МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий
Кафедра Программной инженерии
Специальность 6-05-0612-01 Программная инженерия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

«Разработка компилятора GED-2024»

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

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

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

Курсовой проект защищен с оценкой _____

Минск 2024

Содержание

Введение

Целью курсового проекта является разработка собственного языка программирования и компилятора для него. Язык называется GED-2024. Написание компилятора будет осуществляться на языке C++.

Компилятор – это программа, задачей которого является перевод программы, написанной на одном из языков программирования в программу на язык ассемблера.

Компиляция состоит из двух частей: анализа и генерации. Анализ – это разбиение исходной программы на составные части и создание ее промежуточного представления. Генерация – конструирование требуемой целевой программы из промежуточного представления. В данном курсовом проекте исходный код транслируется на язык ассемблера.

Для выполнения курсового проекта были поставлены следующие задачи:

– разработка спецификации языка программирования;
– разработка структуры транслятора;
– разработка лексического анализатора;
– разработка синтаксического анализатора;
– разработка семантического анализатора;
– преобразование выражений;
– генерация кода на язык ассемблер;
– тестирование транслятора.

Глава 1 Спецификация языка программирования

1.1 Характеристика языка программирования

Язык программирования GED-2024 классифицируется как процедурный, универсальный, строго типизированный, компилируемый.

1.2 Определение алфавита языка программирования

В основе алфавита GED-2024 лежит таблица символов Windows-1251, которая представлена на рисунке 1.1.

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

Рисунок 1.1 – Таблица кодировки Windows-1251

В языке GED-2024 могут использоваться символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, спецсимволы, а также непечатные символы пробела, табуляции и перевода строки. Русские символы разрешены только в строковых литералах.

1.3 Применяемые сепараторы

Сепараторы необходимы для разделения операций языка. Сепараторы, используемые в языке программирования GED-2024, приведены в таблице 1.1.

Таблица 1.1 - Применяемые 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 в условных конструкциях
(…)	Круглые скобки	Приоритет операций, параметры функции
‘…’	Одинарные кавычки	Допускается везде, кроме идентификаторов и ключевых слов
=	Знак «равно»	Присваивание значения
,	Запятая	Разделение параметров
+ - * | & ~	Знаки «плюс», «минус», «астерикс», «прямая черта», «амперсанд», «тильда»	Выражения

1.4 Применяемые кодировки

При трансляции исходного кода применяется кодировка Windows-1251. Описание кодировки представлено в пункте 1.2.

1.5 Типы данных

Допускается использование фундаментальных типов данных. В языке GED-2024 реализованы 2 типа данных: целочисленный и символьный. Описание типов данных, предусмотренных в данном языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка GED-2024

Тип данных	Описание
Беззнаковый целый тип данных (int)	Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных). Инициализация по умолчанию: значение 0. Максимально допустимое значение 2^{31}-1. Минимально допустимым является 0.
Символьный тип данных (symb)	Фундаментальный тип данных. Используется для работы с символами, который в памяти занимает 1 байт. Инициализация по умолчанию: символ конца строки «\0».

Пользовательские типы данных не поддерживаются.

1.6 Преобразование типов данных

В языке программирования GED-2024 преобразование типов данных не поддерживается. Все типы данных определенны однозначно и не могут быть преобразованы в другие.

1.7 Идентификаторы

Для именования функций, параметров и переменных используются идентификаторы. Не предусмотрены зарезервированные идентификаторы. Имя идентификаторов не должно совпадать с ключевыми словами языка и с именами функций стандартной библиотеки. В имени идентификатора допускаются только символы латинского алфавита нижнего регистра [a … z]. Максимальная длина имени идентификатора – 15 символов. Максимальная длина имени идентификатора функции – 15 символов.

1.8 Литералы

В языке GED-2024 предусмотрены 2 вида литералов: целочисленные и строковые. Краткое описание литералов приведено в таблице 1.3.

Таблица 1.3 – Описание литералов

Литерал	Пояснение
Целочисленный	Целочисленные неотрицательные литералы, по умолчанию инициализируются 0. Максимально допустимое значение 2^{31}-1. Минимально допустимым является -2^{31}-1. При выходе за пределы допустимости выводится соответствующая ошибка.

Строковый	Используются символы кодировки ASCII. Максимальный размер строки – 255.

Строковый литерал используется только внутри оператора вывода write. Литералы являются константами и при генерации кода, заносятся в раздел .const.

1.9 Объявления данных

В языке GED-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. Каждая переменная получает префикс – название функции, в которой она объявлена. Недопустимо объявление глобальных переменных.

1.10 Инициализация данных

При объявлении переменной не допускается инициализация данных. Краткое описание способов инициализации переменных языка GED-2024 представлено в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

Конструкция	Описание	Пример
set <тип данных> <идентификатор>;	Автоматическая инициализация: переменные типа int инициализируются нулём, переменные типа symb – пустым символом.	set int sum; set symb abc;

<идентификатор> = <значение>;	Присваивание переменной значения. Целочисленные значения могут представляться как в десятичном, так и шестнадцатеричном виде.	sum = 15;
		sum = 0x1F;
		abc = ‘a’;

Соответствие типов проверяется с помощью семантического анализа.

1.11 Инструкции языка

Все возможные инструкции языка программирования GED-2024 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования GED-2024

Инструкция	Запись на языке GED-2024
Объявление переменной	set <тип данных> <идентификатор>;
Объявление функции	set <тип данных> func <идентификатор> (<тип данных> <идентификатор>, …) {<блок кода>};

Присваивание	<идентификатор> = <значение>/<идентификатор>;

Блок инструкций	main
	{
	…
	}

Возврат из подпрограммы	ret <идентификатор> / <литерал>;
Условная инструкция	if(<условие>)[<блок кода>];
Вывод данных	write <идентификатор> / <литерал>;

Условный оператор и функции входа в программу не требуют закрывающую «;».

1.12 Операции языка

Язык программирования GED-2024 может выполнять операции, представленные в таблице 1.6.

Таблица 1.6 – Операции языка программирования GED-2024

Операция	Примечание	Пример
=	Присваивание	sum = 15; symbol = ‘S’;
&	Побитовое И	a & b;
|	Побитовое ИЛИ	a | b;
~	Отрицание	a ~ 1
(	Приоритет операций	(a + b) * c;
)		
+	Суммирование	a + b;
-	Вычитание	a – b;
*	Умножение	a * b;

1.13 Выражения и их вычисления

Всякое выражение составляется согласно следующим правилам:

- Допускается использовать скобки для смены приоритета операций;
- Выражение записывается в строку без переносов;
- Использование двух подряд идущих операторов не допускается;
- Допускается использовать в выражении вызов функции, вычисляющей и возвращающей целочисленное значение.

