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

Факультет Информационных Технологий
Кафедра Информационных систем и технологий
Специальность 6-05-0612-01 “Программная инженерия”

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

«Разработка компилятора FIA – 2024»

Выполнил студент Филипюк Илья Андреевич
(Ф.И.О.)

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

Заведующий кафедрой к. т. н. доц. Смелов Владимир Владиславович
(учен. степень, звание, должность, подпись, Ф.И.О.)

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

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

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

Курсовой проект защищен с оценкой

Минск 2024

Содержание

Введение

Задачей данного курсового проекта является разработка транслятора для своего языка программирования FIA-2024 и реализация компилятора. Написание транслятора будет осуществляться на языке C++, при этом код на языке FIA-2024 будет транслироваться в язык ассемблера.

Транслятор FIA-2024 состоит из следующих частей:

1. семантический анализатор;
2. синтаксический анализатор;
3. логический анализатор;
4. генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

1. разбработка спецификации языка программирования;
2. разбработка структуры транслятора;
3. разработка лексического и семантического анализаторов;
4. разработка синтаксического анализатора;
5. преобразование выражений;
6. генерация кода на язык ассемблера;
7. тестирование транслятора.

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

1 Спецификация языка программирования

1.1 Характеристика языка программирования

Язык программирования FIA-2024 – это универсальный язык высокого уровня. Он является процедурным, компилируемым, не объектно-ориентированным. Язык строго типизируемый, что говорит о невозможности преобразования типов, транслируемым языком программирования.

1.2 Определение алфавит языка программирования

При написании программы на языке FIA-2024 используется таблица символов Windows-1251. На этапе выполнения могут использоваться символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, спецсимволы, а также непечатные символы пробела, табуляции и перевода строки.

Таблица 1.1 – Алфавит языка программирования FIA-2024

<строчная буква латинского алфавита>::= 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
<прописная буква латинского алфавита>::= 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
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<символ- сепаратор>::= ' '|,|(|)|{|}|;|”|’|=|+|-|*||

1.3 Применяемые сепараторы

Символы-сепараторы служат в качестве разделителей операций языка. Сепараторы, используемые в языке программирования FIA-2024, приведены в таблице 1.2.

Таблица 1.2 – Сепараторы

Сепаратор	Название	Область применения
‘ ‘	Пробел	Допускается везде, кроме идентификаторов и ключевых слов
;	Точка с запятой	Разделение конструкций
{…}	Фигурные скобки	Заключение программного блока
[…]	Квадратные кавычки	Блок кода
(…)	Круглые скобки	Приоритет операций, параметры функции
“…”	Двойные кавычки	Строковый литерал
‘…’	Одинарные кавычки	Допускается везде, кроме идентификаторов и ключевых слов
=	Знак «равно»	Присваивание значения
,	Запятая	Разделение параметров

+ -	Знак «плюс», знак «минус»,	Знаки математических операций, а именно сложение и вычитание. Допускаются только в математических операциях, между идентификаторами или числовыми константами.
== ! < >	Двойное «равно», «восклицательный знак», знак «меньше», знак «больше»	Допускаются в логических или тернарных операторах.
>>	Двойное «больше»	Сдвиг вправо
<<	Двойное «меньше»	Сдвиг влево

1.4 Применяемые кодировки

Для написания исходного кода на языке программирования FIA-2024 используется кодировка Windows-1251.

Содержимое таблицы символов представлено на рисунке 1.1.

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

Рисунок 1.1 – Таблица кодировки Windows-1251

1.5 Типы данных

В языке FIA-2024 реализованы три типа данных: беззнаковый целый (unt), символьный(sym) и логический (bool). Описание типов данных, предусмотренных в данным языке представлено в таблице 1.3.

Таблица 1.3 – Типы данных языка FIA-2024

Тип данных	Описание типа данных
unt	Беззнаковый целый тип данных. Используется для работы с целочисленными значениями. Знаковый тип. В памяти занимает 2 байта. При попытке инициализации значением больше максимального, инициализируется максимальным. При попытке инициализации значением меньше минимального, инициализируется минимальным. Инициализация по умолчанию: 0. Максимальное значение: 65535. Минимальное значение: 0 Применяемые операции: + (бинарный) – сложение; - (бинарный) – вычитание; >> - сдвиг вправо << - сдвиг влево
sym	Символьный тип данных. Используется для работы с символом, который в памяти занимает 4 байт. Сохраняет символы из набора символов ASCII. Допустимый диапазоны значений: от 0 до 255. Инициализация по умолчанию: 0.
bool	Логический тип данных. Фундаментальный тип данных. Используемся для работы с логическими значениями: истина (true) и ложь (false). Переменная этого типа может иметь значения true и false. Занимает в памяти 4 байт. Инициализация по умолчанию: 0.

Пользовательские типы данных не поддерживаются.

1.6 Преобразование типов данных

Преобразование типов, данных не поддерживается, т.е. язык является строготипизированным.

1.7 Идентификаторы

В имени идентификатора допускаются только символы латинского алфавита, знак «_» и цифры, но имя идентификатора должно начинаться с буквы латинского алфавита. Максимальная длина имени идентификатора – 50 символов. При вводе идентификатора длиной, более разрешенного количества символов, он будет усекаться. Имя идентификатора не может совпадать с именем функции, уже содержащаяся в стандартной библиотеке.

Регулярное выражение для записи идентификатора [a-z|A-Z]+ [a-z|A-Z|0-9|_]*

Пример правильного идентификатора: create unt a1;

Пример неправильного идентификатора: create int ff+фф:

1.8 Литералы

С помощью литералов осуществляется инициализация переменных. В языке существует три типа литералов. Краткое описание литералов языка FIA-2024 представлено в таблице 1.4.

Таблица 1.4 – Описание литералов

Тип литерала	Регулярное выражение	Описание	Пример

Беззнаковый целый литерал	[0|1]+
	- для двоичной системы
	[0-9| a-f|A-F]{4}h
	- для шестнадцатеричной системы
		Беззнаковые целые литералы не имеют дробных частей или экспонент. Представлены двоичной и шестнадцатеричной системах счисления. В шестнадцатеричной системе счисления символы a-f (A-F) представляют числа от 10 до 15 соответственно. Шестнадцатеричные литеры заканчиваются постфиксом h.	create unt sum = 1001;
			1001 – целый беззнаковый литерал.
			unt=00A4h;
			00A4h – беззнаковый целый литерал

Символьный литерал	[a-z|A-Z|0-9|!-/]	Символ, заключённый в '' (одинарные кавычки). Литералы могут быть только rvalue.	create sym symbol = 'T'; T – символьный литерал.

Логический Литерал	[true|false]	Логический литерал имеет два возможных значения: true (истина) и false (ложь)	create bool check = true; true – логический литерал

Литералы являются константами и при генерации кода объявляются один раз.

