p>Ошибка 111: [!] Недопустимый символ в исходном файле (-in)</p> <p>строка 15 позиция 11</p>
<pre>tiny func boo() []</pre>	<p>506: line 10, [SX] Неверная конструкция в теле функции</p> <p>503: line 9, [SX] Отсутствует тело функции</p> <p>501: line 8, [SX] Неверный список параметров функции</p>
<pre>when (mb > false) [a: 19+1! show(a)!]</pre>	<p>Ошибка 615: [SM] Логические операции < > можно производить только над типом tiny</p> <p>строка 32 позиция 0</p>
<pre>set logical mb : true! set logical mbd : true + mb!</pre>	<p>Ошибка 614: [SM] Арифметические операции можно производить только над типом tiny</p> <p>строка 27 позиция 0</p>
<pre>set logical mbd : 0!</pre>	<p>Ошибка 613: [SM] Типы данных в выражении не совпадают</p> <p>строка 27 позиция 0</p>

1	2
<pre> symbolic func foo(symbolic str,tiny k,tiny m,logical o) [show("symblen")! set tiny p : symblen(str)! show(str)! giveback str!] </pre>	<p>Ошибка 609: [SM] Превышено максимальное значение в параметрах определяемой функции (3)</p> <p>строка 1 позиция 54</p>
<pre> show(show)! </pre>	<p>504: line 31, [SX] Неверное выражение. Ожидаются только идентификаторы и литералы</p> <p>504: line 31, [SX] Неверное выражение. Ожидаются только идентификаторы и литералы</p> <p>504: line 31, [SX] Неверное выражение. Ожидаются только идентификаторы и литералы</p>