Перед генерацией кода каждое выражение приводится к записи в польской записи для удобства дальнейшего вычисления выражения на языке ассемблера [1].

1.14 Конструкции языка

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

Таблица 1.7 – Программные конструкции языка GED-2024

Конструкция	Запись на языке GED-2024

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

Функция	<тип> func <идентификатор> (<тип> <идентификатор>, …)
	{
	…
	ret <идентификатор> / <литерал>;
	};

Конструкция цикла	while(<переменная>) […]

В данной таблице представлены различные конструкции программирования и их реализация. Главная функция "main" является основной точкой входа в программу. Внешние функции определяются с помощью ключевого слова "func" со списком параметров. Конструкция цикла позволяет выполнить действие столько раз, сколько было указано в переменной.

1.15 Область видимости идентификаторов

В языке GED-2024 все переменные являются локальными, т.е. имеют функциональную область видимости. Они обязаны находится внутри программного блока функций. Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

1.16 Семантические проверки

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

Таблица 1.8 – Семантические проверки

Номер	Правило
1	Идентификаторы не должны повторно объявляться в пределах одной функции.
2	Тип возвращаемого значения должен совпадать с типом функции при её объявлении или подключении
3	Тип данных передаваемых значений в функцию должен совпадать

	с типом параметров при её объявлении или подключении
4	В функцию должно быть передано то число параметров, сколько ожидается
5	Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается
6	Проверка оператора цикла

Если семантическая проверка не проходит, то в лог журнал записывается соответствующая ошибка.

1.17 Распределение оперативной памяти на этапе выполнения

Все переменные размещаются в куче.

1.18 Стандартная библиотека и её состав

Для использования функций стандартной библиотеки GED-2024, нужно явно подключить необходимую функцию с помощью ключевого слова extern, далее 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	Назначение
int stdcall step(int int1, int int2)	Возведение int1 в степень int2
int stdcall iabs(int aint)	Вычисляет абсолютное значение
void stdcall _ConsoleWriteInt(int number)	Выводит значение на консоль
void stdcall output(const char* buffer)	Принимает аргумент типа const char*(buffer) представляющий строку и выводит ее значение на консоль

void_stdcall _ConsoleWriteInt(long number) необходима для вывода целочисленных литералов.

1.19 Ввод и вывод данных

В языке GED-2024 не реализованы средства ввода данных. Для вывода данных в стандартный поток вывода предусмотрен оператор write, который базируется на приватных функциях стандартной библиотеки.

1.20 Точка входа

В языке GED-2024 каждая программа должна содержать главную функцию, точку входа, с которой начнется последовательное выполнение программы. В программе может быть только одна точка входа.

1.21 Препроцессор

Препроцессор в языке программирования GED-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

GED-2024 транслируется в язык ассемблера.

1.24 Классификация сообщений транслятора

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

Таблица 1.10 – Классификация сообщений транслятора

Интервал	Описание ошибок
0-99	Системные ошибки
100-109	Ошибки параметров
110-119	Ошибки открытия и чтения файлов
120-129	Ошибки лексического анализа
130-139	Ошибки таблиц лексем и таблиц идентификаторов
600-699	Ошибки синтаксического анализа
700-900	Ошибки семантического анализа

Компилятор может обрабатывать до 1000 различных ошибок.

1.25 Контрольный пример

Контрольный пример представлен в Приложении А.

Глава 2 Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор преобразует программу, написанную на языке GED-2024 в программу на языке ассемблера. Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.

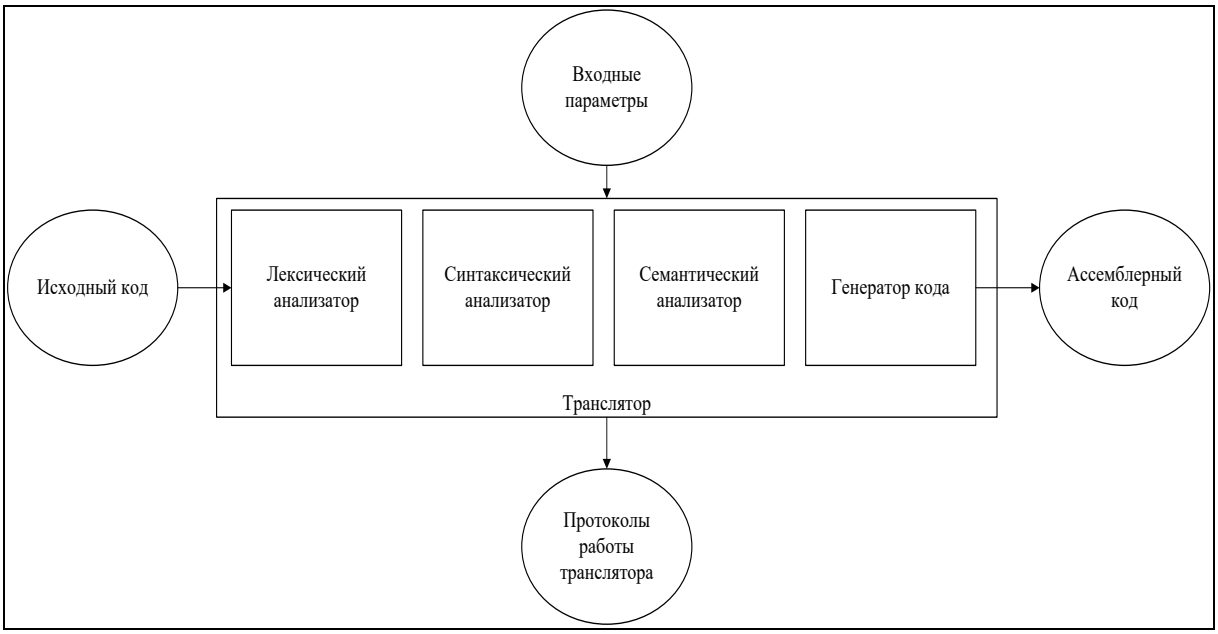

Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Цели лексического анализатора:

− убрать все лишние пробелы;

− выполнить распознавание лексем;

− построить таблицу лексем и таблицу идентификаторов;

− при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

Семантический анализ в свою очередь является проверкой исходной программы GED-2024 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода GED-2024. Для этого используются таблица лексем и идентификаторов. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке GED-2024, прошедший все предыдущие этапы, в код на языке ассемблера [2].