1.9 Объявление данных

Для объявления переменной указывается ключевое слово create, тип данных и имя идентификатора. Допускается инициализация при объявлении.

Пример объявления беззнакового целого типа данных:

create unt num1 ;

Пример объявления символьного типа данных:

create sym num2 ;

Пример объявления логического типа данных:

create bool num3 ;

Для объявления функций и используется ключевое слово function, перед которым указывается тип данных, возвращаемый функцией, а после ключевого слово идут скобки «()», с параметрами внутри (если есть).

Пример объявления функции:

unt function sum (unt a, unt b)

1.10 Инициализация данных

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Инициализация по умолчанию у всех типов данных 0. Описание способов инициализации переменных языка FIA-2024 представлено в таблице 1.5.

Таблица 1.5 – Способы инициализации переменных

Конструкция	Описание	Пример

<идентификатор> = <значение>;	Присваивание переменной значения.	sum = sum;
		chr = 'D';
		is_true= false;

create <тип данных><идентификатор> = <значение>;	Объявление переменной и присвоение ей значения.	create unt number = 1011001;
		create bool check = true;
		create sym znak= '*';

Объектами-инициализаторами могут быть идентификаторы, литералы, ключевые слова true и false для логического типа данных или функции, возвращающие советующий тип данных. Соответствие типов проверяется на синтаксическом анализе.

1.11 Инструкции языка

Все возможные инструкции языка программирования FIA-2024 представлены в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования FIA-2024

Инструкция	Запись на языке FIA-2024	Пример
Объявление переменной	create <тип данных> <идентификатор>;	create unt students;
Объявление функции	<тип данных> func <идентификатор> (<тип данных> <идентификатор>, …) {<блок кода>}	Unt function sum(unt a, unt b) { return a+b};
Объявление и инициализация переменной	create <тип данных> <идентификатор> = <литерал>/<идентификатор>;	create bool is_book= false;
Инициализация	<идентификатор> = <выражение>; Выражением может быть идентификатор, литерал, ключевые слова true и false, для логического типа данных, или вызов функции соответствующего типа. Для беззнакового целого типа выражение может быть дополнено арифметическими операциями с любым количеством операндов с использованием скобок. Для символьного типа выражение может быть только идентификатором, литералом или вызовом функции, возвращающей значение символьного типа.	students = 1+1;

Блок тела функции
Или блок кода
	{
	…
	}
		{
		create unt number;
		}
		
		{sym=’y’;};

Возврат из подпрограммы	return <идентификатор> / <литерал>;	return 0;

Оператор цикла	while(<условие>)[<блок кода>];	while(a ! 5) [write a;];
Вывод данных с переходом на новую строку	writeline <идентификатор> / <литерал>;	writeline a;

Инструкции (кроме функции блока тела функции) требуют закрывающую «;».

1.12 Операции языка

Язык программирования FIA-2024 может выполнять операции, представленные в таблице 1.6. Если у операций одинаковый приоритет, то первой будет выполнена операция, стоящая левее. С помощью круглых скобок может быть изменен приоритет операций. Операции выполняются с права налево. Более подробное про приоритет арифметический операций написано в разделе 6.1

Таблица 1.6 – Операции языка программирования FIA-2024

Операция	Примечание	Типы данных	Пример
( )	Имеют самый высокий приоритет и используются для явного указания порядка выполнения операций	(unt, unt)	sum = (a + b) * c;
+	Суммирование	(unt, unt)	sum = a + b;
-	Вычитание	(unt, unt)	diff = a – b;
<,>	Знаки «больше», «меньше» для условной инструкции	(unt, unt)	while(sum < diff) […]; while(sum > diff) […];
<<	Сдвиг влево	(unt, unt)	num<<10
>>	Сдвиг вправо	(unt, unt)	Num>>10
==	Оператор эквивалентности	(unt, unt)	while(sum == diff) […];
!	Оператор неравенства	(unt, unt)	while(sum ! diff) […];

1.13 Выражения и их вычисления

В языке FIA-2024 предусмотрены арифметические, сдвиговые, сравнительные и комбинаторные выражения. Арифметические операции включают в себя операции сложения и вычитания. Для арифметических операций используется беззнаковый целый тип данных. Сравнительные выражения включают в себя операции сравнения, такие как равно, не равно, больше, меньше. Для сравнительных операций можно использовать только целочисленный тип данных. Комбинированные выражения: включают комбинации различных типов выражений.

Предусмотрены следующие правила составления выражений:

1. Рассматриваются слева направо.
2. Для изменения приоритета операции используются круглые скобки ()
3. Каждое выражение должно заканчиваться сепаратором
4. Не допускается запись двух подряд идущих арифметических операций.
5. Выражения вычисляются только после оператора присваивания.

Перед генерацией кода каждое выражение приводится к записи в польской записи для удобства дальнейшего вычисления выражения на языке ассемблера.

1.14 Программные конструкции языка

Ключевые программные конструкции языка программирования FIA-2024 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка FIA-2024

Конструкция	Запись на языке FIA-2024

Главная функция (точка входа)	main
	{
	…
	return <идентификатор> / <литерал>;
	}

Функция	<тип данных> func <идентификатор> (<тип> <идентификатор>, …) {… return <идентификатор> / <литерал>; };
Цикл	while(<выражение>)[…]; Выражение может быть просто переменной логического типа данных либо любым логическим выражением, которое возвращает true или false.

Программные конструкции языка FIA-2024 представляют собой базовый функционал для выполнения различных операций, что делает возможным решать задачи различного уровня.

1.15 Область видимости

Область видимости «сверху вниз». В языке FIA-2024 требуется обязательное объявление переменной перед её инициализацией и последующим использованием. Все переменные должны находиться внутри программного блока. Имеется возможность объявления одинаковых переменных в разных блоках, т. к. переменные, объявленные в одной функции, недоступны в другой. Объявление функций стандартной библиотеки можно производить в любом месте кода.

1.16 Семантические проверки

Назначение семантического анализа – проверка смысловой правильности конструкций языка программирования. Таблица с перечнем семантических проверок, предусмотренных языком, приведена в таблице 1.8.

Таблица 1.8 – Семантические проверки

Номер	Правило
1	Функция main может быть объявлена только один раз
2	Функция main может возвращать только целочисленное значение
3	Все переменные и функции должны быть объявлены до их использования
4	Переменные могут инициализировать во время объявления или после него
5	Тип возвращаемого значения должен совпадать с типом функции при её объявлении.
6	Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении.
7	В функцию должно быть передано то число параметров, сколько ожидается.
8	Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается.
9	Математические операции разрешены только с целочисленными операнды.
10	Тип данных присваиваемой переменной должен совпадать с типом данных переменной которой присваивается.

