МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий
Кафедра Информационных систем и технологий
Специальность 1-40 01 01 «Программное обеспечение информационных технологий»
Специализация 1-40 01 01 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

«Разработка компилятора GDV-2022»

Выполнил студент Гайков Дмитрий Викторович
(Ф.И.О.)

Руководитель проекта ст.преп. Наркевич А.С.
(учен. степень, звание, должность, подпись, Ф.И.О.)

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

Консультанты преп.-стажер Карпович М. Н.
(учен. степень, звание, должность, подпись, Ф.И.О.)

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

Нормоконтролер преп.-стажер Карпович М. Н.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2022

Содержание

Введение

Задачей данного курсового проекта является разработка собственного языка программирования и компилятора для него. Язык называется GDV-2022. Написание компилятора будет осуществляться на языке C++.

Для выполнения курсового проекта были поставлены следующие задачи:

- Спецификация языка программирования
- Структура транслятора
- Разработка лексического анализатора
- Разработка синтаксического анализатора
- Разработка семантического анализатора
- Вычисление выражений
- Генерация кода на языке C++
- Контрольный пример
- Тестирование транслятора

1 Спецификация языка программирования

1.1 Характеристика языка программирования

Язык программирования GDV-2022 – это процедурный, универсальный, строго типизированный, компилируемый язык. Не является объектно-ориентированным.

1.2 Алфавит языка

Алфавит языка GDV-2022 основывается на таблице ASCII, представленной в таблице 1.1. Используемые в алфавите символы: [a … z], [A … Z], [0 … 9], [А … Я], [а … я] спецсимволы: () , ; :`' , а также символы пробела, табуляции и перевода строки.

Таблица 1.1. Таблица кодировок ASCII

32	пробел	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	

1.3 Применяемые сепараторы

Применяемые сепараторы в языке программирования GDV-2022 описаны в таблице 1.2.

Таблица 1.2. Сепараторы языка GDV-2022

Сепараторы	Назначение сепаратора
{ … }	Для блока функций
(…)	Для фактических или формальных параметров функции, а также для приоритета операций
‘пробел’, ‘табуляция’	Для разделения цепочек (не допускается в названиях идентификаторов и ключевых слов)
,	Для разделения параметров функции, цикла For
&, |, ~	Побитовые операции

Продолжение таблицы 1.2

Сепараторы	Назначение сепаратора
;	Для разделения программных инструкций
=	Оператор присваивания
?	Условный оператор
\n	Символ перехода на новую строку
<, >, ==, !=	Операторы сравнения

1.4 Применяемые кодировки

Для написания исходного кода на языке программирования GDV-2022 используется кодировка Windows-1251.

Таблица 1.3. Кодировка Windows-1251

	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

1.5 Типы данных

Язык GDV-2022 позволяет использовать 3 типа данных для переменных и функций: целочисленный (num), символьный(symb) и числа с плавающей

точкой(float). А также есть особый тип данных для функции — action. Описание этих типов данных приведено в таблице 1.4.

Таблица 1.4. Типы данных языка GDV-2022

Тип данных	Описание
num	Целочисленный четырехбайтный тип данных. Значения по умолчанию нет
symb	Символьный однобайтовый тип данных. Относительно является целочисленным типом данных, значения колеблются от -128 до 127
float	Тип данных, отвечающий за числа с плавающей точкой. Занимает 4 байта в памяти
action	Особый тип данных функции, который указывает, что данная функция является процедурой, которая не возвращает никакого значения

Ключевое слово ref используется только для параметров

1.6 Преобразование типов данных

Преобразование типов данных происходит неявно. Так как все типы данных языка GDV-2022 можно отнести к численным, их взаимодействию особо ничего не мешает, но все-таки бывают такие случаи, где без приведения типов не обойтись. При вызове функций, аргументы, переданные в нее, будут приведены к типу параметров данной функции. Также при использовании побитовых операций, тип данных float будет приведен к типу данных num. При использовании цикла For по типу данных параметра, определяющему шаг, будет определен тип данных, к которому будут приведены первые два параметра цикла.

Приведение в языке GDV-2022 позволяет не ограничивать возможности типа данных float, то есть использовать побитовые операции, позволяет определить, с чем работает цикл For, и позволяет беспрепятственно использовать функцию, передавая в параметры переменные, литералы любого типа данных.

1.7 Идентификаторы

При создании идентификатора можно использовать буквы латинского алфавита как верхнего, так и нижнего регистра, а также можно использовать цифры и символ нижнего подчеркивания. Максимальная длина идентификатора – 50 символов. При превышении длины, идентификаторы усекаются до длины, равной 50 символов. Идентификаторы не должны совпадать с ключевыми словами и с другими идентификаторами, созданных в одной и той же области видимости.

1.8 Литералы

В языке GDV-2022 предусмотрены 4 вида литералов: целочисленные и строковые, символьные и чисел с плавающей точкой. Краткое описание литералов приведено в таблице 1.5.

Таблица 1.5 Литералы

Литерал	Пояснение
Целочисленный	Максимально допустимое значение 2^{31}-1. Минимально допустимым является -2^{31}-1. При выходе за пределы допустимости выводится соответствующая ошибка. В случае отрицательного значения используется знак минус.
Символьный	Используются символы из кодировки Windows-1251. При присвоении целочисленного значения или значения числа с плавающей точкой будет присвоен символ с соответствующей кодировкой. Если значение выше или ниже, ошибки не будет, так как система сама решит проблему с переполнением.
Числа с плавающей точкой	Максимальное значение — 3.402823466e+38F. Минимальное значение — 1.175494351e-38F. При выходе за пределы вызывается ошибка. В случае отрицательного значения используется знак минус. Литерал должен иметь в записи символ точки.
Строковый	Используются символы кодировки ASCII. Максимальный размер строки — 255.

Строковый литерал используется только внутри оператора вывода console

1.9 Объявление данных

В языке программирования GDV-2022 необходимо объявить переменную до ее использования. Есть два типа объявления: с явной типизацией и с неявной. При явной мы должны сразу указывать тип данных переменной. При неявной типизации язык сам вычислит тип данных переменной, но для этого нужно использовать оператор '=>', после которого должен идти либо идентификатор, либо литерал, иначе будет ошибка.

Для определения типа переменной нужно использовать ключевое слово is после идентификатора, а после него сам тип данных (num, symb, float).

Пример объявления переменной целочисленного типа: n is num;

Пример объявления переменной символьного типа: s is symb;

Пример объявления переменной типа числа с плавающей точкой: f is float;

Пример объявления переменной целочисленного типа с неявной типизацией: n => 2;

Пример объявления переменной символьного типа с неявной типизацией: s => ‘s’;

Пример объявления переменной типа числа с плавающей точкой с неявной типизацией: f => .2.

Объявление функции схоже с объявлением обычной переменной, также используется оператор is после идентификатора, после него уже следует ключевое слово foo, что обозначает, что этот идентификатор типа функция, далее идет открывающая круглая скобочка, после которой идет объявление параметров. Объявление параметров проходит также, как объявление обычных переменных. Чтобы объявить несколько параметров, нужно сделать несколько объявлений и разделить их запятыми. После объявления будет следовать закрывающая кругла скобочка. Далее будет определен тип данных возвращаемого значения этой функции. Для определения используется, как правило, оператор is после скобочки. Типы данных функции совпадают с типами данных переменных, только тут добавляется дополнительный тип данных action. Тип данных указывается после оператора is. Далее будет идти тело функции, заключенное в фигурные скобки.

Пример объявления функции целочисленного типа представлен в листинге 1.1.

```
Sum is foo(a is num, b is num) is num
{
     return sum(a, b);
}
```

Листинг 1.1

Также язык программирования GDV-2022 позволяет объявлять шаблонные функции. Шаблонная функция в языке GDV-2022 — функция, объявленная не в глобальной области видимости, а в другой функции или в main. Их объявление такое же, как и объявление обычной функции, только типа данных action нет и после закрывающей фигурной скобки будет идти символ точки с запятой.

1.10 Инициализация данных

После объявления переменной ей можно сразу предать значение с помощью оператора присваивания (=). Он является бинарным, так что тут должно быть два операнда. Слева должна быть переменная, а справа выражение, дающее какое-либо значение.

1.11 Инструкции языка

Инструкции языка программирования GDV-2022 представлены в таблице 1.6

Таблица 1.6. Инструкции языка GDV-2022

Инструкция	Запись на языке GDV-2022
Объявление переменной с явной типизацией	<идентификатор> is <тип данных>;

Продолжение таблицы 1.6

Инструкция	Запись на языке GDV-2022
Инициализация	<идентификатор> => <выражение>;
Вывод данных:	console([<список параметров>]);
Вызов подпрограммы	<идентификатор функции> ([<список параметров>]);
Перевод строки	console();
Присваивание	<идентификатор> = <выражение>;
Разветвление	(<условное выражение>) ? Truth{<тело блока истины>} Lie {<тело блока лжи>}
Цикличность	For(<идентификатор|литерал|функция>,<идентификатор|литерал|функция>, <идентификатор|литерал|функция>, <идентификатор> => {<тело цикла For>});

Тело цикла For не может быть пустым

1.12 Операции языка

В языке программирования GDV-2022 присутствуют операции, описанные в таблице 1.7. Операции, заключенные в (..) (круглые скобки) имеют наивысший приоритет, равный 4. Операция ~ (логическое не) имеет приоритет 3. Операция & (логического умножения) имеет приоритет, равный 2. Операция | (логическое сложение) имеет низший приоритет, равный 1.

Операции логического умножения и логического сложения являются бинарными, то есть требуют. Операция логического не является унарной.

Также язык GDV-2022 предоставляет возможность использовать операции сравнения значений. Они также представлены в таблице 1.7.

Таблица 1.7. Операции языка GDV-2022

Операции	Операторы
Побитовые	| (логическая сложение) & (логическое умножение) ~ (логическое не)
Сравнения	> (больше) < (меньше) == (равенство по значению) != (значения не равны) is (проверить тип переменной)

1.13 Выражения и их вычисление

Предусмотрены следующие правила составления выражений:

- Рассматриваются слева направо.
- Для изменения приоритета операции используются круглые скобки ()
- Каждое выражение должно заканчиваться сепаратором

В выражения можно использовать комбинацию всех типов данных, кроме типа данных процедуры action и строкового литерала. В выражениях можно использовать все виды операций языка GDV-2022. Также есть специальные функции из статической библиотеки sum(), mult(), division(), minus() для выполнения арифметических операций.

Выражения можно использовать после операции присваивания и после оператора возвращения значения return.

1.14 Конструкции языка

Программные конструкции языка программирование GDV-2022 приведены в таблице 1.8.

Таблица 1.8. Программные конструкции языка

Конструкции	Представление в языке
Точка входа в программу	main {…}

Пользовательская функция	<идентификатор> is foo(<идентификатор> is <тип данных>, …) is <тип данных>
	{
	…
	return <выражение>;
	}

Процедура	<идентификатор> is foo(<идентификатор> is <тип данных>, …) is action
	{
	…
	}

Шаблонная функция	<идентификатор> is foo(<идентификатор> is <тип данных>, …) is <тип данных>
	{
	…
	return <выражение>;
	};

В конце функции, не процедуры, всегда должен идти return

1.15 Области видимости идентификаторов.

Область видимости построена по принципу C++. Все идентификаторы должны быть вызваны из текущей области видимости или областей видимости, в которых они находятся. Дополнительных указателей не требуется. Для создания искусственной области видимости используются фигурные скобки {…}.

1.16 Семантические проверки

Перечень семантических проверок, предусмотренных языком программирования GDV-2022, приведен в таблице 1.9.

Таблица 1.9. Семантические проверки

№	Проверка
1	Единственность точки входа
2	Превышение размера целочисленных, строковых и плавающих литералов
3	Правильность выражений
4	Правильность передаваемых в функцию параметров: количество, типы
5	Переопределение идентификаторов
6	Использование идентификаторов без их объявления

1.17 Распределение оперативной памяти на этапе выполнения

Все переменные размещаются в стеке.

1.18 Стандартная библиотека и её состав

В языке GDV-2022 присутствует стандартная библиотека, которая подключает некоторые функции автоматически при трансляции исходного кода в язык C++, а некоторые функции можно подключить с помощью ключевого слова @import. Содержимое библиотеки и описание функций представлено в таблице 1.10.