2.2 Перечень входных параметров транслятора

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка GED-2024

Входной параметр	Описание	Значение по умолчанию
-in:<имя_файла>	Входной файл с любым расширением, в котором содержится исходный код на языке GED-2024. Данный параметр должен быть указан обязательно. В случае если он не будет задан, то выполнение этапа трансляции не начнётся.	Не предусмотрено
-log:<имя_файла>	Файл содержит в себе краткую информацию об исходном коде на языке GED-2024.	<имя_файла>.log
-out:<имя_файла>	В этот файл будет записан результат трансляции кода на язык assembler.	<имя_файла>.asm

В таблице 2.1 представлены входные параметры транслятора языка GED-2024, которые используются для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов**.**

2.3 Протоколы, формируемые транслятором и их содержимое

Таблица с перечнем протоколов, формируемых транслятором языка GED-2024 и их назначением представлена в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка GED-2024

Формируемый протокол	Описание протокола
Файл журнала, “*.log ”	Файл содержит в себе краткую информацию об исходном коде на языке GED-2024. В этот файл выводится протокол работы анализаторов, а также различные ошибки
“Out.asm”	Содержит сгенерированный код на языке Ассемблера.

Глава 3 Разработка лексического анализатора

3.1 Структура лексического анализатора

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке GED-2024. Структура лексического анализатора представлена на рисунке 3.1

Рисунок 3.1 – Структура лексического анализатора GED-2024

Результатом работы лексического анализатора являются заполненные таблица лексем и таблица идентификаторов.

3.2 Входные и выходные данные лексического анализатор

Входными параметрами являются данные исходного кода. Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. В качестве выходных параметров выступают таблица лексем и таблица идентификаторов, которые необходимы для продолжения компиляции, в качестве входных данных синтаксического анализатора.

3.3 Параметры лексического анализатора

Входные параметры используются для вывода результата работы лексического анализатора. Они передаются аргументами и представлены в таблице 2.1.

3.4 Алгоритм лексического анализа

Алгоритм работы лексического анализа заключается в последовательном распознавании и разборе цепочек исходного кода и заполнение таблиц идентификаторов и лексем.

Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. В случае, если подходящий автомат не был обнаружен, запоминается номер строки, в которой находился этот токен и выводится сообщение об ошибке. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является.

Регулярные выражения – аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа [3].

Регулярное выражение для ключевого слова set: «set». Граф конечного автомата для этой лексемы представлен на рисунке 3.1. S0 – начальное состояние, S3 – конечное состояние автомата.

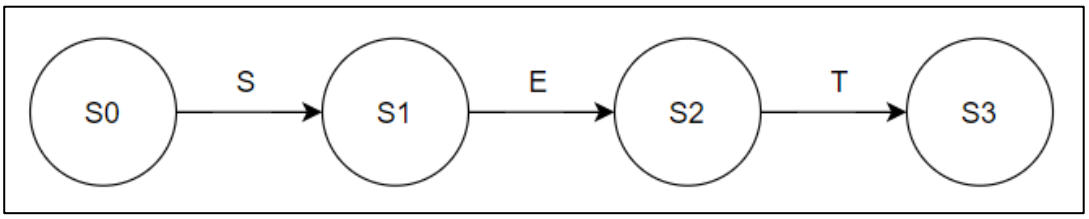

Рисунок 3.1 – Граф переходов для цепочки «set»

3.5 Контроль входных символов

Таблица для контроля входных символов представлена в Приложении Ж.

Принцип работы таблицы заключается в соответствии значения каждому элементу в шестнадцатеричной системе счисления значению в таблице Windows-1251.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, S – пробельный символ, Q – символ одинарной кавычки, I – игнорированный символ, O – операция.

3.6 Удаление избыточных символов

Удаление избыточных символов не предусмотрено, так как после проверки на допустимость символов исходный код на языке программирования GED-2024 разбивается на токены, которые записываются в очередь.

3.7 Перечень ключевых слов

Лексемы – это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы [4]. Данное соответствие описано в таблице 3.1.

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

Тип цепочки	Примечание	Цепочка	Лексема
Тип данных	Целочисленный тип данных	int	t
	Символьный тип данных	symb	t
Лексем	Объявление переменной	set	d
	Оператор вывода	write	p
	Объявление функции	func	f
	Возврат значения из функции	ret	r
	Оператор цикла	while	e
	Блок инструкции цикла	[	[
		]	]
	Блок функции	{	{
		}	}
	Изменение приоритетности в выражении и отделение параметров функций	(	(
		)	)
	Сепараторы	;	;
		,	,
	Оператор присваивания	=	=
Оператор	Знаки арифметических операций	+	+
		-	-
		*	*
		~	~
		&	&
		|	|
Идентификатор	-	[a-z;A-Z]+ [a-z;A-Z;0-9]*	i
Литерал	Целочисленный литерал	[1-9]+[0-9]*	l
	Строковый литерал	[a-z;A-Z]+ [a-z;A-Z;0-9]*	l
Точка входа	-	main	m

В Приложении А находятся конечные автоматы, соответствующие лексемам языка GED-2024.

3.8 Основные структуры данных

Основные структуры таблиц лексем и идентификаторов данных языка GED-2024, используемых для хранения, представлены в Приложении А. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде, номер столбца в исходном коде, индекс таблицы идентификаторов (если нет соответствующего идентификатора, то индекс равен -1), а также специальное поле, в котором хранится значение лексемы. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора, его значение, а также бинарное поле для определения внешний ли идентификатор.

3.9 Структура и перечень сообщений лексического анализатора

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 121-131. Перечень сообщений лексического анализатора представлен в листинге 3.1.

```
ERROR_ENTRY(121, "[LA]: Используется необъявленный идентификатор"),
ERROR_ENTRY(122, "[LA]: Переполнение таблицы идентификаторов"),
ERROR_ENTRY(123, "[LA]: Переполнение таблицы лексем"),
ERROR_ENTRY(124, "[LA]: Отсутствует точка входа или их несколько"),
ERROR_ENTRY(125, "[LA]: Обнаружено несколько точек входа"),
ERROR_ENTRY(126, "[LA]: У точки входа нет тела"),
ERROR_ENTRY(127, "[LA]: Несогласованность скобок"),
ERROR_ENTRY(128, "[LA]: Необъявленный идентификатор"),
ERROR_ENTRY(129, "[LA]: Переопределение идентификатора"),
ERROR_ENTRY(131, "[LA]: Слишком длинное имя идентификатора"),
```

Листинг 3.1 – Перечень ошибок лексического анализатора

Также сам текст ошибки содержит в себе префикс [LA].