Если семантическая проверка не проходит, то в лог журнал записывается соответствующая ошибка и выводится на консоль.

1.17 Распределение оперативной памяти на этапе выполнения

Все переменные размещаются в стеке.

1.18 Стандартная библиотека и её состав

Стандартная библиотека FIA-2024 написана на языке программирования C++.

Функции стандартной библиотеки не требуют явного с помощью ключевых слов, работа с ними производится как с пользовательскими функциями. Функции стандартной библиотеки с описанием представлены в таблице 1.9.

Таблица 1.9 – Состав стандартной библиотеки

Функция(C++)	Возвращаемое значение	Описание

Void get_date()	none	Выводит в консоль текущее время (на момент вызова) вместе с текущей датой
Void get_time()	none	Выводит в консоль текущую дату с точностью до дня

1.19 Ввод и вывод данных

В языке FIA-2024 не реализованы средства ввода данных.

Для вывода данных в стандартный поток вывода предусмотрен оператор writeline.

Пример: writeline 'T';

1.20 Точка входа

В языке FIA-2024 каждая программа должна содержать главную функцию main, т. е. точку входа, с которой начнется последовательное выполнение программы.

1.21 Препроцессор

Препроцессор в языке программирования FIA-2024 не предусмотрен.

1.22 Соглашения о вызовах

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

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

1.23 Объектный код

Исходный код языка транслируется в язык ассемблера.

1.24 Классификация сообщений транслятора

В случае возникновения ошибки в коде программы на языке FIA-2024 и выявления её транслятором в консоль и текущий файл протокола выводится сообщение. Классификация сообщений приведена в таблице 1.10.

Таблица 1.10. – Классификация сообщений транслятора

Интервал	Описание ошибок
0-59	Системные ошибки
60-89	Ошибки таблиц лексем и идентификаторов

90-99	Ошибки лексического анализа
100-129	Ошибки параметров и файлов протоколов
600-699	Ошибки синтаксического анализа
700-799	Ошибки семантического анализа

Компилятор может обрабатывать до 800 различных ошибок.

1.25 Контрольный пример

Код контрольного примера представлен в Приложении А.

2. Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор языка программирования FIA-2024 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.

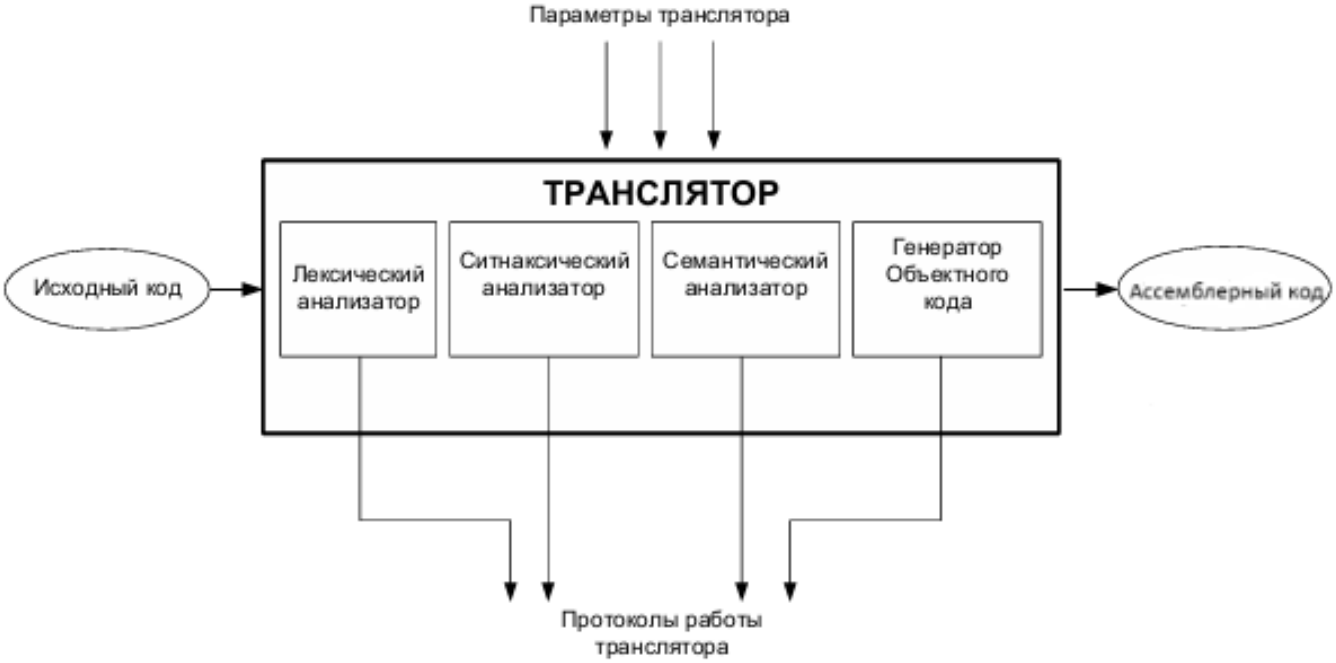

Рисунок 2.1 – Структура транслятора языка программирования FIA-2024

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 FIA-2024

Входной параметр	Описание параметра	Значение по умолчанию
-in:<путь к in-файлу>	Файл с исходным кодом на языке FIA-2024, имеющий расширение .txt	Не предусмотрено
-log:<путь к log-файлу>	Файл журнала для вывода протоколов работы программы.	Значение по умолчанию: <имя in-файла>.log
-out:<путь к out-файлу>	Выходной файл – результат работы транслятора. Содержит исходный код на языке ассемблера.	Значение по умолчанию: <имя in-файла>.out

Входные параметры указываются через командную строку вручную, в графическом интерфейсе – автоматически.

2.3 Протоколы, формируемые транслятором и их содержимое

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 FIA-2024

Формируемый протокол	Описание выходного протокола
Файл журнала, заданный параметром "-log:"	Файл с протоколом работы транслятора языка программирования FIA-2024. Содержит информацию про входные параметры, общем количестве символов и строк(исходные данные), протокол работы синтаксического анализатора, полученный на этапе синтаксического анализа.

Выходной файл, заданный параметром "-out:"	Результат работы программы – файл, содержащий исходный код на языке ассемблера.

Все файлы создаются в корневом каталоге.

3. Разработка лексического анализатора

3.1 Структура лексического анализатора

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся исходный код входного языка. На данном этапе распознаётся правильность составления лексем и идентификаторов языка. Для работы лексический анализатор использует исходный код на языке FIA-2024. В итоге будут сформированы таблица лексем и таблица идентификаторов.

Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

Исходный код программы представлен в приложении А, структура лексического анализатора представлена на рисунке 3.1.