Таблица 1.10. Стандартная библиотека языка GDV-2022

Функция	Описание
sum is foo(a is num, b is num) is num;	Математическая функция. Заменяет арифметическое суммирование. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный
sum is foo(a is float, b is float) is float;	Математическая функция. Заменяет арифметическое суммирование. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение

Продолжение таблицы 1.10

Функция	Описание
minus is foo(a is num, b is num) is num;	Математическая функция. Заменяет арифметическую разницу. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный
minus is foo(a is float, b is float) is float;	Математическая функция. Заменяет арифметическую разницу. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение
mult is foo(a is num, b is num) is num;	Математическая функция. Заменяет арифметическое произведение. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный
mult is foo(a is float, b is float) is float;	Математическая функция. Заменяет арифметическое произведение. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение
division is foo(a is num, b is num) is num;	Математическая функция. Заменяет арифметическое деление. Подключается автоматически. Тип параметров и возвращаемого значения целочисленный
division is foo(a is float, b is float) is float;	Математическая функция. Заменяет арифметическое деление. Подключается автоматически. Тип параметров и возвращаемого значения плавающее значение
pow is foo(number is num, power is num) is num;	Математическое функция. Выполняет роль возведения числа в степень. Подключается с помощью ключевого слова @import. Тип параметров и возвращаемого значения целочисленный
abs is foo(number is num) is num;	Математическая функция. Возвращает модуль от числа, переданного в параметр. Подключается при помощи ключевого слова @import. Типы данных параметра и возвращаемого значения совпадают и равны num
round is foo(number is num) is float;	Математическая функция. Возвращает округленное число, переданное в параметр. Подключается с помощью ключевого слова @import. Типы данных параметра и возвращаемого значения совпадают и равны float

1.19 Ввод и вывод данных

В языке GDV-2022 вывод данных осуществляется с помощью оператора console. В качестве аргумента могут выступать литералы, вызываемые функции и идентификаторы. В операторе console может быть неограниченное количество передаваемых аргументов. Ввод в данном языке программирования не предусмотрен.

1.20 Точка входа

Точкой входа в программе является ключевое слово “main”. Точка входа может отсутствовать. Но более чем одной точки входа быть не может.

1.21 Препроцессор

В языке GDV-2022 препроцессор не предусмотрен.

1.22 Соглашения о вызовах

Используется соглашение _cdecl, то есть все параметры передаются в стек справа налево, память высвобождает сама функция.

1.23 Объектный код

Объектный код реализован на основе языка программирования C++.

1.24 Классификация сообщений транслятора

Классификация сообщений транслятора приведена в таблице 1.11.

Таблица 1.11. Сообщения транслятора

Коды сообщений	Принадлежность
0 – 200	Системные ошибки, ошибки лексического анализа
600-800	Ошибки, семантического анализа, синтаксического анализа
201-599, 801-999	Зарезервированные коды ошибок

1.25 Контрольный пример

Исходный код контрольного примера представлен в листинге 1.2.

```
Fact is foo(a is num) is num
{
    res => 1;
```

Листинг 1.2 Контрольный пример

```
        For(1, a, 1, el =>
        {
            res = mult(res, el);
        });

        return res;
    }

    main
    {
        a => Fact(3);

         (a == 6) ?
         {
              `a равно 6`
         } : {
              `a не равно 6`
         }

         @import round;

         half is float = round(78.67);

         console(`half = `, half);
    }
```

Продолжение листинга 1.2

2 Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор языка программирования GDV-2022 состоит из следующих частей:

Лексический анализатор – часть транслятора, на котором выполняется лексический анализ. На данном этапе распознаётся правильность составления лексем и идентификаторов.

Синтаксический анализатор – часть транслятора, на которой выполняется синтаксический анализ. Проверяется правильность расположения идентификаторов и ключевых слов в исходном коде. Для того, чтобы провести данную операцию используются таблица лексем и идентификаторов.

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть исходный код проверяется на наличие ошибок. Входными данными являются таблица лексем и идентификаторов.

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

2.2 Перечень входных параметров транслятора

Для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов используются входные параметры транслятора, которые приведены в таблице 2.1.

Таблица 2.1. Входные параметры транслятора языка GDV-2022

Входной параметр	Описание параметра	Значение по умолчанию
-in:<имя in-файла>	Файл с исходным кодом на языке программирования GDV-2022, имеющий расширение .txt	Не предусмотрено
-log:<имя log-файла>	Файл, содержащий вывод протокола работы программы.	Значение по умолчанию: <имя in-файла>.log
-out:<имя out-файла>	Выходной файл – результат работы транслятора. Содержит исходный код на языке ассемблера.	Значение по умолчанию: <имя in-файла>.out

2.3 Протоколы, формируемые транслятором

В ходе работы программы формируются протоколы работы лексического, синтаксического и семантического анализаторов, которые содержат в себе перечень

протоколов работы. В таблице 2.2 приведены протоколы, формируемые транслятором и их содержимое.

Таблица 2.2 Протоколы, формируемые транслятором языка GDV-2022

Формируемый протокол	Описание выходного протокола
Файл журнала, заданный параметром "-log:" log.txt	Файл с протоколом работы транслятора языка программирования GDV-2022. Содержит таблицу лексем.
Выходной файл, заданный параметром "-out:" out.cpp	Результат работы программы – файл, содержащий исходный код на языке C++.
LT.txt	Сформированная таблица лексем
IT.txt	Сформированная таблица идентификаторов
SNT.txt	Содержит протокол работы синтаксического анализатора и дерево разбора, полученные на этапе синтаксического анализа.

Лог состоит из 4 файлов: сам лог файл, таблицы лексем, идентификаторов, разбора

3 Разработка лексического анализатора

3.1 Структура лексического анализатора

Лексический анализатор – часть компилятора, которая выполняет лексический анализ. На данном этапе распознаётся правильность составления лексем и идентификаторов языка. Для работы лексический анализатор использует исходный код на языке GDV-2022. В итоге будут сформированы таблица лексем и таблица идентификаторов. Также во время лексического анализа проверяется много синтаксических и семантических ошибок. А также тут строятся области видимости для идентификаторов. Сделано при помощи такой структуры данных, как стек. Во время выполнения лексического анализа некоторые элементы могут заменятся, примером является оператор =>. Допустим выражение a => 1 заменяется на a is num = 1.

Структура лексического анализатора представлена на рисунке 3.1.

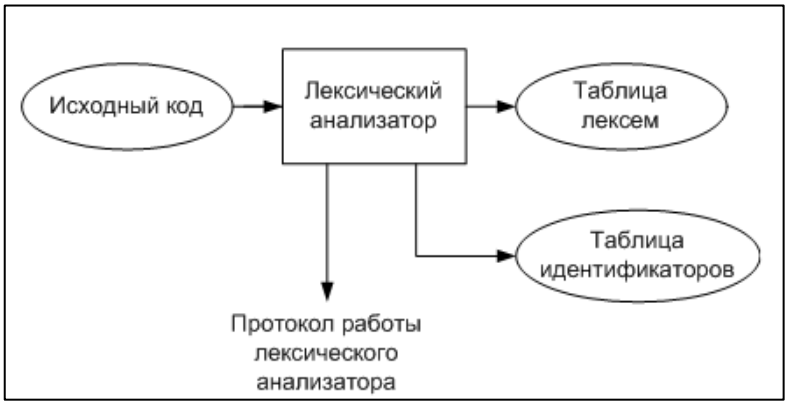


Рис. 3.1 Структура лексического анализатора

3.2 Контроль входных символов

При передаче исходного кода в лексический анализатор, все символы разделяются по определённым категориям, для дальнейшего использования. Категории входных символов представлены в таблице 3.1.

Таблица 3.1 Соответствие символов и их значений в таблице

Значение в таблице входных символов	Символы
Разрешенный	T
Запрещенный	F
Игнорируемый	I

3.3 Удаление избыточных символов

В языке программирования GDV-2022 предусмотрено удаление избыточных символов. Удаление избыточных символов происходит на этапе формирования слов, которые поступят на вход лексического анализатора. Пробелы и символы

табуляции не участвуют в формировании слов, если только они не внутри строкового или символьного литерала.

3.4 Перечень ключевых слов

Соответствие ключевых слов, сепараторов, символов операций с лексемами приведено в таблице 3.2.

Таблица 3.2 Соответствие ключевых слов и сепараторов с лексемами

Конструкция	Лексема	Примечание
num float symb ref num ref float ref symb	t	Названия типов данных языка. Ключевое слово ref обозначает то, что параметр будет передан по ссылке.
Идентификатор	i	Длина идентификатора – 50 символов.
Литерал	l	Литерал любого доступного типа.
foo	f	Объявление функции.
action	a	Ключевое слово для процедур – функций, не возвращающих значения. Указывается после второго оператора is.
return	r	Выход из функции типа action. Возвращение значения из функций других типов.
main	m	Точка входа в программу.
is	s	Оператор, позволяющий присвоить идентификатору любой тип данных. Используется также для проверки переменной на тип.
console	c	Поток вывода.
break	b	Оператор для выхода из цикла For
;	;	Разделение выражений.
,	,	Разделение параметров функций.
{	{	Начало блока/тела функции.
}	}	Закрытие блока/тела функции.
(	(	Передача параметров в функцию, приоритет операций.
)	)	Закрытие блока для передачи параметров, приоритет операций.
=	=	Знак присваивания.
&, |	v	Бинарные операции.
~	~	Унарная операция.

Продолжение таблицы 3.2

Конструкция	Лексема	Примечание
For	p	Вызов цикла For.
skip	z	Оператор, позволяющий пропустить итерацию в цикле For.
>, <, ==, !=	V	Операторы сравнения.
?	?	Условный оператор.
Truth	T	Блок, куда перейдет действие, если выражение в условном операторе истинно.
Lie	L	Блок, куда перейдет действие, если выражение в условном операторе ложно.

Реализации графов переходов находятся в приложении А.

Пример реализованного конечного автомата ключевого слова main языка GDV-2022 представлен на рисунке 3.2.

```
#define CHECK_MAIN (char*)"",\
                    5,\
                    NODE(1, RELATION('m', 1)),\
                    NODE(1, RELATION('a', 2)),\
                    NODE(1, RELATION('i', 3)),\
                    NODE(1, RELATION('n', 4)),\
                    NODE()
```

Рис. 3.2 Реализация конечного автомата для ключевого слова main

3.5 Основные структуры данных

Основные структуры данных языка GDV-2022 являются таблица лексем и таблица идентификаторов.

За таблицу лексем отвечает структура данных, называемая LexTable. У нее есть поле maxsize, которое отвечает за максимальное количество элементов в таблице. Следующее поле size, оно уже отвечает за количество элементов, находящихся внутри таблицы. И главное хранилище элементов — это поле table, которое является массивом элементов типа LT::Entry.

За элемент массива, который находится внутри таблицы лексем, отвечает тип данных Entry. Это структура данных, находящаяся внутри пространства имен LT. Тут есть такие поля, как lexema, которое отвечает за значение лексемы; как view, которое отвечает за то, какая бинарная операция используется вместо лексемы 'v'; как поле sn, которое отвечает за то, какая строка в исходном коде соответствует данной лексеме; как поле idxTI, которое используется только идентификаторами или же лексемами для отображения индекса данной лексемы в таблице идентификаторов.

За таблицу идентификаторов отвечает структура данных IdTable. Ее поля абсолютно такие же, как и ys таблицы лексем. Только вот элементом массива table

является структура Entry, находящаяся внутри пространства имен IT. У нее есть поля: idxfirstLE, countParams, id, iddatatype, idtype, hasValue, isRef, isFromStatic, params, needToInt, value. Поле idxfirstLE отображает индекс данного идентификатора, лексемы в таблице лексем. Поле countParams используется только для идентификаторов, отвечающих за функции, отображает количество параметров данной функции. Поле id — это строка, в которой содержится имя идентификатора. Поле iddatatype — это перечисление, отвечающее за тип данных идентификатора, может принимать значения IT::ACTION, IT::NUM, IT::SYMB, IT::FLOAT, IT::STR. Поле idtype также является перечислением, только оно уже отвечает за то, какого типа данный идентификатор — функция, литерал, параметр, переменная. Поле hasValue — это булевая переменная, имеет значение true, если данный идентификатор инициализирован, false, если нет. Поле isRef — также булевая переменная, только она уже отвечает за то, чтобы проверять данный идентификатор, который является параметром, передается по ссылке. Поле isFromStatic также является булевой переменной, оно уже отвечает за то, чтобы понять, что данный идентификатор из стандартной библиотеки. Поле params — это вектор, в котором хранятся типы данных параметров функции. Используется для проверки передаваемых параметров в вызываемую функцию. Поле value — является объединением, в котором есть поля для хранения значений целочисленного, вещественного, символьного и строкового литералов.