3.10 Принцип обработки ошибок

Все ошибки являются критическими и приводят к прекращению работы транслятора и выводу диагностического сообщения в лог файл.

3.11 Контрольный пример

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

Глава 4 Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций. Структура синтаксического анализатора представлена на рисунке 4.1.

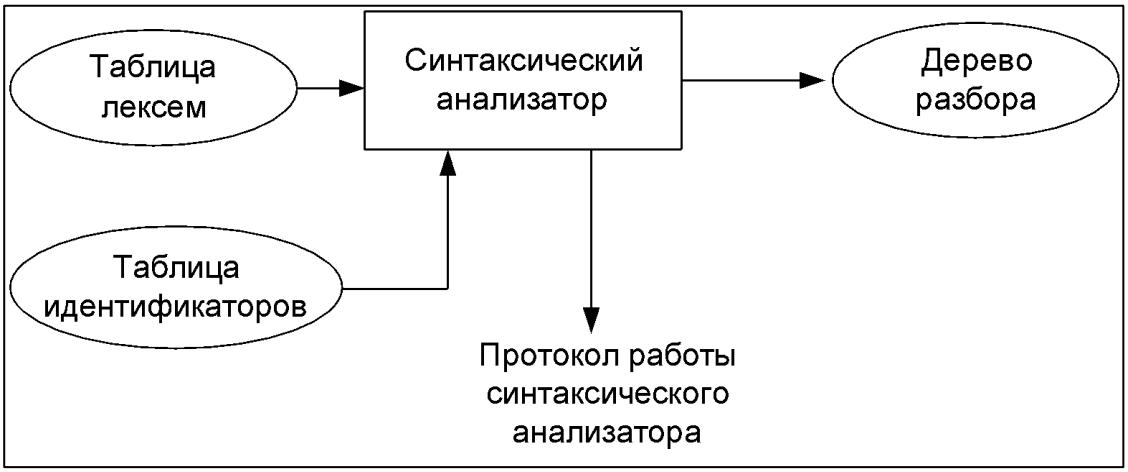

Рисунок 4.1 – Структура синтаксического анализатора

Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выходом – дерево разбора.

4.2 Контекстно свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка GED-2024 используется контекстно-свободная грамматика $G = \langle T, N, P, S \rangle$, где

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

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

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

Грамматика языка GED-2024 представлена в приложении Б.

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

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

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов GED-2024

Нетерминал	Цепочки правил	Описание
S	m{NrE;} m{rE;} tfi(F){NrE;}S	Порождает правила, описывающее общую структуру программы

N	dti;
	dti;N
	dtfi(F);
	dtfi(F);N
	dtfi();
	dtfi()N;
	i=E;
	i=E;N
	pi;
	pi;N
	pl;
	pl;N
	pi(W);
	pi(W);N
	T(C)[N]
	T(C)[N]N
	e(C)[N]
	e(C)[N]
	rE;
		Порождает правила, описывающие конструкции языка

E	i l (E) (E)M i() i()E iM lM (E)M i(W)M	Порождает правила, описывающие выражения
F	ti ti,f	Порождает правила, описывающие параметры локальной функции при её объявлении

Продолжение таблицы 4.1

Нетерминал	Цепочки правил	Описание
W	i l i,W l,W	Порождает правила, описывающие принимаемые параметры функции
C	i l i&l i|i i&i i|l	Порождает правила, описывающие условное выражение в операторе цикла
M	vE v(E) v(E)M vEM	Порождает правила, описывающие знаки арифметических операций (v – знаки операций: +,-,*,|,~,&)

В таблице 4.1 представлено описание нетерминальных символов и соответствующих правил переходов в контекстно-свободной грамматике языка GED-2024.

4.3 Построение конечного магазинного автомата

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

Таблица 4.2 – Описание компонентов магазинного автомата

Компонента	Определение	Описание
Q	Множество состояний автомата	Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата
V	Алфавит входных символов	Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1.
δ	Функция переходов автомата	Функция представляет из себя множество правил грамматики, описанных в таблице 4.1.
Z	Алфавит специальных магазинных символов	Алфавит магазинных символов содержит стартовый символ и маркер дна стека

Продолжение таблицы 4.2

Компонента	Определение	Описание
q_0	Начальное состояние автомата	Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ A)
z_0	Начальное состояние магазина автомата	Символ маркера дна стека ($)
F	Множество конечных состояний	Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты

Протокол и ошибки работы синтаксического анализатора выводятся в лог журнал.

4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного конечного автомата и структуру грамматики Грейбах, описывающей правила языка GED-2024. Данные структуры представлены в Приложении В.

4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата:

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

4.6 Параметры синтаксического анализатора и режимы его работы

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

4.7 Структура и перечень сообщений синтаксического анализатора

Индексы ошибок, обнаруживаемых синтаксическим анализатором, находятся в диапазоне 600-606. Перечень сообщений синтаксического анализатора представлен в листинге 4.1.

```
ERROR_ENTRY(600, "[SA]: Неверная структура программы"),
ERROR_ENTRY(601, "[SA]: Ошибочный оператор"),
ERROR_ENTRY(602, "[SA]: Ошибка в выражении"),
ERROR_ENTRY(603, "[SA]: Ошибка в параметрах функции"),
ERROR_ENTRY(604, "[SA]: Ошибка в параметрах вызываемой функции"),
ERROR_ENTRY(605, "[SA]: Ошибка в подвыражении"),
ERROR_ENTRY(606, "[SA]: Ошибка в if"),
```

Листинг 4.1 – Перечень сообщений синтаксического анализатора

Также сам текст ошибки содержит в себе префикс [SA].

4.8 Принцип обработки ошибок

Обработка ошибок происходит следующим образом:

- Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
- Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
- В случае ошибки выводится соответствующее сообщение в журнал лога и компилятор прекращает работу.

4.9 Контрольный пример

Пример разбора синтаксическим анализатором исходного кода на языке GED-2024 представлен в Приложении Г. Дерево разбора исходного кода приложен к проекту.

Глава 5 Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализ происходит при выполнении фазы лексического анализа и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях: установки флага или нахождении в особом месте программы (оператор выхода из функции, оператор ветвления, вызов функции стандартной библиотеки) [5]. Структура семантического анализатора представлена на рисунке 5.1.