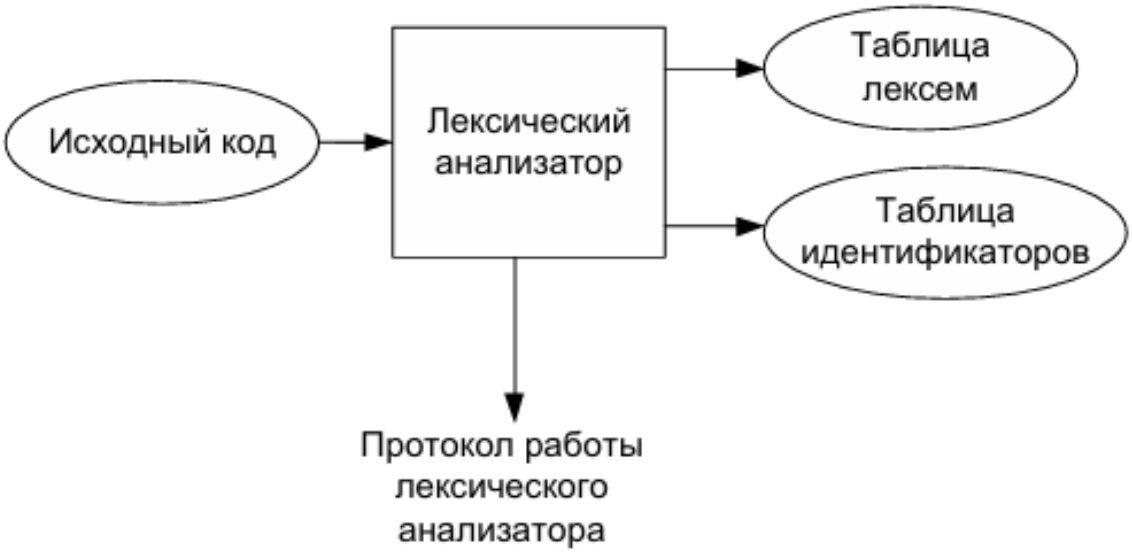

Рисунок 3.1 Структура лексического анализатора

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
Одинарные кавычки	Q
Пробел	S
Лексически значимые символы(символы-сепараторы)	L
Замена символа на указанный в одинарных кавычках	‘|’

На рисунке 3.2 приведена таблица контроля входных символов

```
//       0      1      2      3      4      5      6      7      8      9      A      B      C      D      E      F
 #define IN_CODE_TABLE {\
 /*0*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,  '|' , IN::T, IN::T, IN::I, IN::T, IN::T,\
 /*1*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\
 /*2*/   IN::S,  IN::T, IN::F, IN::T, IN::F, IN::T, IN::T, IN::Q, IN::L, IN::L, IN::L, IN::L, IN::L, IN::L, IN::T, IN::L,\
 /*3*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::L, IN::T, IN::L, IN::T, IN::T,\
 /*4*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::I, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\
 /*5*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::I, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\
 /*6*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\
 /*7*/   IN::T , IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::L, IN::T, IN::L, IN::T, IN::T,\
                                                                                                                      \
 /*8*/   IN::T,  IN::T, IN::T, IN::T, IN::I, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\
 /*9*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\
 /*A*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\
 /*B*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\
 /*C*/   IN::T,  IN::T, IN::I, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::I,\
 /*D*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\
 /*E*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\
 /*F*/   IN::T,  IN::T, IN::T, IN::T, IN::T, IN::T, IN::T ,IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T\
```

Рисунок 3.2 Таблица контроля входных символов

3.3 Удаление избыточных символов

В языке программирования FIA-2024 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
unt sym bool	t	Названия типов данных языка.
Идентификатор	i	Длина идентификатора – 50 символов.

Литерал	l	Литерал любого доступного типа.
function	f	Объявление функции.
return	r	Выход из функции. Возвращение значения из функций других типов.
Main	m	Точка входа в программу.
create	c	Объявление переменной
while	w	Оператор цикла
writeline	P	Поток вывода с переходом на новую строку
;	;	Разделение выражений.
,	,	Разделение параметров функций.
{	{	Начало тела функции или блока кода.
}	}	Закрытие тела функции или блока кода.
(	(	Передача параметров в функцию, приоритет операций.
)	)	Закрытие блока для передачи параметров, приоритет операций.
+ -	+ -	Знаки математических операций
> < !	> < !	Знаки логических операторов
=	=	Знак присваивания.
==	e	Сравнение
<< >>	<< >>	Сдвиги вправо и влево

Реализации графов переходов находятся в приложении Б.

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному

графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов.

Пример реализованного конечного автомата ключевого слова main языка FIA-2024 представлен на рисунке 3.3.

```
FST main_fst(
    word,
    5,
    NODE(1, RELATION('m', 1)),
    NODE(1, RELATION('a', 2)),
    NODE(1, RELATION('i', 3)),
    NODE(1, RELATION('n', 4)),
    NODE()
);

if (execute(main_fst)) {
    return LEX_MAIN;
}
```

Рисунок 3.3 Реализация конечного автомата для ключевого слова main

3.5 Основные структуры данных

Основные структуры данных языка AAM-2024 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 lexem, которое отвечает за значение лексемы; как поле idxTI, которое используется только идентификаторами или же лексемами для отображения индекса данной лексемы в таблице идентификаторов и в таблице лексем; как поле src_str_num, которое используется для отображения номера строки, в которой распознана данная лексема.

За таблицу идентификаторов отвечает структура данных IdTable. В ней есть те же поля что и в таблице лексем а также дополнительно поля: count_literals для подсчета количества литералов, dups и count_dups для хранения информации и повторном вхождении идентификатора, с целью дальнейших семантических проверок. Только вот элементом массива table является структура Entry, находящаяся внутри пространства имен IT. У нее есть поля: first_line_ID, id, lex_link, IDDataType, IDType, value, scope . Поле first_line_ID отображает номер строки исходного кода в которой находится идентификатор. Поле id -имя идентификатора. Поле IDDataTepe —-это перечисление, отвечающее за тип данных идентификатора, может принимать значения IT::INT, IT::CHR, IT::BOO.Поле lex_link- перекрестная ссылка таблицы

лексем с таблицей идентификаторов, показывает по каким номер в таблице индификаторов находится советующая лексема. Поле IDType также является перечислением, только оно уже отвечает за то, какого типа данный идентификатор — функция, литерал, параметр, переменная. Поле scope- область видимости идентификатора. Поле value — является объединением, в котором есть поля для хранения значений целочисленного, логического, символьного литералов.

Основные структуры данных приведены в приложении Б.