Основные структуры данных приведены в приложении А.

3.6 Структура и перечень сообщений лексического анализатора

Структура сообщений содержит информацию о номере сообщения, номер строки, где было вызвано сообщение в исходном коде, информацию об ошибке. Перечень сообщений представлены в таблице 3.3.

Таблица 3.3. Перечень ошибок лексического анализатора

Код сообщения	Содержание сообщения
113	Ошибка при создании таблицы лексем. Превышена емкость таблицы лексем
114	Ошибка при добавлении лексемы в таблицу. Таблица лексем заполнена
115	Ошибка при получении лексемы из таблицы. Недопустимый номер лексемы

3.7 Принцип обработки ошибок

В случае возникновения ошибок происходит их протоколирование и описание в командной строке с номером ошибки и сообщением.

3.8 Параметры лексического анализатора

Параметром лексического анализатора является очередь из структур, полями которых являются лексемы в исходном коде, полученные на этапе проверки кода на допустимость символов.

3.9 Алгоритм лексического анализа

Алгоритм работы лексического анализа заключается в распознавании и разборе цепочек исходного кода на основе конечных автоматов, а также заполнение таблиц идентификаторов и лексем. Работу конечного автомата можно показать с помощью графа переходов. Пример графа для цепочки «main» приведен на рисунке 3.3.

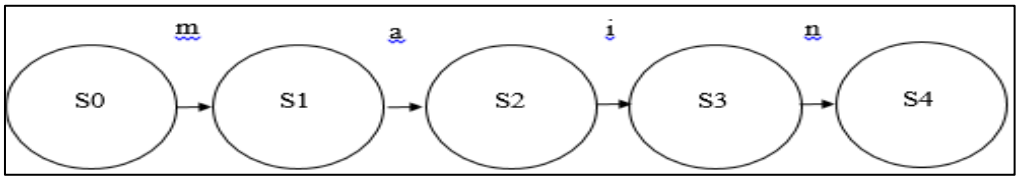


Рис 3.3 – Пример графа для цепочки main

3.10 Контрольный пример

Контрольный пример в виде таблиц лексем и идентификаторов представлен в приложении Б.

4 Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

Синтаксический анализ – фаза компилятора, которая выполняется после лексического анализа. В этой фазе будут распознаваться синтаксические конструкции. На вход синтаксического анализатора будет подаваться таблица лексем и таблица идентификаторов, а результатом работы будет дерево разбора.

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

Синтаксис языка GDV-2022 описывается грамматикой типа 2 по иерархии Хомского:

G = <T, N, P, S>

T – множество терминальных символов (алфавит языка GDV-2022),

N – множество нетерминальных символов,

P – множество правил языка,

S – начальный символ грамматики, представленный нетерминальным символом «A».

Множество терминальных символов соответствует элементам, содержащимся в таблице лексем. Правила нетерминальных символов описаны в таблице 4.1

Таблица 4.1. Правила нетерминальных символов

Нетерминалы	Назначение	Правила
A	Стартовый символ	iTA
		iT
		iW
		iWA
		iU=BXA
		iU=BX
		iUXA
		iUX
		i=BXA
		i=BX
		m{J}A
		m{J}
		m{G}
		m{G}A
T	Инициализация и объявление функции	sfCU{G}
		sfCU{J}

Продолжение таблицы 4.1

Нетерминалы	Назначение	Правила
U	Определение типа данных	st
V	Тело функции типа Action	iUXV
		iUX
		i(FXV
		i(FX
		i(YXV
		i(YX
		iU=BXV
		iU=BX
		i=BXV
		i=BX
		{V}V
		{V}
		{}V
		{}
		cLXV
		cLX
		pOV
		pO
		(KVKY?MV
		(KVKY?M
		iTXV
		iTX
		rX
		rXV
W	Инициализация функции типа action	sfCsa{V}
		sfCsa{}
X	Точка с запятой	;
Y	Закрывающая круглая скобка	)
B	Выражения	l
		i
		i(FY
		I(Y
		ivB
		lvB
		I(FYvB
		I(YvB
		(BY
		(BYvB
		~B
C	Конструкция параметров инициализируемой функции	(Y

Продолжение таблицы 4.1

		(DY
D	Объявление параметров	iU
		iU,D
E	Вызов функции	i(Y
		i(FY
F	Аргументы вызываемой функции	i
		l
		i(FY
		i(Y
		i,F
		l,F
		i(FY,F
		i(Y,F
G	Тело функции с возвращаемым значением	iUXG
		iUXJ
		i(FXG
		i(FXJ
		i(YXG
		i(YXJ
		iU=BXG
		iU=BXJ
		i=BXG
		i=BXJ
		{N}G
		{N}J
		{}G
		{}J
		cLXG
		cLXJ
		pOG
		pOJ
		(KVKY?MG
		(KVKY?MJ
		iTXG
		iTXJ
H	Инициализация переменной	ist
I	Тело условного оператора	i(FYXJ
		i(FYXI
		i(FYX
		i(YXJ
		i(YXI
		i(YX
		iUXI

Продолжение таблицы 4.1

		iUXJ
		iUX
		iU=BXI
		iU=BXJ
		iU=BX
		i=BXI
		i=BXJ
		i=BX
		{I}I
		{I}J
		{I}
		{}
		{}I
		{}J
		iTXI
		iTXJ
		iTX
		cLXI
		cLXJ
		cLX
		(KBKY?MI
		(KBKY?MJ (KBKY?M
		rBX
		rX
		bX
		bXI
		zX
		zXI
J	Возвращение значения функций	rBX
		rX
K	Отвечает за все виды значений(функция, литерал, идентификатор)	i
		l
		i(Y
		I(FY
M	Отвечает за блоки условного оператора	T{I}
		T{I}L{I}
		L{I}
		T{}
		L{}
		T{}L{I}
		T{I}L{}
		T{}L{}

Продолжение таблицы 4.1

L	Тело оператора потока вывода	(FY
		(Y
N	Отвечает за искусственную область видимости	i(FYXJ
		i(FYXN
		i(FYX
		i(YXJ
		i(YXN
		i(YX
		iUXN
		iUXJ
		iUX
		iU=BXN
		iU=BXJ
		iU=BX
		i=BXN
		i=BXJ
		i=BX
		{I}N
		{I}J
		{I}
		{}
		{}N
		{}J
		iTXN
		iTXJ
		iTX
		cLXN
		cLXJ
		cLX
		(KBKY?MN
		(KBKY?MJ
		(KBKY?M
		rBX
		rX
		bX
		bXN
		zX
		zXN
O	Параметры цикла For	(K,K,K,i{P}YX
P	Тело цикла For	i(FYXP
		i(FYX
		i(YP
		i(Y

Продолжение таблицы 4.1

		iUXP
		iUX
		iU=BXP
		iU=BX
		i=BXP
		i=BX
		{P}P
		{P}
		{}
		{}P
		iTXP
		iTX
		cLXP
		cLX
		pOP
		pO
		(KVKY?MP
		(KVKY?M
		bX
		bXP
		zX
		zXP

Множество терминалов дает возможность наиболее точно вывести ошибку

4.3 Построение конечного магазинного автомата

Распознавателем грамматики является конечный автомат с магазинной памятью, который представляет собой семерку $M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$. Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2. Описание компонент магазинного автомата

Компонента	Определение
Q	Множество состояний автомата
V	Алфавит входных символов
Z	Алфавит специальных магазинных символов
δ	Функция переходов автомата
q_0	Начальное состояние автомата
z_0	Начальное состояние магазина автомата
F	Множество конечных состояний

4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора представлены в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила и цепочки правил. Данные структуры представлены в приложении В.

4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора можно описать следующим образом:

- В магазин записывается стартовый символ.
- На основе полученной таблицы лексем формируется входная лента.
- Запускается автомат и выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке.
- Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется с ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала.
- Если в магазине встретился нетерминал, переходим к пункту 3.
- Если символ достиг символа дна стека, и лента в этот момент имеет символ дна стека, то синтаксический анализ выполнен успешно. Иначе генерируется ошибка.

4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен в таблице 4.3.

Таблица 4.3. Перечень ошибок синтаксического анализатора

Код сообщения	Содержание сообщения
650	Ошибка в глобальной области!
651	Ошибка в выражении
652	Ошибка при построении контрукции параметров
653	Ошибки при указании типов данных для параметров
654	Ошибка при вызове функции
655	Ошибка в параметрах вызываемой функции
656	Ошибка в теле функции
657	Ошибка в инициализации переменных
658	Ошибка в использовании условного оператора
659	Ошибка при возвращении значения функции
660	Ошибка в при использовании все возможных типов значений
661	Ошибка в теле условного оператора
662	Ошибка при вызове оператора console(). Неверный синтаксис. console(Значение);
663	Ошибка в искусственной области видимости!

Продолжение таблицы 4.3

664	Ошибка в параметрах цикла
665	Ошибка в теле цикла
666	Ошибка при инициализации функции
667	Ошибка в определении типа переменной/функции
668	Ошибка в теле функции типа Action"),
669	Ошибка при инициализации функции Action
670	Ошибка. Не хватает точки с запятой
671	Ошибка. Ожидалась закрывающая круглая скобка ‘)’

Количество правил соответствует количеству ошибок

4.7 Параметры синтаксического анализатора и режимы его работы

Входными параметрами для синтаксического анализатора в языке программирования GDV-2022 являются таблица лексем и таблица идентификаторов.

4.8 Принцип обработки ошибок

Синтаксический анализатор перебирает все возможные правила и цепочки правила грамматики в целях поиска подходящего соответствия. Если ни одна из цепочек правила не подошла для рассматриваемой конструкции, то генерируется ошибка в соответствии с таблицей 4.3. Ошибка заносится в протокол.

4.9 Контрольный пример

Пример разбора исходного кода на языке программирования GDV-2022 синтаксическим анализатором представлен в приложении Г.

5 Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализатор принимает на свой вход таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Также семантические проверки предусмотрены на этапе лексического анализа, а также на этапе генерации кода на язык C++.

5.2 Функции семантического анализатора

Семантический анализатор выполняет проверку на основе правил языка, описанных в п. 1.16. Он и есть та самая подпрограмма, которая занимается автоматическим приведением типов.

5.3 Структура и перечень сообщений семантического анализатора

Перечень сообщений семантического анализатора представлен в таблице 5.1.