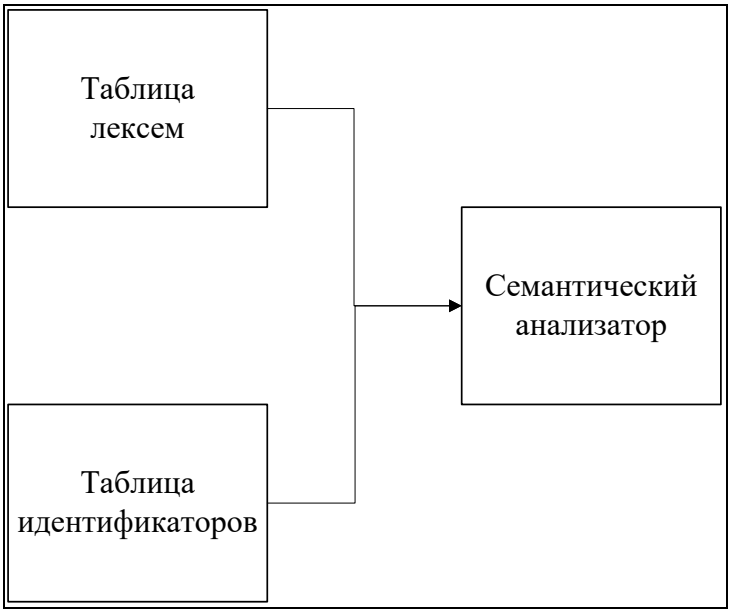

Рисунок 5.1 – структура семантического анализатор

5.2 Функции семантического анализатора

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

5.3 Структура и перечень сообщений семантического анализатора

Все ошибка семантического анализатора имеют идентификатор свыше 700. Сообщения, формируемые семантическим анализатором, представлены в Листинге 5.1.

```
        ERROR_ENTRY(700, "[SemA]: Ошибка в возвращаемом значении"),
        ERROR_ENTRY(701, "[SemA]: Ошибка в параметрах функции"),
        ERROR_ENTRY(702, "[SemA]: Ошибка в параметрах функции
библиотеки"),
        ERROR_ENTRY(704, "[SemA]: Ошибка в значении для вывода в консоль"),
        ERROR_ENTRY(705, "[SemA]: Несоответствие возвращаемого и
присваемого типов данных"),
        ERROR_ENTRY(706, "[SemA]: Несоотвтсвие присваимаего типа
данных"),
        ERROR_ENTRY(707, "[SemA]: Несоответствие типа функции и
```

```
возвращаемого значения"),
        ERROR_ENTRY(708, "[SemA]: Main Должен возвращать числовое
значение"),
        ERROR_ENTRY(709, "[SemA]: Использование имени переменной в
качестве функции"),
         ERROR_ENTRY(710, "[SemA]: Ошибка в условном операторе"),
```

Листинг 5.1 – Перечень сообщений семантического анализатора

Все ошибки имеют приставку [SemA].

5.4 Принцип обработки ошибок

При обнаружении хотя бы одной ошибки транслятор завершит свою работу с Записью информации об ошибке в лог файл.

5.5 Контрольный пример

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

Глава 6 Преобразование выражений

6.1 Выражения, допускаемые языком

В языке GED-2024 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как: +, -, * и (). А также вызовы функций как операнды арифметических выражений. Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке GED-2024

Операция	Значение приоритета
(	0
)	0
,	1
-	2
+	2
*	3
|	3
~	3
&	3

Все операции используются только для целочисленного типа данных.

6.2 Польская запись и принцип её построения

Выражения в языке GED-2024 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок [2].

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

Алгоритм построения:

– читаем очередной символ;

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

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

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

– исходная строка просматривается слева направо;

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

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

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

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

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись.

Исходная строка	Результирующая строка	Стек
x+y*5/(z-2)		
+y*5/(z-2)	x	
y*5/(z-2)	x	+
*5/(z-2)	xy	+
5/(z-2)	xy	+*
/(z-2)	xy5	+*
(z-2)	xy5*	+/
z-2)	xy5*	+/(
-2)	xy5*z	+/(
2)	xy5*z	+/(-
)	xy5*z2	+/(-
	xy5*z2-	+/
	xy5*z2-/	+
	xy5*z2-/+	

Все выражения могут быть преобразованы в обратную польскую запись по такому же принципу.

6.3 Программная реализация обработки выражений

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

6.4 Контрольный пример

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

Глава 7 Генерация кода

7.1 Структура генератора кода

Генерация объектного кода – это перевод компилятором внутреннего представления исходной программы GED-2024 в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

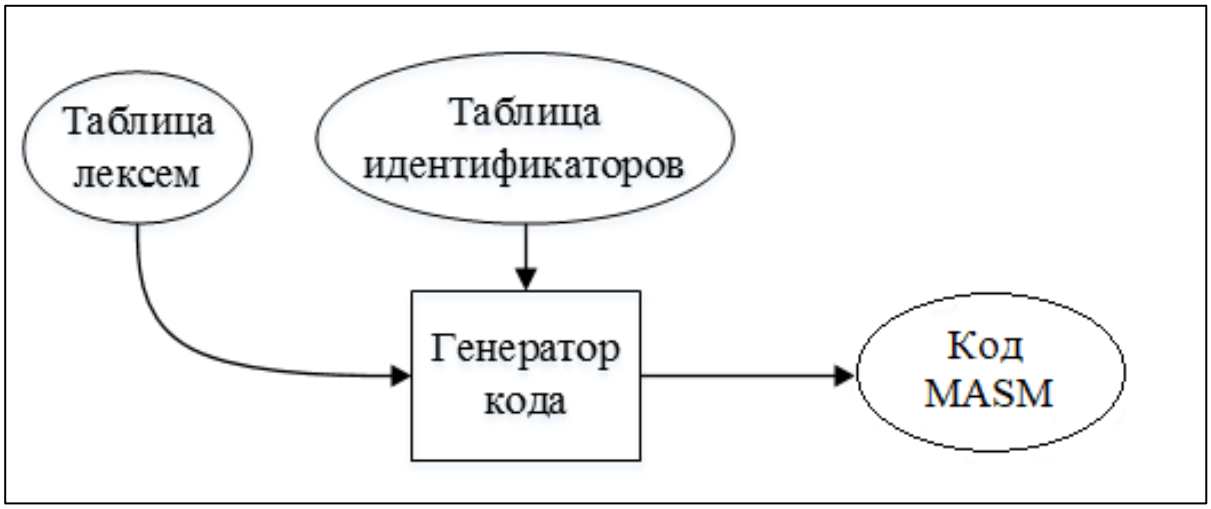

Рисунок 7.1 — Структура генератора кода

7.2 Представление типов данных в оперативной памяти

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

Таблица 7.1 – Соответствия типов идентификаторов языка GED-2024 и языка Ассемблера

Тип идентификатора на языке GED-2024	Тип идентификатора на языке ассемблера	Пояснение
symb	BYTE	Хранит символьный тип данных
int	SDWORD	Хранит беззнаковый целый тип данных.
Лексема	BYTE SDWORD	Литералы: целочисленные, строковые

Идентификаторы языка GED-2024 размещены в сегменте данных (.data). Литералы – в сегменте констант (.const).