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
60	Таблица лексем: Невозможно добавить новый элемент. Превышен максимальный размер таблицы
61	Таблица лексем: Невозможно получить элемент из таблицы. Индекс меньше или больше возможного.
63	Таблица лексем: Невозможно открыть поток для вывода таблицы лексем
65	Таблица идентификаторов: Имя идентификатора длиннее разрешённого значения
66	Таблица идентификаторов: Невозможно добавить новый элемент. Превышен максимальный размер таблицы
67	Таблица идентификаторов: Невозможно получить элемент из таблицы. Индекс меньше или больше возможного
68	Таблица идентификаторов: Невозможно создать таблицу. Размер таблицы больше возможного
90	Лексический анализатор: Слово не распознано
91	Лексический анализатор: Длина литерала больше разрешённого значения
92	Лексический анализатор: Литерал не был распознан
93	Лексический анализатор: Идентификатор не был распознан

94	Лексический анализатор: Найдено больше одной или ни одной функции main
95	Лексический анализатор: Функция была определена больше чем один раз
96	Лексический анализатор: Переменная была объявлена больше одного раза
97	Лексический анализатор: Нераспознанная переменная
98	Лексический анализатор: Ошибки пунктуации

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

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся в файл-протокол.

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 3.4.

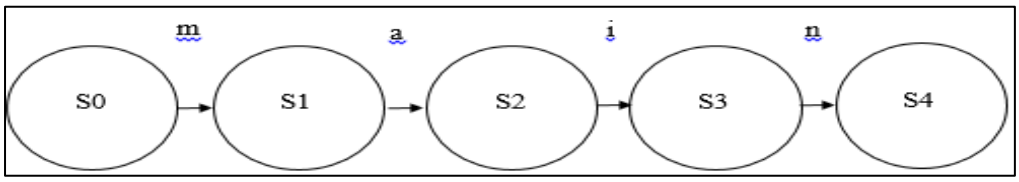

Рисунок 3.4 – Пример графа для цепочки main

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.

Структура синтаксического анализатора представлена на рисунке 4.1

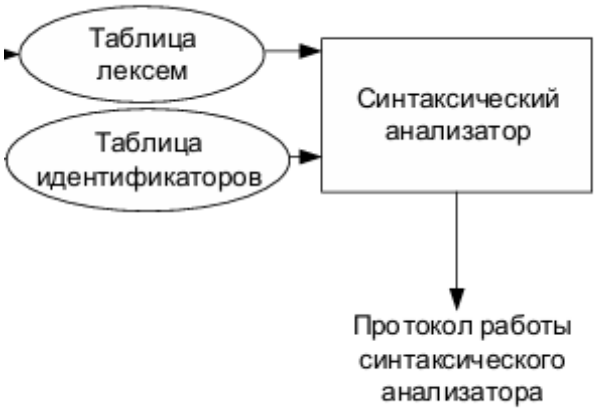

Рисунок 4.1 Структура синтаксического анализатора

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

Синтаксис языка FIA-2024 описывается грамматикой типа 2 по иерархии Хомского:

G = <T, N, P, S>

T – множество терминальных символов (алфавит языка FIA-2024),

N – множество нетерминальных символов,

P – множество правил языка,

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

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

Таблица 4.1. Правила нетерминальных символов

Символ	Правила	Какие правила порождает
S	S->m{NrE}S S->tfi(F){NrE}S S->m{NrE} S->tfi(F){NrE} S->tfiI{NrE}	Стартовые правила, описывающее общую структуру программы
N	N->cti;N	Правила набора операций

	N->cti; N->cti=E;N N->cti=E; N->i=E;N N->i=E; N->cti=S(W);N N->cti= S(W); N->cti=C(W);N N->cti= C(W); N->i=S(W);N N->i= S(W); N->i=C(W);N N->i= C(W); N->PE; N->PE:N N->T(EC)[N];N N->T(EC)[N]; N->w(EC)[N];N N->w(EC)[N]; N->rE; N->iM; N->iM;N	
E	E->i E->l E->(E) E->i(W) E->i() E->iM E->lM E->i(W)M	Правила выражений
W	W->i W->l W->i,W W->l,W	Правила для вызываемых параметров
F	F->ti F->ti,	Правила для передаваемых параметров
M	M->+E M->+EM M->-E	Математические операции

	M->-EM M->*E M->*EM M->/E M->/EM M->%E M->%EM	
C	C->>E C-><E C->eE C->!E	Правила записи логических выражений
I	I->()	Правила для функции без парметров

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. Так структура Rule представляет собой правило в грамматике Грейбаха и содержит поля nn- для нетерминального символа, iderror - идентификатор диагностического сообщения и size обозначающее количество цепочек. Также структура содержит вложенную структуру Chain, которая нужна для представления цепочки(правой части правила). Струкутра Chain содержит поля size для хранения длины цепочки и nt- указатель на массив символов цепочки, которые могут терминалами и не терминалами.

Структура Greibach представляет собой грамматику Грейбаха и содержит поля: size, startN, stbottomT, rules. Поле size нужно для хранения количества правил, startN хранит стартовый символ грамматики, stbottomT хранит символ обозначающий дно стека, и rules- указатель на массив правил.

Данные структуры представлены в приложении В.

4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора можно описать следующим образом:

1. В магазин записывается стартовый символ.
2. На основе полученной таблицы лексем формируется входная лента.
3. Запускается автомат и выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке.
4. 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.
5. Если в магазине встретился нетерминал, переходим к пункту 3.
6. 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 Параметры синтаксического анализатора

Входными параметрами для синтаксического анализатора в языке программирования FIA-2024 являются таблица лексем и таблица идентификаторов.

4.7 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
600	Синтаксический анализатор: Неверная структура программы
601	Синтаксический анализатор: Неверный идентификатор, использования ключевого слова или ошибка пунктуации
602	Синтаксический анализатор: Неверное выражение
603	Синтаксический анализатор: Неверное выражение в параметрах функции
605	Синтаксический анализатор: Неверный математический оператор или его использование

606	Синтаксический анализатор: Неверное условие выхода из цикла
607	Синтаксический анализатор: Неверно заданы параметры функции, не принимающей параметров

4.7 Параметры синтаксического анализатора и режимы его работы

Входными параметрами для синтаксического анализатора в языке программирования FIA-2024 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 FIA-2024 синтаксическим анализатором представлен в приложении Г.

5 Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализ выполняется после синтаксического. 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 ассемблера.

5.2 Функции семантического анализатора

Семантический анализатор выполняет проверку на основе правил языка, описанных в разделе 1.16, а также обрабатывает конструкции, ошибочные с точки зрения логики языка, и прочие непредвиденные ситуации.