Таблица 5.1. Перечень ошибок семантического анализатора

Код сообщения	Содержание сообщения
119	Ошибка. В стандартной библиотеке GDV-2022 такой функции нет
120	Ошибка. Знак '-' может быть использован, только перед литералом!
121	Ошибка в синтаксисе!
600	Нет закрывающей кавычки
601	Найдено неопознанное слово
602	Использованна неинициализированная переменная
603	Некорректное объявление функции
604	При инициализации функции не хватает закрывающей скобки ')'
605	При инициализации функции невозможно узнать тип данных возвращаемого значения
606	Ошибка. Повторное объявление переменной!
607	Ошибка при присвоении значения. Левый аргумент операции присваивания должен быть идентификатором!
608	Невозможно определить тип переменной, инициализированной неявно! После оператора => должны быть либо идентификатор, либо литерал
609	Использована необъявленная переменная
610	Ошибка при вызове функции! Количество параметров не совпадает

Продолжение таблица 5.1

611	При использовании лямбда-выражения el => {} не найдено ключевое слово For
612	Ошибка. В лямбда выражении в цикле For передаваемый параметр должен быть идентификатор
613	Ошибка. Неправильно употребленна лямбда функции. Она предназначена для цикла For(start, end, step, el => {})
614	Ошибка. Параметр перед входящим в цикл For лямбда выражением должен быть идентификатором или литералом
615	Ошибка. Неправильная структура цикла For
616	Ошибка. Оператор break может быть использован только в теле цикла!
617	Ошибка. После имени функции ожидалась открывающая скобка '('
618	Ошибка. Как ссылки можно инициализировать только параметры
619	Ошибка. Нельзя использовать литералы/функции в качестве передаваемого параметра, берущегося по ссылке
620	Синтаксическая ошибка. После имени функции ожидалась открывающая скобка '('
621	Использован неверные управляющий символ(доступны только \\n, \\t
622	Ошибка. Не найдена пара для косой скобки '`'
623	Ошибка. Строковый литерал может использоваться только в качестве аргумента потока вывода
624	Ошибка. Превышено значение литерала типа num
625	Ошибка. Слишком маленькое значение литерала типа num
626	Ошибка в значение литерала типа float
627	Ошибка. Превышен размер строкового литерала
628	Ошибка. Функцию типа action нельзя использовать в выражениях или передавать в качестве параметра
629	Ошибка. Точка входа в программу main может быть описана лишь один раз
630	Ошибка. Функция типа action не может ничего возвращать
631	Ошибка. Функции, тип данных которых не action, должны возвращать значение

5.4 Принцип обработки ошибок

Семантический анализатор, в случае возникновения ошибки, заносит её в протокол. Следующий этап трансляции не будет запущен при возникновении ошибки. Семантический анализ начинает проверки уже на стадии лексического

анализа, если на этой стадии обнаружены семантические ошибки – программа завершит свою работу, оповестив пользователя, где и что произошло.

5.5 Контрольный пример

Обработка ошибок семантического анализатора представлена в п. 8.4.

6 Преобразование выражений

6.1 Выражения, допускаемые языком

В языке программирования GDV-2022 выражения могут содержать вычисления целочисленных типов данных, а также допускаются вызов функций (возвращающих тип) внутри выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1 Приоритет операция языка GDV-2022

Операция	Значение приоритета
( )	4
~	3
&	2
|	1

6.2 Польская запись и принцип ее построения

Польская запись — довольно-таки полезный принцип разбора выражений. С помощью ее можно определять, как и когда будут вызываться определенные операции с их аргументами. Но так как язык GDV-2022 транслируется на C++, использование польской записи нельзя раскрыть по полной. Она используется во время семантического анализа, для проверки, чтобы процедуры не использовались в выражениях, чтобы строковые литералы не могли использоваться в выражения, чтобы, если используются побитовые операции, тип данных float приводился к типу данных num и об этом оповещали пользователя.

6.3 Программная реализация обработки выражений

Программная реализация обработки выражений представлена в приложении Е.

6.4 Контрольный пример

Пример преобразования выражения к польской записи приведен в таблице 6.2.

Таблица 6.2. Пример выражения в польской записи

Исходная строка	Результирующая строка
i = i&l|i;	i=il&i|;

7 Генерация кода

7.1 Структура генератора кода

Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится выходной файл на языке C++, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода GDV-2022 представлена на рисунке 7.1.

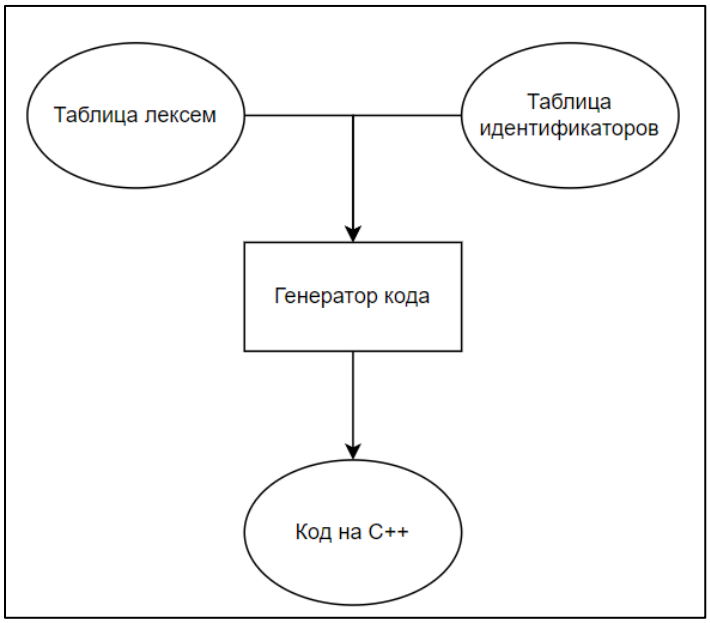


Рисунок 7.1 Структура генератора кода

7.2 Представление типов данных в оперативной памяти

Переменные, функции и все другие части кода будут написаны на языке C++ на своих местах, где они написаны на языке GDV-2022. Соответствие типов данных языка GDV-2020 с типами данных C++ представлено в таблице 7.1.

Таблица 7.1 Соответствия типов идентификаторов

Тип идентификатора	Тип идентификатора на языке C++	Пояснение
num	int	Хранит четырехбайтовое целое число
symb	char	Хранит символ из кодировки Windows-1251
float	float	Хранит четырехбайтовое вещественное число
ref num	int&	Целочисленный параметр, передающий по ссылке

Продолжение таблицы 7.1

ref symb	char&	Символьный параметр, передающийся по ссылке
ref float	float&	Вещественный параметр, передающийся по ссылке

7.3 Статическая библиотека

В языке GDV-2022 предусмотрена статическая библиотека, которая содержит функции, написанные на языке C++, приведенные в таблице 7.2. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера.

Таблица 7.2 Статическая библиотека

Идентификатор	Параметры	Возвращаемое выражение	Пояснение
sum	int/float a, int/float b	a + b	Арифметическая сумма
mult	int/float a, int/float b	a * b	Арифметическое произведение
division	int/float a, int/float b	a / b	Арифметическое деление
minus	int/float a, int/float	a – b	Арифметическая разность
For	int/float/char start, int/float/char end, int/float/char step, function<bool(int/float/char)> foo	Нет	Функция заменяющая цикл For.

Данные функции входят в стандартную библиотеку GDV-2022

7.4 Особенности алгоритма генерации кода

Алгоритм генерации объектного кода выглядит следующим образом:

- С помощью цикла for идем по таблице лексем.
- С помощью условного оператора switch разыскиваем лексемы, соответствие которым есть на языке C++.
- Если лексема равна i, нужно понять — это объявление новой переменной, или часть лямбда-выражения в цикле For, или просто использование идентификатора. Также нужно выяснить, нужно ли приводить данный идентификатор к какому-либо типу данных.
- Если лексема равна l, нужно понять, нужно ли приводить ее к другому типу данных.

- Если лексемы равны ;, или, }, или {, то это означает, что при генерации следующего кода, его надо перенести на следующую строку и добавить табуляторы.

7.5 Контрольный пример

Контрольный пример C++ кода приведен в приложении Ж. Результат генерации C++ кода приведен на рисунке 7.2.

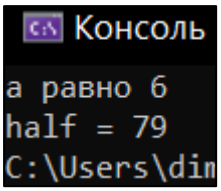


Рис.7.2 Результат генерации C++ кода

8 Тестирование транслятора

8.1 Тестирование проверки на допустимость символов

В языке GDV-2022 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 Тестирование проверки на допустимость символов

Исходный код	Диагностическое сообщение

main
{
a => ‘/’;
return 0;
}
	Ошибка: 111 : Недопустимый символ в исходном файле (-in)
	Строка: 3 Символ: 7

8.2 Тестирование лексического анализатора

Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 Тестирование лексического анализатора

Исходный код	Диагностическое сообщение

main
{
a is char = '1;
return 0;
}
	Ошибка: 600 : Нет закрывающей кавычки
	Строка: 3 Символ: 0

8.3 Тестирование синтаксического анализатора

Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 Тестирование синтаксического анализатора

Исходный код	Диагностическое сообщение

main
{
a => 23
return 0;
}
	670: строка 4, Ошибка. Не хватает точки с запятой
	670: строка 4, Ошибка. Не хватает точки с запятой
	670: строка 4, Ошибка. Не хватает точки с запятой
	Ошибка: 121 : Ошибка в синтаксисе!

8.4 Тестирование семантического анализатора

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 Тестирование семантического анализатора

Исходный код	Диагностическое сообщение

main
{
b is num = a & 23;
return 0;
}
	Ошибка: 609 : Использована необъявленная переменная
	Строка: 3 Символ: 0

Заключение

В ходе выполнения курсовой работы был разработан компилятор и генератор кода для языка программирования GDV-2022. Таким образом, были выполнены основные задачи данной курсовой работы:

- Сформулирована спецификация языка GDV-2022;
- Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
- Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
- Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
- Осуществлена программная реализация синтаксического анализатора, позволяющая расширять набор синтаксических конструкций языка только за счёт внесения изменений в разработанную грамматику;
- Разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
- Разработан транслятор кода на C++;
- Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка GDV-2022 включает:

- 4 типа данных;
- Поддержка оператора вывода;
- Возможность вызова функций стандартной библиотеки;
- Наличие 3 побитовых операторов для вычисления выражений;
- Структурированная и классифицированная система для обработки ошибок пользователя.

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении компиляторов.

Литература

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

2. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

3. 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

```
IN::IN()
{
    this->add('a', 'z', IN::T);
    this->add('A', 'Z', IN::T);
    this->add('А', 'Я', IN::T);
    this->add('а', 'я', IN::T);
    this->add('0', '9', IN::T);

    this->code[(unsigned int)' '] = IN::T;
    this->code[(unsigned int)'\t'] = IN::T;
    this->code[(unsigned int)'\n'] = IN::T;

    this->code[(unsigned int)'('] = IN::T;
    this->code[(unsigned int)')'] = IN::T;
    this->code[(unsigned int)'{'] = IN::T;
    this->code[(unsigned int)'}'] = IN::T;
    this->code[(unsigned int)';'] = IN::T;

    this->code[(unsigned int)'='] = IN::T;
    this->code[(unsigned int)'\''] = IN::T;
    this->code[(unsigned int)'\\'] = IN::T;
    this->code[(unsigned int)'>'] = IN::T;
    this->code[(unsigned int)'<'] = IN::T;
    this->code[(unsigned int)'?'] = IN::T;
    this->code[(unsigned int)'&'] = IN::T;
    this->code[(unsigned int)','] = IN::T;
    this->code[(unsigned int)'|'] = IN::T;
    this->code[(unsigned int)'~'] = IN::T;
    this->code[(unsigned int)'*'] = IN::T;
    this->code[(unsigned int)'_'] = IN::T;
    this->code[(unsigned int)'-'] = IN::T;
    this->code[(unsigned int)'.'] = IN::T;
    this->code[(unsigned int)':'] = IN::T;
    this->code[(unsigned int)'`'] = IN::T;
    this->code[(unsigned int)'%'] = IN::T;
    this->code[(unsigned int)'!'] = IN::T;
    this->code[(unsigned int)'"'] = IN::T;
    this->code[(unsigned int)'@'] = IN::T;
    this->code[(unsigned int)'#'] = IN::T;
    this->code[(unsigned int)'$'] = IN::T;
    this->code[(unsigned int)'+'] = IN::T;
}
```