7.3 Статическая библиотека

В языке GED-2024 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера.

Таблица 7.2 – Функции статической библиотеки

Функция	Назначение
int stdcall step(int int1, int int2)	Возведение int1 в степень int2
int stdcall iabs(long aint)	Вычисляет абсолютное значение
void stdcall ConsoleWriteInt(int number)	Выводит значение на консоль
void stdcall output(const char* buffer)	Принимает аргумент типа const char*(buffer) представляющий строку и выводит ее значение на консоль

В итоге, таблица 7.2 описывает функции, доступные в статической библиотеке языка GED-2024. Эти функции могут быть полезны при работе с выводом на консоль, а также выполняют математические операции.

7.4 Особенности алгоритма генерации кода

Алгоритм генерации кода выглядит следующим образом:

- Генерирует заголовочную информацию (Листинг 7.1): модель памяти, подключение библиотек, прототипы внешних функций, размер стека.

```
1) .586
2) .model flat, stdcall
3) includelib libucrt.lib
4) includelib kernel32.lib
5) includelib ../Debug/GED_Lib.lib
6) ExitProcess PROTO : DWORD
7) ConsoleWriteInt    PROTO : SDWORD
8) iabs PROTO      :     SDWORD
9) step PROTO    :   SDWORD,     : SDWORD
10) output PROTO       :   SDWORD
11) stack 4096
```

Листинг 7.1 – пример заголовочной информации

- Проходит полностью таблицу идентификаторов и заполняет поле .const литералами. Результат представлен в листинге 7.2.

```
.const
```

```
l0 SDWORD 4
l1 SDWORD 6
l2 SDWORD 1
l3 SDWORD 0
l4 BYTE '... func step ...', 0
l5 SDWORD 2
l6 BYTE '... summ 4 and 6 ...', 0
l7 BYTE '... sub 4 and 6 ...', 0
l8 BYTE '... func iabs ...', 0
l9 BYTE '.. begin ..', 0
l10 BYTE '.. later ..', 0
l11 BYTE '... mul 4 and 6 ...', 0
l12 BYTE '... summ hex ...', 0
l13 SDWORD 10
l14 SDWORD 31
l15 BYTE '... AND 1 and 0 ...', 0
l16 BYTE '... OR 1 and 0 ...', 0
l17 BYTE '... NOT a ...', 0
l18 BYTE ' ... func expression ...', 0
l19 BYTE 'W', 0
l20 BYTE '... Out symb ...', 0
l21 BYTE '... If and else ...', 0
l22 BYTE 'if true', 0
l23 BYTE 'else', 0
l24 BYTE 'if', 0
```

Листинг 7.2 – Пример заполнения поля .const

- Проходим таблицу идентификаторов и объявляем переменные в поле .data. Результат заполнения поля .data представлен в листинге 7.3.

```
.data
    expressionsm        SDWORD 0
    maina               SDWORD 0
    mainb               SDWORD 0
    mainc               SDWORD 0
    maind               SDWORD 0
    mainz               SDWORD 0
    mainone             SDWORD 0
    mainzero            SDWORD 0
    mainsa              SDWORD 0
```

Листинг 7.3 – Пример заполнения поля .data

Генерируем сегмент данных .code. Сперва проходим по таблице идентификаторов и ищем функции. Объявляем их и генерируем код, содержащийся в функциях. При генерации кода, при встрече оператора присваивания, описываем вычисление выражения. Описание алгоритма преобразования выражений представлено в пункте 7.3. Пример сгенерированной функции представлен в листинге 7.4.

```
.code
expression PROC b: SDWORD, a: SDWORD
        push        a
        push        b
        ;\/(MUL)*\/
        pop         eax
        pop         ebx
        MUL         ebx
        push        eax
        mov         eax, expressionsm
        ret
expression ENDP
```

Листинг 7.4 – Пример функции, полученной в результате генерации

После генерации всех пользовательских функций, генерируется функция начала программы main по такому же принципу.

7.5 Параметры, управляющие генерацией кода

На вход генератору кода поступают таблицы лексем и идентификаторов исходного кода программы на языке GED-2024. Результаты работы генератора кода выводятся в файл с расширением .asm.

7.6 Контрольный пример

Генерируемый код записывается в файл «in.txt.asm». Сгенерированный код можно посмотреть в Приложении Е.

Глава 8 Тестирование транслятора

8.1 Общие положения

Тестирование должно покрывать как можно больше сценариев использования языка и его конструкций. Все тесты были представлены для типичных ошибок пользователей при использовании языка. Когда компилятор обнаруживает ошибку, он записывает информацию о ней в протокол, содержащий номер ошибки и диагностическое сообщение, помогающее разработчику понять причину ошибки компиляции. После обнаружения ошибки компилятор может продолжить анализ, чтобы найти другие возможные ошибки. Результаты тестирования записываются в файл .log.