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
700	Семантический анализатор: Ключевое слово используется в качестве идентификатора
701	Семантический анализатор: Две функции названы одинаково
702	Семантический анализатор: Две переменные названы одинаково
703	Семантический анализатор: Функция используется до инициализации
704	Семантический анализатор: Переменная используется до инициализации
705	Семантический анализатор: Некорректный тип данных для математической операции
706	Семантический анализатор: Некорректные параметры функции
707	Семантический анализатор: Функция принимает больше параметров, чем возможно
708	Семантический анализатор: Некорректный тип операции

709	Ошибка. Повторное объявление переменной!
710	Семантический анализатор: Тип данных возвращаемого значения не соответствует типу данных функции
711	Семантический анализатор: В функцию передано больше параметров, чем функция может принять
712	Семантический анализатор: Неверный тип данных операции больше/меньше
713	Семантический анализатор: Неверный тип данных при операции сравнения
714	Семантический анализатор: Неверный тип данных в условии выхода из цикла
715	Семантический анализатор: Повторяющийся возврат значения функции

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 FIA-2024 представлены 3 типа выражений: арифметические, сравнительные, и комбинаторные. Арифметические выражения содержат вычисления целочисленных типов данных, а также допускаются вызов функций (возвращающих тип) внутри выражений. Приоритет операций представлен на таблице 6.1.

Приоритет операций представлен на таблице 6.1.

Таблица 6.1. – Приоритеты операций

Операция	Значение приоритета
( )	2
+	1
-	1

Операторы в скобках имеют наивысший приоритет, за ними следуют умножение и деление, а затем сложение и вычитание.

В сравнительных выражениях можно использовать только целый беззнаковый тип данных. Операции которые используются в сравнительных выражениях: >,<,==,! Также можно использовать скобки для изменения приоритета операций.

В сдвиговых выражениях только можно использовать целый беззнаковый тип данных. Операции которые используются в выражениях: <<,>> Также можно использовать скобки для изменения приоритета операций.

Комбинаторные выражения представляют комбинацию сравнительных, арифметических и сдвиговых выражений.

6.2 Польская запись и принцип ее построения

Все выражения языка FIA-2024 преобразовываются к обратной польской записи.

Алгоритм построения польской записи:

1) исходная строка: выражение;
2) результирующая строка: польская запись;
3) стек: пустой;
4) исходная строка просматривается слева направо;
5) операнды переносятся в результирующую строку;
6) операция записывается в стек, если стек пуст;
7) операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
8) отрывающая скобка помещается в стек;

9) закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.

6.3 Программная реализация обработки выражений

Программная реализация обработки выражений представлена в приложении Е.

6.4 Контрольный пример

Пример преобразования выражений из контрольных примеров к обратной польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления и преобразования к ассемблерному коду. В приложении Г приведены изменённые таблицы лексем и идентификаторов, отображающие результаты преобразования выражений в польский формат.

Таблица 6.2. – Преобразование выражений к ПОЛИЗ

Выражение	Обратная польская запись для выражения
i[2]=(((l[3]+l[4])-i[0])*l[5])/l[6];	i[2]=l[3]l[4]+i[0]-l[5]*l[6]/
i[23]=(i[23]+l[26])*l[26]	i[23]=i[23]l[26]+l[26]*
i[3]=(((l[4]+l[5])-i[0])*l[6])	i[3]=l[4]l[5]+i[0]-l[6]*

Преобразование выражений в обратную польскую запись в языке FIA-2024 упрощает алгоритмы их вычисления и преобразования к ассемблерному коду.

7 Генерация кода

7.1 Структура генератора кода

В языке FIA-2024 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, дерево разбора полученные в результате лексического анализа и синтаксического анализа. В соответствии с таблицей лексем строится выходной файл на языке ассемблера, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода FIA-2024 представлена на рисунке 7.1.

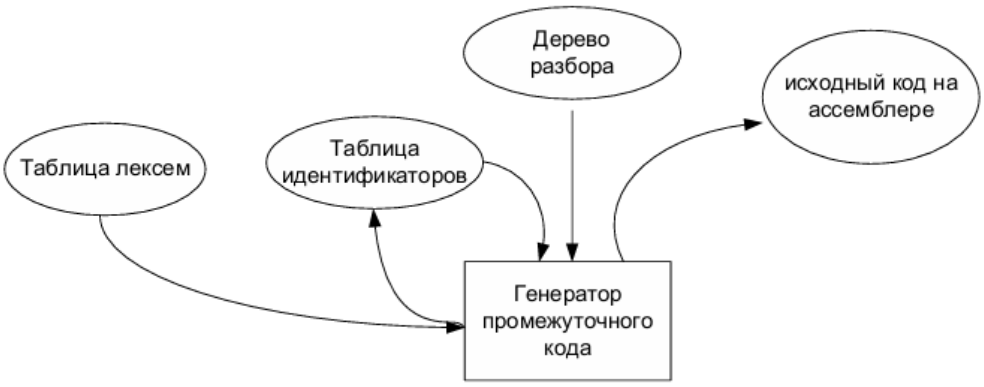

Рисунок 7.1 Структура генератора кода

7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке FIA-2024 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка и ассемблера

Тип идентификатора на языке FIA-2024	Тип идентификатора на языке ассемблера	Пояснение
unt	sdword	Хранит целочисленный тип данных.
sym	byte	Один символ хранится в поле размером 1 байт.
bool	byte	Хранится в поле размером 1 байт. Может принимать значение 1 дя истины и 0 для лжи.

Следовательно, таблица 7.1 показывает соответствия между типами идентификаторов на языке FIA-2024 и языке ассемблера. Это важно при переводе кода с

языка FIA-2024 на язык ассемблера, чтобы правильно определить типы данных и использовать соответствующие инструкции и регистры для работы с идентификаторами.

7.3 Статическая библиотека

В языке FIA-2024 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	Возвращаемый тип	Пояснение
get_date()	none	none	Выводит в консоль время на момент вызова с точностью до секунд и текущую дату
get_time()	none	none	Выводит в консоль текущую дату с точностью до дня

Данные функции входят в стандартную библиотеку FIA-2024

7.4 Особенности алгоритма генерации кода

Алгоритм генерации объектного кода выглядит следующим образом:

1. С помощью цикла for идем по таблице лексем.
2. С помощью тернарного оператора if разыскиваем лексемы, соответствие которым есть на языке ассемблера.
3. Если лексема равна i, нужно понять — это объявление новой переменной, часть лямбда-выражения в цикле while, или просто использование идентификатора.
4. Если лексема равна f, нужно понять какой тип данных имеет эта функция, сколько параметров принимает и типы данных этих параметров. Также нужно открыть новый процесс для этой функции, вписать в него необходимые данные и операции над ними и закрыть этот процесс.
5. Если обнаружен вызов функции в главной функции, то проверить какая функция вызывается, сколько параметров принимает и переместить эти параметры в регистры перед вызовом самой функции.
6. Если лексема равна m необходимо открыть главный процесс, после этого обработать все данные входящие в главную функцию в соответствии с пунктами, изложенными выше.
7. 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 кода на языке ассемблера приведен в приложении Ж.