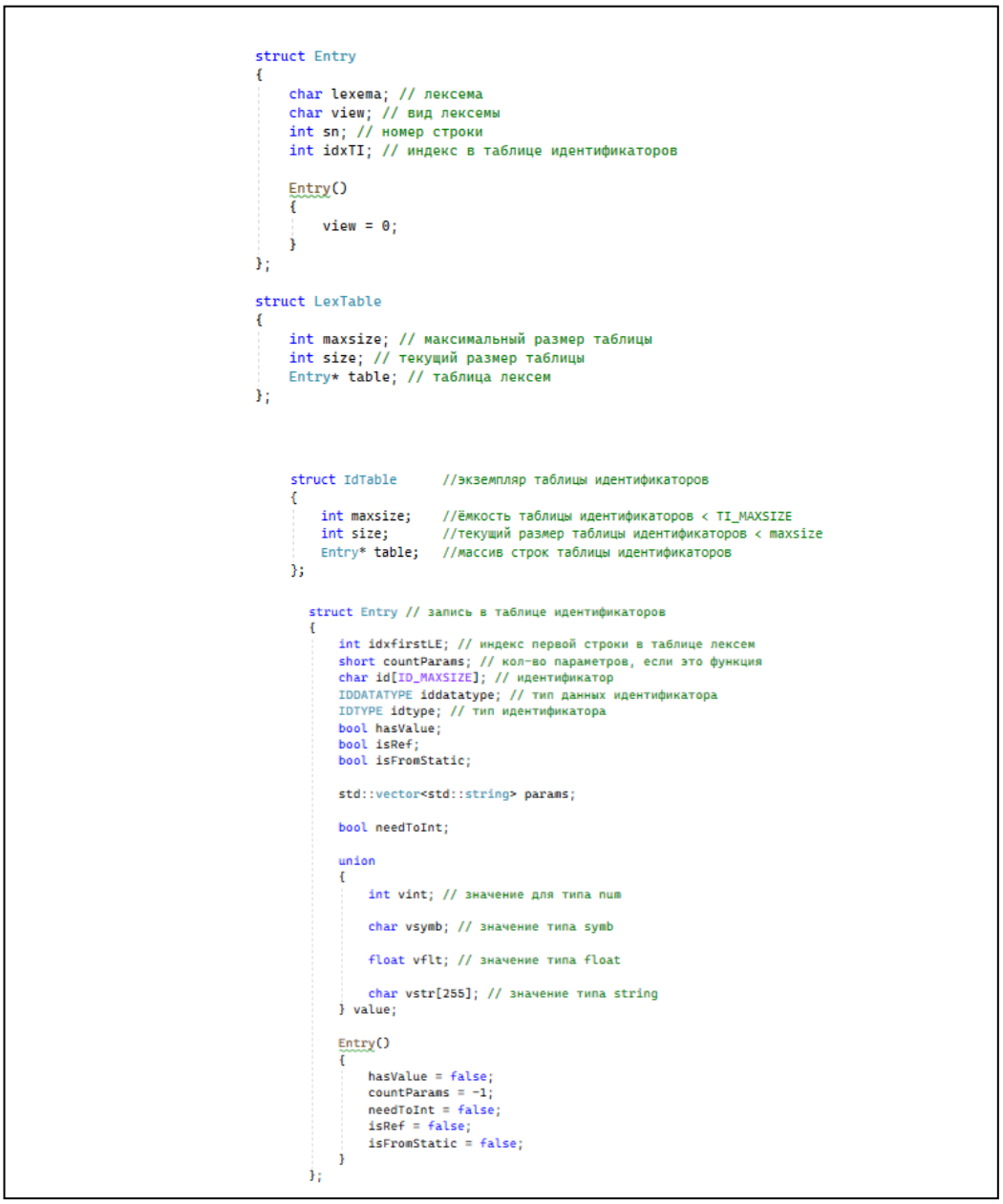
Рис. 1 Контроль входных символов

Идентификаторов	i
Литерал	l
is	s
num	t
symb	t
float	t
foo	F
=	=
‘	‘
;	;
(	(
)	)
,	,
>	V
<	V
==	V
!=	V
?	?
Truth	T
Lie	L
{	{
}	}
&	v
|	v
~	~
main	m
return	r
For	p
break	b
skip	z

Рис. 2 Лексемы языка GDV-2022

```
struct FST // недотерминированный конечный автомат
{
    char* string; // цепочка(строка завершается 0x00)
    short position; // текущая позиция в цепочке
    short nstates; // количество состояний КА
    NODE* nodes; // граф переходов: [0] – начальное состояние, [nstate-1] – конечное
    short* rstates; // возможные состояния автомата на данной позиции

    FST(
        char* s, // цепочка
        short ns, // количество состояний
        NODE n, // граф переходов/ список состояний
        ...
    );
};
```

Рис.3 Реализация конечных автоматов



```
struct Entry
{
    char lexema; // лексема
    char view; // вид лексемы
    int sn; // номер строки
    int idxTI; // индекс в таблице идентификаторов

    Entry()
    {
        view = 0;
    }
};

struct LexTable
{
    int maxsize; // максимальный размер таблицы
    int size; // текущий размер таблицы
    Entry* table; // таблица лексем
};
```

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

  struct Entry // запись в таблице идентификаторов
  {
      int idxfirstLE; // индекс первой строки в таблице лексем
      short countParams; // кол-во параметров, если это функция
      char id[ID_MAXSIZE]; // идентификатор
      IDDATATYPE iddatatype; // тип данных идентификатора
      IDTYPE idtype; // тип идентификатора
      bool hasValue;
      bool isRef;
      bool isFromStatic;

      std::vector<std::string> params;

      bool needToInt;

      union
      {
          int vint; // значение для типа num

          char vsymb; // значение типа symb

          float vflt; // значение типа float

          char vstr[255]; // значение типа string
      } value;

      Entry()
      {
          hasValue = false;
          countParams = -1;
          needToInt = false;
          isRef = false;
          isFromStatic = false;
      }
  };
```

Рис. 4 Основные структуры данных лексического анализатора

Приложение Б

```
ERROR errors[ERROR_MAX_ENTRY] = // таблица ошибок
{
    ERROR_ENTRY(0, "Недопустим код ошибки"),
    ERROR_ENTRY(1, "Системный сбой"),
    ERROR_ENTRY_NODEF(2), ERROR_ENTRY_NODEF(3), ERROR_ENTRY_NODEF(4), ERROR_ENTRY_NODEF(5),
    ERROR_ENTRY_NODEF(6), ERROR_ENTRY_NODEF(7), ERROR_ENTRY_NODEF(8), ERROR_ENTRY_NODEF(9),
    ERROR_ENTRY_NODEF10(10), ERROR_ENTRY_NODEF10(20), ERROR_ENTRY_NODEF10(30), ERROR_ENTRY_NODEF10(40), ERROR_ENTRY_NODEF10(50),
    ERROR_ENTRY_NODEF10(60), ERROR_ENTRY_NODEF10(70), ERROR_ENTRY_NODEF10(80), ERROR_ENTRY_NODEF10(90),
    ERROR_ENTRY(100, "Параметр -in должен быть задан"),
    ERROR_ENTRY_NODEF(101),ERROR_ENTRY_NODEF(102),ERROR_ENTRY_NODEF(103),
    ERROR_ENTRY(104, "Превышена длина входного параметра"),
    ERROR_ENTRY_NODEF(105),ERROR_ENTRY_NODEF(106),ERROR_ENTRY_NODEF(107),
    ERROR_ENTRY_NODEF(108),ERROR_ENTRY_NODEF(109),
    ERROR_ENTRY(110, "Ошибка при открытии файла с исходным кодом (-in)"),
    ERROR_ENTRY(111, "Недопустимый символ в исходном файле (-in)"),
    ERROR_ENTRY(112, "Ошибка при создании файла протокола (-log)"),
    ERROR_ENTRY(113, "Ошибка при создании таблицы лексем. Превышена емкость таблицы лексем"),
    ERROR_ENTRY(114, "Ошибка при добавлении лексемы в таблицу. Таблица лексем заполнена"),
    ERROR_ENTRY(115, "Ошибка при получении лексемы из таблицы. Недопустимый номер лексемы"),
    ERROR_ENTRY(116, "Ошибка при создании таблицы идентификаторов. Превышена емкость таблицы идентификаторов"),
    ERROR_ENTRY(117, "Ошибка при добавлении идентификатора в таблицу. Таблица идентификаторов заполнена"),
    ERROR_ENTRY(118, "Ошибка при получении записи из таблицы идентификаторов. Недопустимый номер идентификатора"),
    ERROR_ENTRY(119, "Ошибка. В стандартной библиотеке GDV-2022 такой функции нет"),
    ERROR_ENTRY(120, "Ошибка. Знак '-' может быть использован, только перед литералом!"),
    ERROR_ENTRY(121, "Ошибка в синтаксисе!"),
    ERROR_ENTRY_NODEF(122),
    ERROR_ENTRY_NODEF(123),
    ERROR_ENTRY_NODEF(124),
    ERROR_ENTRY_NODEF(125),
    ERROR_ENTRY_NODEF(126),
    ERROR_ENTRY_NODEF(127),
    ERROR_ENTRY_NODEF(128),
    ERROR_ENTRY_NODEF(129),
    ERROR_ENTRY_NODEF10(130),ERROR_ENTRY_NODEF10(140),ERROR_ENTRY_NODEF10(150),
    ERROR_ENTRY_NODEF10(160),ERROR_ENTRY_NODEF10(170),ERROR_ENTRY_NODEF10(180),ERROR_ENTRY_NODEF10(190),
    ERROR_ENTRY_NODEF100(200),ERROR_ENTRY_NODEF100(300),ERROR_ENTRY_NODEF100(400),ERROR_ENTRY_NODEF100(500),
    ERROR_ENTRY(600, "Нет закрывающей кавычки"),
    ERROR_ENTRY(601, "Найдено неопознанное слово"),
    ERROR_ENTRY(602, "Использованна неинициализированная переменная"),
    ERROR_ENTRY(603, "Некорректное объявление функции"),
    ERROR_ENTRY(604, "При инициализации функции не хватает закрывающей скобки ')'"),
    ERROR_ENTRY(605, "При инициализации функции невозможно узнать тип данных возвращаемого значения"),
    ERROR_ENTRY(606, "Ошибка. Повторное объявление переменной!"),
    ERROR_ENTRY(607, "Ошибка при присвоении значения. Левый аргумент операции присваивания должен быть идентификатором!"),
    ERROR_ENTRY(608, "Невозможно определить тип переменной, инициализированной неявно! После оператора => должны быть либо идентификатор, либо литерал"),
    ERROR_ENTRY(609, "Использована необъявленная переменная"),
    ERROR_ENTRY(610, "Ошибка при вызове функции! Количество параметров не совпадает"),
    ERROR_ENTRY(611, "При использовании лямбда-выражения el => {} не найдено ключевое слово For"),
    ERROR_ENTRY(612, "Ошибка. В лямбда выражении в цикле For передаваемый параметр должен быть идентификатор"),
    ERROR_ENTRY(613, "Ошибка. Неправильно употребленна лямбда функции. Она предназначена для цикла For(start, end, step, el => {})"),
    ERROR_ENTRY(614, "Ошибка. Параметр перед входящим в цикл For лямбда выражением должен быть идентификатором или литералом"),
    ERROR_ENTRY(615, "Ошибка. Неправильная структура цикла For"),
    ERROR_ENTRY(616, "Ошибка. Оператор break может быть использован только в теле цикла!"),
    ERROR_ENTRY(617, "Ошибка. После имени функции ожидалась открывающая скобка '('"),
    ERROR_ENTRY(618, "Ошибка. Как ссылки можно инициализировать только параметры"),
    ERROR_ENTRY(619, "Ошибка. Нельзя использовать литералы/функции в качестве передаваемого параметра, берущегося по ссылке"),
    ERROR_ENTRY(620, "Синтаксическая ошибка. После имени функции ожидалась открывающая скобка '('"),
    ERROR_ENTRY(621, "Использован неверные управляющий символ(доступны только \\n, \\t)"),
    ERROR_ENTRY(622, "Ошибка. Не найдена пара для косой скобки '`'"),
    ERROR_ENTRY(623, "Ошибка. Строковый литерал может использоваться только в качестве аргумента потока вывода"),
    ERROR_ENTRY(624, "Ошибка. Превышено значение литерала типа num"),
    ERROR_ENTRY(625, "Ошибка. Слишком маленькое значение литерала типа num"),
    ERROR_ENTRY(626, "Ошибка в значение литерала типа float"),
    ERROR_ENTRY(627, "Ошибка. Превышен размер строкового литерала"),
    ERROR_ENTRY(628, "Ошибка. Функцию типа action нельзя использовать в выражениях или передавать в качестве параметра"),
    ERROR_ENTRY(629, "Ошибка. Точка входа в программу main может быть описана лишь один раз"),
    ERROR_ENTRY(630, "Ошибка. Функция типа action не может ничего возвращать"),
    ERROR_ENTRY(631, "Ошибка. Функции, тип данных которых не action, должны возвращать значение"),
    ERROR_ENTRY_NODEF(632),
    ERROR_ENTRY_NODEF(633),
    ERROR_ENTRY_NODEF(634),
    ERROR_ENTRY_NODEF(635),
    ERROR_ENTRY_NODEF(636),
    ERROR_ENTRY_NODEF(637),
    ERROR_ENTRY_NODEF(638),
    ERROR_ENTRY_NODEF(639),
    ERROR_ENTRY_NODEF10(640),
    ERROR_ENTRY(650, "Ошибка в глобальной области!"),
    ERROR_ENTRY(651, "Ошибка в выражении"),
    ERROR_ENTRY(652, "Ошибка при построении контрукции параметров"),
    ERROR_ENTRY(653, "Ошибки при указании типов данных для параметров"),
    ERROR_ENTRY(654, "Ошибка при вызове функции"),
    ERROR_ENTRY(655, "Ошибка в параметрах вызываемой функции"),
    ERROR_ENTRY(656, "Ошибка в теле функции"),
    ERROR_ENTRY(657, "Ошибка в инициализации переменных"),
    ERROR_ENTRY(658, "Ошибка в использовании условного оператора"),
    ERROR_ENTRY(659, "Ошибка при возвращении значения функции"),