8.2 Результаты тестирования

В языке GED-2024 не разрешается использовать запрещённые входным алфавитом символы где-либо кроме строковых или символьных переменных. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

Исходный код	Диагностическое сообщение
aint	Ошибка 111: Недопустимый символ в исходном файле (-in), позиция 0, строка 0

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

Таблица 8.2 – Тестирование лексического анализатора

Исходный код	Диагностическое сообщение
write f;	f - 21 ---> Ошибка 128 [LA]: Необъявленный идентификатор
{}	Ошибка 124 [LA]: Отсутствует точка входа или их несколько в функции
main{ main{} }	Ошибка 124 [LA]: Отсутствует точка входа или их несколько в функции
set int a; set symb a;	a - 10 ---> Ошибка 129 [LA]: Переопределение идентификатора
set int qwertyqwertyqwerty;	Ошибка 131 [LA]: Слишком длинное имя идентификатора

Ошибка лексического анализатора приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

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

Таблица 8.3 – Тестирование синтаксического анализатора

Исходный код	Диагностическое сообщение

while(flag)
z = a;
];
	606: строка 48,[SA]: Ошибка в while

c a+b;	601: строка 26,[SA]: Ошибочный оператор

Ошибка синтаксического анализатора также приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

Исходный код	Диагностическое сообщение
set symb a; set int b; c = st(a, b);	Ошибка 706 [SemA]: Несоответсвие присваимаего типа данных, строка 20
int func sum(int a, int b){ set symb f; ret f;}	Ошибка 707 [SemA]: Несоответствие типа функции и возвращаемого значения, строка 4
set symb f; f =1;	Ошибка 706 [SemA]: Несоответсвие присваимаего типа данных, строка 17

main{
set symb f;
ret f;
};
	Ошибка 708 [SemA]: Main должен возвращать числовое значение, строка 5

Ошибка семантического анализатора также приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

Заключение

По окончании выполнения всех пунктов, изложенных ранее, получили рабочий транслятор языка программирования GED-2024 на язык ассемблера.

Таким образом, были выполнены основные задачи данной курсовой работы:

- Сформулирована спецификация языка GED-2024;
- 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;
- Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
- Разработан транслятор кода на язык ассемблера;
- Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка GED-2024 включает:

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

Список используемых источников

1. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва: Вильямс, 2003. – 429 с.
2. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – М.: Вильямс, 2003. – 768 с.
3. Прата, С. Язык программирования C++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 с.
4. Принципы работы транслятора [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/435520/. – Дата доступа: 25.11.2024.
5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с.

ПРИЛОЖЕНИЕ А

```
int func express(int a, int b){
 set int summab;
 summab = a + b;
 ret summab;
}

int func expression(int a, int b){
 set int sm;
 sm =  a + (express(a,b) + express(a,b) + (b - a)) - b * a;
 ret sm;
}

main{
 set int func step(int x, int y);
 set int func iabs(int x);
 set int a;
 set int b;
 set int c;
 set int d;
 set int z;
 set int one;
 set int zero;

 a = 4;
 b = 6;
 one = 1;
 zero = 0;

 write '... func step ...';
 c = step(a,2);
 write c;
 write '... summ 4 and 6 ...';
 c = a + b;
 write c;
 write '... sub 4 and 6 ...';
 c = b - a;
 write c;
 write '... func iabs ...';
 d = iabs(c);
 write '.. begin ..';
 write c;
 write '.. later ..';
 write d;
 write '... mul 4 and 6 ...';
 c = b * a;
 write c;
 write '... summ hex ...';
 a = 0x0A;
 b = 0x1F;
 c = b + a;
```

```
 write c;

 write '... AND 1 and 0 ...';
 z = zero & one;
 write z;
 write '... OR 1 and 0 ...';
 z = zero | one;
 write z;
 write '... NOT a ...';
 z = a ~ 4;
 write z;

 write ' ... func expression ...';
 set int result;
 result = expression(2, 6);
 write result;

 set symb sa;
 sa = 'W';

 write '... Out symb ...';
 write sa;

 write '... If and else ...';

 set int y;
 y = 47;
 while(y)
 [
     write '10';
 ]
 ret 0;
}
```

Листинг А.1 – Контрольный пример

```
+-+-+-+-+-+-+-+-+- Таблица идентификаторов +-+-+-+-+-+-+-+-+-
1       ---->           express - a - INT - PARAMETER - 5 - 0
2       ---->           express - b - INT - PARAMETER - 8 - 0
3       ---->           express - summab - INT - VARIABLE - 13 - 0
4       ---->            - expression - INT - FUNCTION - 27 - 0
5       ---->           expression - sm - INT - VARIABLE - 38 - 0
6       ---->            - main - INT - FUNCTION - 74 - 0
7       ---->            - step - INT - FUNCTION - 79 - 0
8       ---->           step - x - INT - PARAMETER - 82 - 0
9       ---->           step - y - INT - PARAMETER - 85 - 0
10      ---->            - iabs - INT - FUNCTION - 91 - 0
11      ---->           iabs - x - INT - PARAMETER - 94 - 0
12      ---->           main - a - INT - VARIABLE - 99 - 0
13      ---->           main - b - INT - VARIABLE - 103 - 0
14      ---->           main - c - INT - VARIABLE - 107 - 0
15      ---->           main - d - INT - VARIABLE - 111 - 0
16      ---->           main - z - INT - VARIABLE - 115 - 0
```

```
17      ---->          main - one - INT - VARIABLE - 119 - 0
18      ---->          main - zero - INT - VARIABLE - 123 - 0
19      ---->          l0 - INT - LITERAL - 127 - 4
20      ---->          l1 - INT - LITERAL - 131 - 6
21      ---->          l2 - INT - LITERAL - 135 - 1
22      ---->          l3 - INT - LITERAL - 139 - 0
23      ---->          l4 - STR - LITERAL - 142 - '... func step ...'
24      ---->          l5 - INT - LITERAL - 150 - 2
25      ---->          l6 - STR - LITERAL - 157 - '... summ 4 and 6 ...'
26      ---->          l7 - STR - LITERAL - 169 - '... sub 6 and 4 ...'
27      ---->          l8 - STR - LITERAL - 181 - '... func iabs ...'
28      ---->          l9 - STR - LITERAL - 191 - '.. begin ..'
29      ---->          l10 - STR - LITERAL - 197 - '.. later ..'
30      ---->          l11 - STR - LITERAL - 203 - '... mul 4 and 6 ...'
31      ---->          l12 - STR - LITERAL - 215 - '... summ hex ...'
32      ---->          l13 - INT - LITERAL - 219 - 10
33      ---->          l14 - INT - LITERAL - 223 - 31
34      ---->          l15 - STR - LITERAL - 235 - '... AND 1 and 0 ...'
35      ---->          l16 - STR - LITERAL - 247 - '... OR 1 and 0 ...'
36      ---->          l17 - STR - LITERAL - 259 - '... NOT a ...'
37      ---->          l18 - STR - LITERAL - 271 - ' ... func expression ...'
38      ---->          main - result - INT - VARIABLE - 275 - 0
39      ---->          main - sa - CHAR - VARIABLE - 291 -
40      ---->          l19 - CHAR - LITERAL - 295 - 'W'
41      ---->          l20 - STR - LITERAL - 298 - '... Out symb ...'
42      ---->          l21 - STR - LITERAL - 304 - '... If and else ...'
43      ---->          main - y - INT - VARIABLE - 308 - 0
44      ---->          l22 - INT - LITERAL - 312 - 47
45      ---->          l23 - STR - LITERAL - 320 - '10'
```