8 Тестирование транслятора

8.1 Общие положения

В основе тестов лежит проверка работоспособности всех анализаторов. При обнаружении компилятором ошибки она будет обрабатываться одним из анализаторов в зависимости от типа ошибки. Все сообщения об ошибках будут храниться в файле text.txt.log.

8.2 Результаты тестирования

В языке FIA-2024 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
{
…
}
Main
{
…
}
	Ошибка: 94: Лексический анализатор: Найдено больше одной или ни одной функции main

На этапе лексического анализа в языке FIA-2024 могут возникнуть ошибки, описанные в пункте 3.7. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 - Тестирование лексического анализатора

Исходный код	Диагностическое сообщение
main{} main{}	Ошибка: 94: Лексический анализатор: Найдено больше одной или ни одной функции main

На этапе синтаксического анализа в языке FIA-2024 могут возникнуть ошибки, описанные в пункте 4.6. Примеры результатов тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 – Тестирование синтаксического анализатора

Исходный код	Диагностическое сообщение

main{
create int x;
}
	Ошибка: 601: Синтаксический анализатор: Неверный идентификатор, использования ключевого слова или ошибка пунктуации
	601: строка 2,

int func fi(int x){}
main{
fi();
}
	Ошибка: 600: Синтаксический анализатор: Неверная структура программы

int func wr(){
write 'r';
}
main{
wr('6');
return 0;
}
	Ошибка: 604: Синтаксический анализатор: Неверные параметры функции
	604: строка 0,Синтаксический анализатор: Неверные параметры функции

Семантический анализ в языке FIA-2024 содержит множество проверок по семантическим правилам, описанным в пункте 1.16. Примеры тестирования семантического анализатора на корректное обнаружение семантических ошибок приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

Исходный код	Диагностическое сообщение

main{
create int x;
create char h='h';
x=h;
return 0;
}
	Ошибка: 708: Семантический анализатор: Некорректный тип операции в строке 3

main
{
create int x;
create int x;
}
	Ошибка: 702: Семантический анализатор: Две переменные названы одинаково в строке 3

main{
fi();
return 0;
}
	Ошибка: 703: Семантический анализатор: Функция используется до инициализации в строке 1

main
{
x=1;
}
	Ошибка: 704: Семантический анализатор: Переменная используется до инициализации в строке 2 : x

main{
create char chr='o';
return chr;
}
	Ошибка: 710: Семантический анализатор: Тип данных возвращаемого значения не соответствует типу данных функции в строке 30

main{
return 8;
return 0;
}
	Ошибка: 715: Семантический анализатор: Повторяющийся возврат значения функции в строке 1 :4

Заключение

В ходе выполнения курсовой работы был разработан компилятор и генератор кода для языка программирования FIA-2024. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка FIA-2024;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. Осуществлена программная реализация синтаксического анализатора, позволяющая расширять набор синтаксических конструкций языка только за счёт внесения изменений в разработанную грамматику;
6. Разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
7. Разработан транслятор кода на язык ассемблера;
8. Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка FIA-2024 включает:

1. 3 типа данных;
2. Поддержка оператора вывода;
3. Возможность вызова функций стандартной библиотеки;
4. Наличие 5 арифметических операторов для вычисления выражений;
5. Наличие 4 операндов для сравнения.
6. Поддержка функций, операторов цикла и условия;
7. Структурированная и классифицированная система для обработки ошибок пользователя.

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

Список использованных источников

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

2. Вирт, Н. Построение компиляторов / Пер. с англ. Борисов Е. И., Чернышов Л.Н. – М.: ДМК Пресс, 2010. – 192с.: ил.

3. Ирвин, К. Язык ассемблера для процессоров Intel, 4-е издание.: Пер. с англ. – М.: Вильямс, 2005. – 912с.

4. Курс лекций по КПО / Наркевич А.С.

Приложение А

Листинг 1 – исходный код программы на языке FIA-2024

```
int func add(int a, int b)
{
create int sum;
sum = a + b;
return sum;
}

main
{

create int num1;
create int num2;
num1 = 10;
num2 = 25;

create int num3 = 10;
writeline num3;
num3 >> 2;
create int num4 = 0001;
writeline num4;

create int sum_12 = add(num1,num2);
writeline sum_12;

get_time();
get_date();

create int iter = 0;
while(iter !  10)[ writeline iter; iter = iter + 1; ];
create bool More = false;


return 0;

}
```

Приложение Б

Листинг 1- Таблица лексем

```
00 t [0] f [1] i [2] ( [3] t [4] i [5] , [6] t [7] i [8] ) [9]

01 { [10]

02 c [11] t [12] i [13] ; [14]

03 i [15] = [16] i [17] + [18] i [19] ; [20]

04 r [21] i [22] ; [23]

05 } [24]

07 m [25]

08 { [26]

10 c [27] t [28] i [29] ; [30]

11 c [31] t [32] i [33] ; [34]

12 i [35] = [36] l [37] ; [38]

13 i [39] = [40] l [41] ; [42]

14 i [43] = [44] i [45] ; [46]

15 c [47] t [48] i [49] = [50] i [51] ( [52] i [53] , [54] i [55] ) [56] ; [57]

17 c [58] t [59] i [60] = [61] l [62] ; [63]

19 w [64] ( [65] i [66] ! [67] l [68] ) [69] [ [70] i [71] = [72] i [73] + [74] l
[75] ; [76] i [77] = [78] i [79] + [80] l [81] ; [82] ] [83] ; [84]

20 c [85] t [86] i [87] = [88] l [89] ; [90]

22 T [91] ( [92] i [93] e [94] l [95] ) [96] [ [97] r [98] l [99] ; [100] ] [101]
; [102]

23 T [103] ( [104] i [105] e [106] l [107] ) [108] [ [109] r [110] l [111] ;
[112] ] [113] ; [114]

24 c [115] t [116] i [117] = [118] S [119] ( [121] l [122] , [123] l [124] ,
[125] l [126] ) [127] ; [128]

25 c [129] t [130] i [131] = [132] C [133] ( [135] l [136] , [137] l [138] )
[139] ; [140]

26 P [141] i [142] ; [143]

27 p [144] i [145] ; [146]

28 r [147] l [148] ; [149]

30 } [150]
```

Листинг 2- Таблица идентификаторов

№	Identifier	Data type	Identifier type	Scope
00/2	add	1	2	NULL
00/5	a	1	3	add
00/8	b	1	3	add
02/13	sum	1	1	add
10/29	num1	1	1	Main
11/33	num2	1	1	Main
12/37	Literal_0	1	4 (00FFh)	Main
13/41	Literal_1	1	4 (25)	Main
15/49	sum_12	1	1	Main
17/60	cond	1	1	Main
17/62	Literal_2	1	4 (0)	Main
19/68	Literal_3	1	4 (10)	Main
19/75	Literal_4	1	4 (1)	Main
19/81	Literal_5	1	4 (1)	Main
20/87	More	3	1	Main
20/89	Literal_6	3	4(false)	Main
22/95	Literal_7	3	4(true)	Main
22/99	Literal_8	1	4 (1)	Main