     ERROR_ENTRY(660, "Ошибка в при использовании все возможных типов значений"),
     ERROR_ENTRY(661, "Ошибка в теле условного оператора"),
     ERROR_ENTRY(662, "Ошибка при вызове оператора console(). Неверный синтаксис. console(Значение);"),
     ERROR_ENTRY(663, "Ошибка в искусственной области видимости!"),
     ERROR_ENTRY(664, "Ошибка в параметрах цикла"),
     ERROR_ENTRY(665, "Ошибка в теле цикла"),
     ERROR_ENTRY(666, "Ошибка при инициализации функции"),
     ERROR_ENTRY(667, "Ошибка в определении типа переменной/функции"),
     ERROR_ENTRY(668, "Ошибка в теле функции типа Action"),
     ERROR_ENTRY(669, "Ошибка при инициализации функции Action"),
     ERROR_ENTRY(670, "Ошибка. Не хватает точки с запятой"),
     ERROR_ENTRY(671, "Ошибка. Ожидалась закрывающая кругла скобка ')'"),
     ERROR_ENTRY_NODEF(672),
     ERROR_ENTRY_NODEF(673),
     ERROR_ENTRY_NODEF(674),
     ERROR_ENTRY_NODEF(675),
     ERROR_ENTRY_NODEF(676),
     ERROR_ENTRY_NODEF(677),
     ERROR_ENTRY_NODEF(678),
     ERROR_ENTRY_NODEF(679),
     ERROR_ENTRY_NODEF10(680),ERROR_ENTRY_NODEF10(690),
     ERROR_ENTRY_NODEF100(700),ERROR_ENTRY_NODEF100(800),ERROR_ENTRY_NODEF100(900)
};
```

Рис.4 Таблица ошибок языка GDV-2022

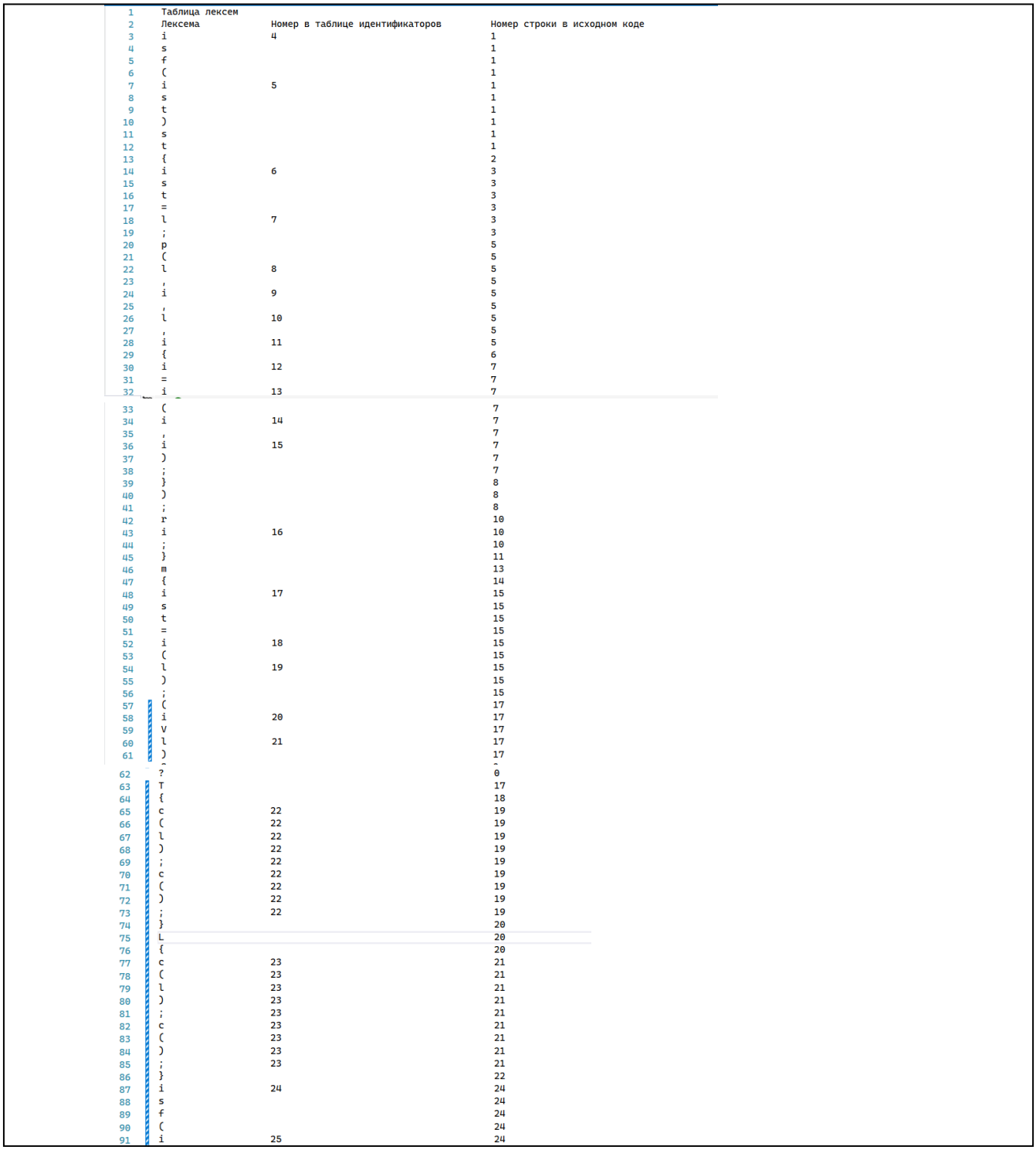
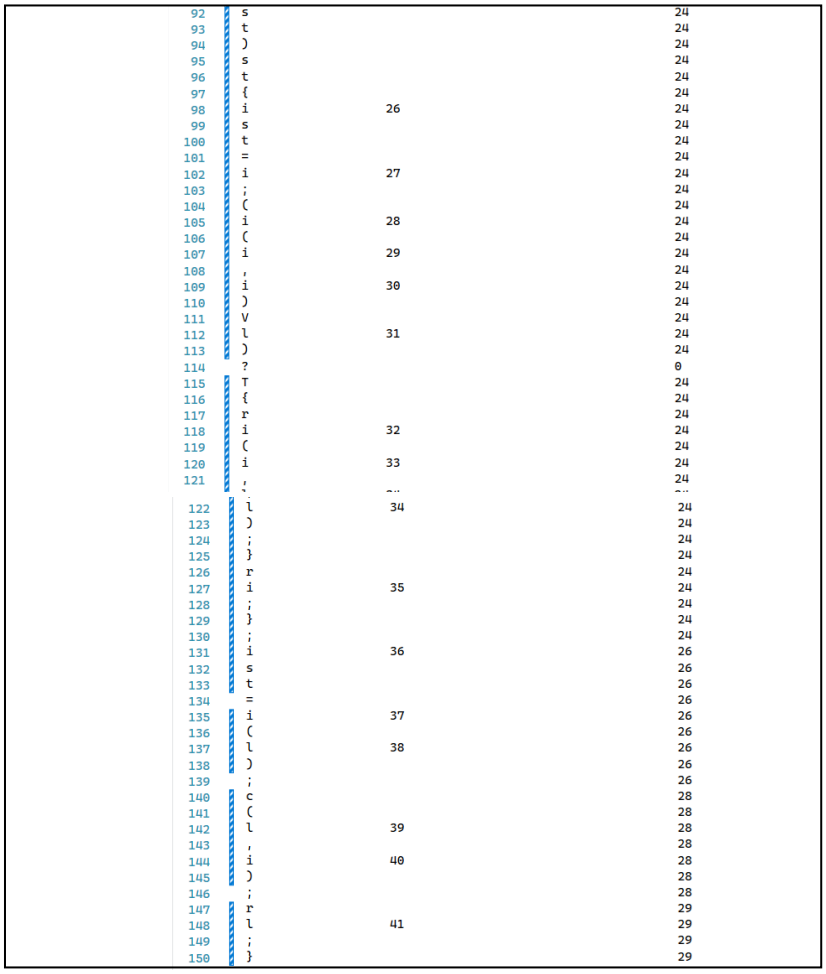


Таблица лексем

Лексема	Номер в таблице идентификаторов	Номер строки в исходном коде
i	4	1
s		1
f		1
(		1
i	5	1
s		1
t		1
)		1
s		1
t		1
{		2
i	6	3
s		3
t		3
=		3
l	7	3
;		3
p		5
(		5
l	8	5
,		5
i	9	5
,		5
l	10	5
,		5
i	11	5
{		6
i	12	7
=		7
i	13	7
(		7
i	14	7
,		7
i	15	7
)		7
;		7
}		8
)		8
;		8
r		10
i	16	10
;		10
}		11
m		13
{		14
i	17	15
s		15
t		15
=		15
i	18	15
(		15
l	19	15
)		15
;		15
(		17
i	20	17
V		17
l	21	17
)		17
~		~
?		0
T		17
{		18
c	22	19
(	22	19
l	22	19
)	22	19
;	22	19
c	22	19
(	22	19
)	22	19
;	22	19
}		20
L		20
{		20
c	23	21
(	23	21
l	23	21
)	23	21
;	23	21
c	23	21
(	23	21
)	23	21
;	23	21
}		22
i	24	24
s		24
f		24
(		24
i	25	24

Рис.5 Таблица лексем языка GDV-2022



92	s			24
93	t			24
94	)			24
95	s			24
96	t			24
97	{			24
98	i	26		24
99	s			24
100	t			24
101	=			24
102	i	27		24
103	;			24
104	(			24
105	i	28		24
106	(			24
107	i	29		24
108	,			24
109	i	30		24
110	)			24
111	V			24
112	l	31		24
113	)			24
114	?			0
115	T			24
116	{			24
117	r			24
118	i	32		24
119	(			24
120	i	33		24
121	,			24
122	l	34		24
123	)			24
124	;			24
125	}			24
126	r			24
127	i	35		24
128	;			24
129	}			24
130	;			24
131	i	36		26
132	s			26
133	t			26
134	=			26
135	i	37		26
136	(			26
137	l	38		26
138	)			26
139	;			26
140	c			28
141	(			28
142	l	39		28
143	,			28
144	i	40		28
145	)			28
146	;			28
147	r			29
148	l	41		29
149	;			29
150	}			29

Рис.5 (продолжение) Таблица лексем языка GDV-2022

Таблица идентификаторов

Идентификатор	Тип данных	Тип идентификатора	Индекс в таблице лексем	Значение
sum	float	функция	Из статической библиотеки	Не определено
minus	float	функция	Из статической библиотеки	Не определено
mult	float	функция	Из статической библиотеки	Не определено
division	float	функция	Из статической библиотеки	Не определено
Fact.$	num	функция	0	Не определено
a.Fact.$	num	параметр	4	Не определено
res.Fact.$	num	переменная	11	Не определено
literal	num	литерал	15	1
literal	num	литерал	19	1
a.Fact.$	num	параметр	21	Не определено
literal	num	литерал	23	1
el.$For1.Fact.$	num	параметр	25	Не определено
res.Fact.$	num	переменная	27	Не определено
mult	float	функция	29	Не определено
res.Fact.$	num	переменная	31	Не определено
el.$For1.Fact.$	num	параметр	33	Не определено
res.Fact.$	num	переменная	40	Не определено
a.$l1.main.$	num	переменная	45	Не определено
Fact.$	num	функция	49	Не определено
literal	num	литерал	51	3
a.$l1.main.$	num	переменная	55	Не определено
literal	num	литерал	57	6
literal	string	литерал	64	a равно 6
literal	string	литерал	76	a не равно 6
round.$l1.main.$	float	функция	84	Не определено
a.round.$l1.main.$	float	параметр	88	Не определено
numa.round.$l1.main.$	num	переменная	95	Не определено
a.round.$l1.main.$	float	параметр	99	Не определено
minus	float	функция	102	Не определено
a.round.$l1.main.$	float	параметр	104	Не определено
numa.round.$l1.main.$	num	переменная	106	Не определено
literal	float	литерал	109	0,500000
sum	float	функция	115	Не определено
numa.round.$l1.main.$	num	переменная	117	Не определено
literal	num	литерал	119	1
numa.round.$l1.main.$	num	переменная	124	Не определено
half.$l1.main.$	float	переменная	128	Не определено
round.$l1.main.$	float	функция	132	Не определено
literal	float	литерал	134	78,669998
literal	string	литерал	139	half =
half.$l1.main.$	float	переменная	141	Не определено
literal	num	литерал	145	0

Рис.7 Таблица идентификаторов языка GDV-2022

Приложение В