23/107)	Literal_9 Main		3	4(false
23/111 Main	Literal_10		1	4 (0)
24/117 Main	answer	1	1	
24/119 Main	L_strToint		1	2
24/122 Main	Literal_11		2	4 (1)
24/124 Main	Literal_12		2	4 (2)
24/126 Main	Literal_13		2	4 (3)
25/131 Main	answ1	1	1	
25/133 Main	L_cmpChr		1	2
25/136 Main	Literal_14		2	4 (A)
25/138 Main	Literal_15		2	4 (a)
28/148 Main	Literal_16		1	4 (0)

Приложение В

Листинг 1. Грамматика языка FIA-2024

```
        Greibach greiibach(NS('S'), TS('$'),
      8,
      Rule(NS('S'), GRB_ERROR_SERIES + 0,
          5,
          Rule::Chain(7, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'),
TS('}')),
          Rule::Chain(13, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'),
TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), NS('S')),
          Rule::Chain(13, TS('t'), TS('f'), TS('i'), TS('('), NS('I'), TS(')'),
TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), NS('S')),
          Rule::Chain(8, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'),
TS('}'), NS('S')),
          Rule::Chain(12, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'),
TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'))

      ),
      Rule(NS('N'), GRB_ERROR_SERIES + 1,
          18,

          Rule::Chain(4, TS('c'), TS('t'), TS('i'), TS(';')),
          Rule::Chain(5, TS('c'), TS('t'), TS('i'), TS(';'), NS('N')),
          Rule::Chain(6, TS('c'), TS('t'), TS('i'), TS('='), NS('E'), TS(';')),
          Rule::Chain(7, TS('c'), TS('t'), TS('i'), TS('='), NS('E'), TS(';'),
NS('N')),
          Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),
          Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),
          Rule::Chain(8, TS('c'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'),
TS(')'), TS(';')),
          Rule::Chain(9, TS('c'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'),
TS(')'), TS(';'), NS('N')),
          Rule::Chain(3, TS('r'), NS('E'), TS(';')),
          Rule::Chain(4, TS('r'), NS('E'), TS(';'), NS('N')),
          Rule::Chain(3, TS('p'), NS('E'), TS(';')),
          Rule::Chain(4, TS('p'), NS('E'), TS(';'), NS('N')),
          Rule::Chain(3, TS('P'), NS('E'), TS(';')),
          Rule::Chain(4, TS('P'), NS('E'), TS(';'), NS('N')),
          Rule::Chain(10, TS('T'), TS('('),NS('E'), NS('C'), TS(')'), TS('['),
NS('N'), TS(']'), TS(';'), NS('N')),
          Rule::Chain(9, TS('T'), TS('('),NS('E'), NS('C'), TS(')'), TS('['),
NS('N'), TS(']'), TS(';')),
          Rule::Chain(9, TS('w'), TS('('),NS('E'), NS('C'), TS(')'), TS('['),
NS('N'), TS(']'), TS(';')),
          Rule::Chain(10, TS('w'), TS('('),NS('E'), NS('C'), TS(')'), TS('['),
NS('N'), TS(']'), TS(';'), NS('N'))),
      Rule(NS('E'), GRB_ERROR_SERIES + 2,
          9,
          Rule::Chain(1, TS('i')),
          Rule::Chain(1, TS('l')),
          Rule::Chain(3, TS('('), NS('E'), TS(')')),
          Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),
          Rule::Chain(3, TS('i'), TS('('), TS(')')),
```

```
          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(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M'))
      ),
      Rule(NS('W'), GRB_ERROR_SERIES + 3,
          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'))
      ),
      Rule(NS('F'), GRB_ERROR_SERIES + 4,
          2,
          Rule::Chain(2, TS('t'), TS('i')), Rule::Chain(4, TS('t'), TS('i'),
TS(','), NS('F'))
      ),
      Rule(NS('M'), GRB_ERROR_SERIES + 5,
          10,
          Rule::Chain(2, TS('+'), NS('E')),
          Rule::Chain(3, TS('+'), NS('E'), NS('M')),
          Rule::Chain(2, TS('-'), NS('E')),

          Rule::Chain(3, TS('-'), NS('E'), NS('M')),
          Rule::Chain(2, TS('*'), NS('E')),
          Rule::Chain(3, TS('*'), NS('E'), NS('M')),
          Rule::Chain(2, TS('/'), NS('E')),

          Rule::Chain(3, TS('/'), NS('E'), NS('M')),
          Rule::Chain(2, TS('%'), NS('E')),
          Rule::Chain(3, TS('%'), NS('E'), NS('M'))
      ),
      Rule(NS('C'), GRB_ERROR_SERIES + 6,
          4,
          Rule::Chain(2, TS('>'), NS('E')),

          Rule::Chain(2, TS('<'), NS('E')),

          Rule::Chain(2, TS('e'), NS('E')),

          Rule::Chain(2, TS('!'), NS('E'))
      ),
      Rule(NS('I'), GRB_ERROR_SERIES + 8,
          1,
          Rule::Chain(2, TS('('), TS(')'))
      )
  );
```

Приложение Г

Листинг 1- дерево разбора языка FIA2024

```
S->tfi(F){NrE;}S
4   : F->ti,F
7   : F->ti
11  : N->cti;N
15  : N->i=E;
17  : E->iM
18  : M->+E
19  : E->i
22  : E->i
25  : S->m{NrE;}
27  : N->cti;N
31  : N->cti;N
35  : N->i=E;N
37  : E->l
39  : N->i=E;N
41  : E->l
43  : N->i=E;N
45  : E->i
47  : N->cti=E;N
51  : E->i(W)
53  : W->i,W
55  : W->i
58  : N->cti=E;N
62  : E->l
64  : N->w(EC)[N];N
66  : E->i
67  : C->!E
68  : E->l
71  : N->i=E;N
```

```
73  : E->iM
74  : M->+E
75  : E->l
77  : N->i=E;
79  : E->iM
80  : M->+E
81  : E->l
85  : N->cti=E;N
89  : E->l
91  : N->T(EC)[N];N
93  : E->i
94  : C->eE
95  : E->l
98  : N->rE;
99  : E->l
103 : N->T(EC)[N];N
105 : E->i
106 : C->eE
107 : E->l
110 : N->rE;
111 : E->l
115 : N->cti=S(W);N
121 : W->l,W
123 : W->l,W
125 : W->l
128 : N->cti=C(W);N
134 : W->l,W
136 : W->l
139 : N->PE;N
140 : E->i
142 : N->pE;
```

```
143 : E->i

146 : E->l
```