```
     GRB::Greibach greibach(
          NS(GLOBAL),
          TS('$'),
          22, // edit

          Rule(NS(GLOBAL),  GRB_ERROR_SERIES  +  0,  //  глобальное
пространство
               14,

               Rule::Chain(3, TS('i'), NS(INIT_FUNC), NS(GLOBAL)),
               Rule::Chain(2, TS('i'), NS(INIT_FUNC)),

               Rule::Chain(3, TS('i'), NS(INIT_ACTION), NS(GLOBAL)),
               Rule::Chain(2, TS('i'), NS(INIT_ACTION)),

               Rule::Chain(6,   TS('i'),   NS(DEF_TYPE),   TS('='),
NS(EXPR), NS(SEMICOLON), NS(GLOBAL)),
               Rule::Chain(5,   TS('i'),   NS(DEF_TYPE),   TS('='),
NS(EXPR), NS(SEMICOLON)),

               Rule::Chain(4,  TS('i'),  NS(DEF_TYPE),  NS(SEMICOLON),
NS(GLOBAL)),
               Rule::Chain(3, TS('i'), NS(DEF_TYPE), NS(SEMICOLON)),

               Rule::Chain(5,     TS('i'),     TS('='),     NS(EXPR),
NS(SEMICOLON), NS(GLOBAL)),
               Rule::Chain(4,     TS('i'),     TS('='),     NS(EXPR),
NS(SEMICOLON)),

               Rule::Chain(5, TS('m'), TS('{'), NS(BODY_FUNC),
```

Рис. 8 Структура Грамматики Грейбах

```
          Rule::Chain(4,   TS('i'),   NS(DEF_TYPE),   NS(SEMICOLON),
NS(FOR_BODY)),
               Rule::Chain(3, TS('i'), NS(DEF_TYPE), NS(SEMICOLON)),

               Rule::Chain(6,   TS('i'),   NS(DEF_TYPE),   TS('='),
NS(EXPR), NS(SEMICOLON), NS(FOR_BODY)),
               Rule::Chain(5,   TS('i'),   NS(DEF_TYPE),   TS('='),
NS(EXPR), NS(SEMICOLON)),

               Rule::Chain(5,    TS('i'),    TS('='),    NS(EXPR),
NS(SEMICOLON), NS(FOR_BODY)),
               Rule::Chain(4,    TS('i'),    TS('='),    NS(EXPR),
NS(SEMICOLON)),

               Rule::Chain(4,   TS('{'),   NS(FOR_BODY),   TS('}'),
NS(FOR_BODY)),
               Rule::Chain(3, TS('{'), NS(FOR_BODY), TS('}')),
               Rule::Chain(2, TS('{'), TS('}')),
               Rule::Chain(3, TS('{'), TS('}'), NS(FOR_BODY)),

               Rule::Chain(4, TS('i'), NS(INIT_FUNC), NS(SEMICOLON),
NS(FOR_BODY)),
               Rule::Chain(3, TS('i'), NS(INIT_FUNC), NS(SEMICOLON)),

               Rule::Chain(4,  TS('c'),  NS(CONSOLE),  NS(SEMICOLON),
NS(FOR_BODY)),
               Rule::Chain(3, TS('c'), NS(CONSOLE), NS(SEMICOLON)),

               Rule::Chain(3, TS(FOR), NS(FOR_PARAM), NS(FOR_BODY)),
               Rule::Chain(2, TS(FOR), NS(FOR_PARAM)),

               Rule::Chain(8,  TS('('),  NS(TYPES_VALUES),  TS('V'),
NS(TYPES_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY), NS(FOR_BODY)),
               Rule::Chain(7,  TS('('),  NS(TYPES_VALUES),  TS('V'),
NS(TYPES_VALUES), NS(RIGHTSCOPE), TS('?'), NS(IFBODY)),

               Rule::Chain(2, TS('b'), NS(SEMICOLON)),
               Rule::Chain(3, TS('b'), NS(SEMICOLON), NS(FOR_BODY)),

               Rule::Chain(2, TS(SKIP), NS(SEMICOLON),
               Rule::Chain(3, TS(SKIP), NS(SEMICOLON), NS(FOR_BODY))
               )
          )
);
```

Листинг 8 (продолжение) Структура Грамматики Грейбах

Приложение Г

```
1    Шаг  : Правило                  Входная лента                  Стек
2    0    : A->iTA                   isf(ist)st{ist=l;p(l,i,l,      A$
3    0    : SAVESTATE:               1
4    0    :                          isf(ist)st{ist=l;p(l,i,l,      iTA$
5    1    :                          sf(ist)st{ist=l;p(l,i,l,i      TA$
6    2    : T->sfCU{G}               sf(ist)st{ist=l;p(l,i,l,i      TA$
7    2    : SAVESTATE:               2
8    2    :                          sf(ist)st{ist=l;p(l,i,l,i      sfCU{G}A$
9    3    :                          f(ist)st{ist=l;p(l,i,l,i{      fCU{G}A$
10   4    :                          (ist)st{ist=l;p(l,i,l,i{i      CU{G}A$
11   5    : C->(Y                    (ist)st{ist=l;p(l,i,l,i{i      CU{G}A$
12   5    : SAVESTATE:               3
13   5    :                          (ist)st{ist=l;p(l,i,l,i{i      (YU{G}A$
14   6    :                          ist)st{ist=l;p(l,i,l,i{i=      YU{G}A$
15   7    : TNS_NORULECHAIN/NS_NORULE
16   7    : RESTATE
17   7    :                          (ist)st{ist=l;p(l,i,l,i{i      CU{G}A$
18   8    : C->(DY                   (ist)st{ist=l;p(l,i,l,i{i      CU{G}A$
19   8    : SAVESTATE:               3
```

Рис. 125 Разбор исходного кода синтаксическим анализатором

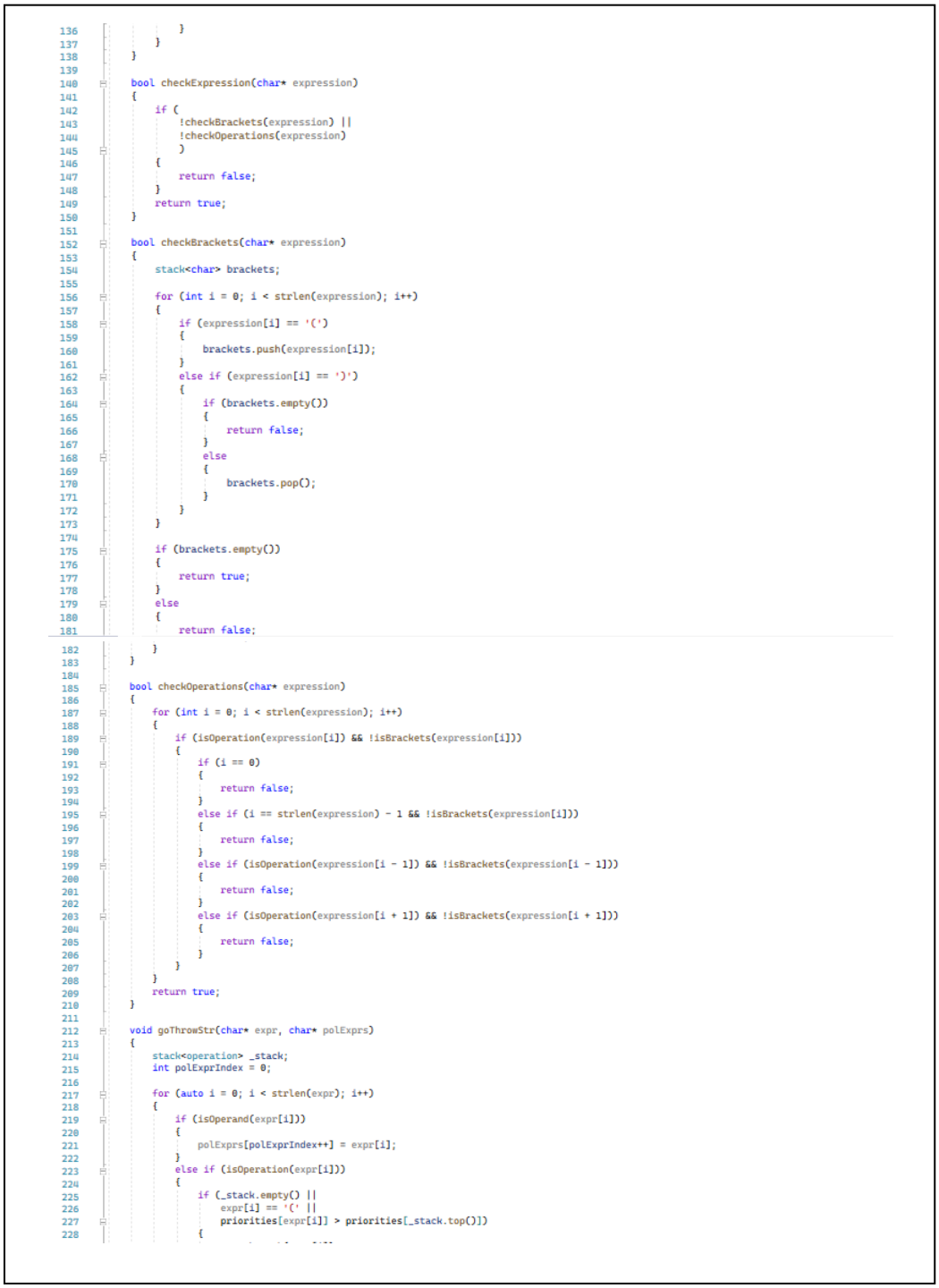
```
3098  1420:                    i);rl;}                  iYXJ}A$
3099  1421:                    );rl;}                   YXJ}A$
3100  1422: Y->)               );rl;}                   YXJ}A$
3101  1422: SAVESTATE:         97
3102  1422:                    );rl;}                   )XJ}A$
3103  1423:                    ;rl;}                    XJ}A$
3104  1424: X->;              ;rl;}                    XJ}A$
3105  1424: SAVESTATE:         98
3106  1424:                    ;rl;}                    ;J}A$
3107  1425:                    rl;}                     J}A$
3108  1426: J->rBX            rl;}                     J}A$
3109  1426: SAVESTATE:         99
3110  1426:                    rl;}                     rBX}A$
3111  1427:                    l;}                      BX}A$
3112  1428: B->l               l;}                      BX}A$
3113  1428: SAVESTATE:         100
3114  1428:                    l;}                      lX}A$
3115  1429:                    ;}                       X}A$
3116  1430: X->;              ;}                       X}A$
3117  1430: SAVESTATE:         101
3118  1430:                    ;}                       ;}A$
3119  1431:                    }                        }A$
3120  1432:                                             A$
3121  1433: LENTA_END
3122  1434: ------>LENTA_END
```

Рис. 125 (продолжение) Разбор исходного кода синтаксическим анализатором(конец)

Приложение Е

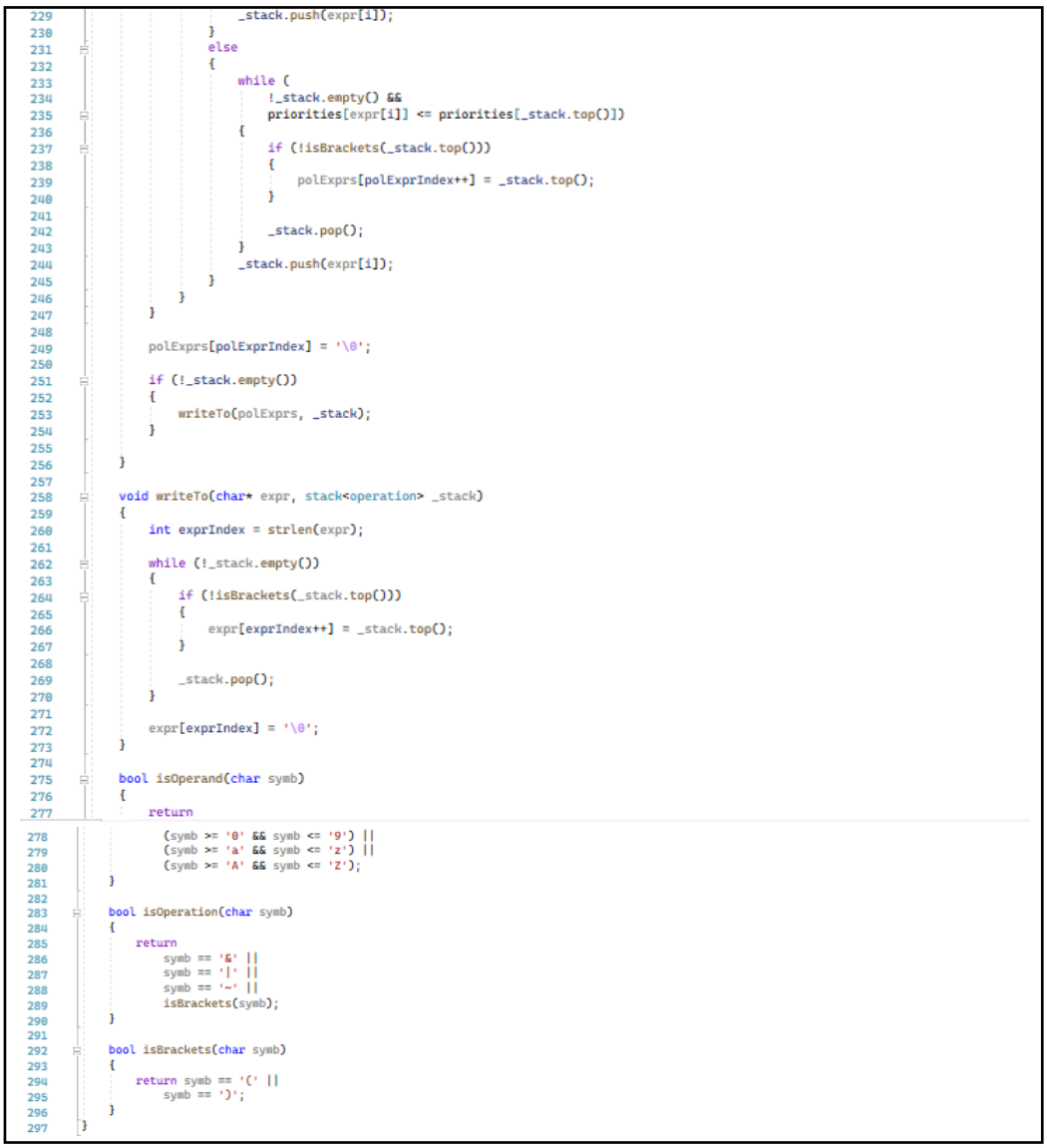
```
namespace PN
{
    typedef char operation;
    void go_to_pol(char* expr, map<char, IT::Entry*> dict, LT::LexTable& lextable, IT::IdTable& idtable);
    bool Polish(LT::LexTable& lextable, IT::IdTable& idtable)
    {
        auto expr = new char[200];
        auto cls = (char*)"";
        int size;
        int index;
        char* polExprs;
        char el;
        map<char, IT::Entry*> dict;
        IT::Entry* ide;
        LT::Entry* lte;
        auto table = lextable.table;

        bool isFunc;
        int cntScopes;

        for (int i = 0; i < lextable.size; i++)
        {
            if (table[i].lexema == '=' || table[i].lexema == 'r')
            {
                i++;
                strcpy_s(expr, cls);
                dict.clear();
                size = 0;
                isFunc = false;
                cntScopes = 0;
                el = 'a';

                while (table[i].lexema != ';')
                {
                    if (table[i].lexema == '(' && table[i - 1].lexema == 'i')
                    {
                        isFunc = true;
                        cntScopes = 0;
                    }
                    if (!isFunc)
                    {
                        if (table[i].lexema != 'i' && table[i].lexema != 'l')
                        {
                            expr[size++] = (table[i].lexema == 'v' ? table[i].view : table[i].lexema);
                        }
                        else
                        {
                            expr[size++] = el++;
                            dict[el - 1] = &idtable.table[table[i].idxTI];
                        }
                    }

                    if (table[i].lexema == '(' && isFunc)
                    {
                        cntScopes++;
                    }
                    else if (table[i].lexema == ')' && isFunc)
                    {
                        cntScopes--;

                        if (cntScopes == 0)
                        {
                            isFunc = false;
                        }
                    }
                    i++;
                }
                expr[size] = 0;

                polExprs = PolishNotation(expr);

                go_to_pol(polExprs, dict, lextable, idtable);
            }
        }

        delete[] expr;

        return true;
    }

    map<operation, int> priorities =
    {
        {'(', 1},
        {')', 1},
        {'|', 2},
        {'&', 3},
        {'~', 4}
    };

    char* PolishNotation(char* expression)
    {
        /*if (!checkExpression(expression))
        {
            return false;
        }*/

        char* polishExpression = new char[strlen(expression)];
        strcpy_s(polishExpression, strlen(expression), "");

        goThrowStr(expression, polishExpression);

        return polishExpression;
    }

    void go_to_pol(char* expr, map<char, IT::Entry*> dict, LT::LexTable& lextable, IT::IdTable& idtable)
    {
        ushort len = strlen(expr);
        IT::Entry* edi;
        for (ushort i = 0; i < len; i++)
        {
            if (!isOperation(expr[i]))
            {
                edi = dict[expr[i]];

                if (edi->iddatatype == IT::FLT && len > 1)
                {
                    edi->needToInt = true;
                    cout << "| Предупреждение: переменная/функция/литерал ";
                    if (edi->idtype == IT::L)
                    {
                        cout << edi->value.vflt;
                    }
                    else
                    {
                        cout << edi->id;
                    }
                    cout << " приведена к типу пшв при выполение побитовых операций! Строка: " << lextable.table[edi->idxfirstLE].sn << " |\n";
                }
                else if (edi->iddatatype == IT::ACTION)
                {
                    throw ERROR_THROW_IN(628, lextable.table[edi->idxfirstLE].sn, -1);
                }
```

Рис. 1 Преобразование выражений к польской записи



```
        }
    }
}

bool checkExpression(char* expression)
{
    if (
        !checkBrackets(expression) ||
        !checkOperations(expression)
        )
    {
        return false;
    }
    return true;
}

bool checkBrackets(char* expression)
{
    stack<char> brackets;

    for (int i = 0; i < strlen(expression); i++)
    {
        if (expression[i] == '(')
        {
            brackets.push(expression[i]);
        }
        else if (expression[i] == ')')
        {
            if (brackets.empty())
            {
                return false;
            }
            else
            {
                brackets.pop();
            }
        }
    }

    if (brackets.empty())
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool checkOperations(char* expression)
{
    for (int i = 0; i < strlen(expression); i++)
    {
        if (isOperation(expression[i]) && !isBrackets(expression[i]))
        {
            if (i == 0)
            {
                return false;
            }
            else if (i == strlen(expression) - 1 && !isBrackets(expression[i]))
            {
                return false;
            }
            else if (isOperation(expression[i - 1]) && !isBrackets(expression[i - 1]))
            {
                return false;
            }
            else if (isOperation(expression[i + 1]) && !isBrackets(expression[i + 1]))
            {
                return false;
            }
        }
    }
    return true;
}

void goThrowStr(char* expr, char* polExprs)
{
    stack<operation> _stack;
    int polExprIndex = 0;

    for (auto i = 0; i < strlen(expr); i++)
    {
        if (isOperand(expr[i]))
        {
            polExprs[polExprIndex++] = expr[i];
        }
        else if (isOperation(expr[i]))
        {
            if (_stack.empty() ||
                expr[i] == '(' ||
                priorities[expr[i]] > priorities[_stack.top()])
            {
```

Рис. 1 (Продолжение) Преобразование выражений к польской записи



```
                    _stack.push(expr[i]);
                }
                else
                {
                    while (
                        !_stack.empty() &&
                        priorities[expr[i]] <= priorities[_stack.top()])
                    {
                        if (!isBrackets(_stack.top()))
                        {
                            polExprs[polExprIndex++] = _stack.top();
                        }

                        _stack.pop();
                    }
                    _stack.push(expr[i]);
                }
            }
        }

        polExprs[polExprIndex] = '\0';

        if (!_stack.empty())
        {
            writeTo(polExprs, _stack);
        }

    }

    void writeTo(char* expr, stack<operation> _stack)
    {
        int exprIndex = strlen(expr);

        while (!_stack.empty())
        {
            if (!isBrackets(_stack.top()))
            {
                expr[exprIndex++] = _stack.top();
            }

            _stack.pop();
        }

        expr[exprIndex] = '\0';
    }

    bool isOperand(char symb)
    {
        return
            (symb >= '0' && symb <= '9') ||
            (symb >= 'a' && symb <= 'z') ||
            (symb >= 'A' && symb <= 'Z');
    }

    bool isOperation(char symb)
    {
        return
            symb == '&' ||
            symb == '|' ||
            symb == '~' ||
            isBrackets(symb);
    }

    bool isBrackets(char symb)
    {
        return symb == '(' ||
            symb == ')';
    }
}
```

Рис. 1 (Продолжение) Преобразование выражений к польской записи

Приложение Ж

Таблица 1. Результат генерации кода

```
     #include <iostream>
     #include <windows.h>

     #pragma    comment(lib,    "C:\\Users\\dimag\\OneDrive\\Рабочий
стол\\Курсач КПО\\GDV-2022\\Debug\\LIB.lib")
     #include    "C:\\Users\\dimag\\OneDrive\\Рабочий    стол\\Курсач
КПО\\LIB\\framework.h"


     int Fact_$(int a_Fact_$)
     {
          int res_Fact_$ = 1;
          For((int)1, (int)a_Fact_$, 1, [&](int el_$For1_Fact_$)
          {
               res_Fact_$           =           mult((int)res_Fact_$,
(int)el_$For1_Fact_$);


               return false;
          });
          return (int)(res_Fact_$);

     }
     int main()
     {
          SetConsoleCP(1251);
          SetConsoleOutputCP(1251);

          int a_$l1_main_$ = Fact_$((int)3);
          if (a_$l1_main_$ == 6)
          {
               std::cout << "a равно 6";
               std::cout << '\n';

          }
          else
          {
               std::cout << "a не равно 6";
               std::cout << '\n';

          }
          auto round_$l1_main_$ = [&](float a_round_$l1_main_$)
          {
               int numa_round_$l1_main_$ = a_round_$l1_main_$;
               if                   (minus((float)a_round_$l1_main_$,
(float)numa_round_$l1_main_$) > 0.5)
               {
                    return    (float)(sum((int)numa_round_$l1_main_$,
(int)1));
```

Продолжение таблицы 1

```
                }
                return (float)(numa_round_$l1_main_$);

        };
        float half_$l1_main_$ = round_$l1_main_$((float)78.67);
        std::cout << "half = " << half_$l1_main_$;
        return (int)(0);

    }
```