Листинг А.2 – Таблица идентификаторов

```
+-+-+-+-+-+-+-+-+- Таблица лексем +-+-+-+-+-+-+-+-+-
1       ---->          -1 - f - 0
2       ---->          0 - i - 0
3       ---->          -1 - ( - 0
4       ---->          -1 - t - 0
5       ---->          1 - i - 0
6       ---->          -1 - , - 0
7       ---->          -1 - t - 0
8       ---->          2 - i - 0
9       ---->          -1 - ) - 0
10      ---->          -1 - { - 0
11      ---->          -1 - d - 1
12      ---->          -1 - t - 1
13      ---->          3 - i - 1
14      ---->          -1 - ; - 1
15      ---->          3 - i - 2
16      ---->          -1 - = - 2
17      ---->          1 - i - 2
163     ---->          13 - i - 32
164     ---->          -1 - ; - 32
165     ---->          -1 - p - 33
166     ---->          14 - i – 33
167     ---->          -1 - ; - 33
168     ---->          -1 - p - 34
169     ---->          26 - l - 34
170     ---->          -1 - ; - 34
171     ---->          14 - i - 35
172     ---->          -1 - = - 35
173     ---->          12 - i - 35
174     ---->          -1 - - - 35
175     ---->          13 - i - 35
176     ---->          -1 - ; - 35
177     ---->          -1 - p - 36
178     ---->          14 - i - 36
179     ---->          -1 - ; - 36
180     ---->          -1 - p - 37
181     ---->          27 - l - 37
```

18	---->	-1 - + - 2	182	---->	-1 - ; - 37
19	---->	2 - i - 2	183	---->	15 - i - 38
20	---->	-1 - ; - 2	184	---->	-1 - = - 38
21	---->	-1 - r - 3	185	---->	10 - i - 38
22	---->	3 - i - 3	186	---->	-1 - (- 38
23	---->	-1 - ; - 3	187	---->	14 - i - 38
24	---->	-1 - } - 4	188	---->	-1 -) - 38
25	---->	-1 - t - 5	189	---->	-1 - ; - 38
26	---->	-1 - f - 6	190	---->	-1 - p - 39
27	---->	4 - i - 6	191	---->	28 - l - 39
28	---->	-1 - (- 6	192	---->	-1 - ; - 39
29	---->	-1 - t - 6	193	---->	-1 - p - 40
30	---->	1 - i - 6	194	---->	14 - i - 40
31	---->	-1 - , - 6	195	---->	-1 - ; - 40
32	---->	-1 - t - 6	196	---->	-1 - p - 41
33	---->	2 - i - 6	197	---->	29 - l - 41
34	---->	-1 -) - 6	198	---->	-1 - ; - 41
35	---->	-1 - { - 6	199	---->	-1 - p - 42
36	---->	-1 - d - 7	200	---->	15 - i - 42
37	---->	-1 - t - 7	201	---->	-1 - ; - 42
38	---->	5 - i - 7	202	---->	-1 - p - 43
39	---->	-1 - ; - 7	203	---->	30 - l - 43
40	---->	5 - i - 8	204	---->	-1 - ; - 43
41	---->	-1 - = - 8	205	---->	14 - i - 44
42	---->	1 - i - 8	206	---->	-1 - = - 44
43	---->	-1 - + - 8	207	---->	13 - i - 44
44	---->	-1 - (- 8	208	---->	-1 - * - 44
45	---->	0 - i - 8	209	---->	12 - i - 44
46	---->	-1 - (- 8	210	---->	-1 - ; - 44
47	---->	1 - i - 8	211	---->	-1 - p - 45
48	---->	-1 - , - 8	212	---->	14 - i – 45
49	---->	2 - i - 8	213	---->	-1 - ; - 45
50	---->	-1 -) - 8	214	---->	-1 - p - 46
51	---->	-1 - + - 8	215	---->	31 - l - 46
52	---->	0 - i - 8	216	---->	-1 - ; - 46
53	---->	-1 - (- 8	217	---->	12 - i - 47
54	---->	1 - i - 8	218	---->	-1 - = - 47
55	---->	-1 - , - 8	219	---->	32 - l - 47
56	---->	2 - i - 8	220	---->	-1 - ; - 47
57	---->	-1 -) - 8	221	---->	13 - i - 48
58	---->	-1 - + - 8	222	---->	-1 - = - 48
59	---->	-1 - (- 8	223	---->	33 - l - 48
60	---->	2 - i - 8	224	---->	-1 - ; - 48
61	---->	-1 - - - 8	225	---->	14 - i - 49
62	---->	1 - i - 8	226	---->	-1 - = - 49
63	---->	-1 -) – 8	227	---->	13 - i - 49

64	---->	-1 -) - 8	228	---->	-1 - + - 49
65	---->	-1 - - - 8	229	---->	12 - i - 49
66	---->	2 - i - 8	230	---->	-1 - ; - 49
67	---->	-1 - * - 8	231	---->	-1 - p - 50
68	---->	1 - i - 8	232	---->	14 - i - 50
69	---->	-1 - ; - 8	233	---->	-1 - ; - 50
70	---->	-1 - r - 9	234	---->	-1 - p - 51
71	---->	5 - i - 9	235	---->	34 - l - 52
72	---->	-1 - ; - 9	236	---->	-1 - ; - 52
73	---->	-1 - } - 10	237	---->	16 - i - 53
74	---->	6 - m - 11	238	---->	-1 - = - 53
75	---->	-1 - { - 12	239	---->	18 - i - 53
76	---->	-1 - d - 13	240	---->	-1 - & - 53
77	---->	-1 - t - 13	241	---->	17 - i - 53
78	---->	-1 - f - 13	242	---->	-1 - ; - 53
79	---->	7 - i - 13	243	---->	-1 - p - 54
80	---->	-1 - (- 13	244	---->	16 - i - 54
81	---->	-1 - t - 13	245	---->	-1 - ; - 54
82	---->	8 - i - 13	246	---->	-1 - p - 55
83	---->	-1 - , - 13	247	---->	35 - l - 55
84	---->	-1 - t - 13	248	---->	-1 - ; - 55
85	---->	9 - i - 13	249	---->	16 - i - 56
86	---->	-1 -) - 13	250	---->	-1 - = - 56
87	---->	-1 - ; - 13	251	---->	18 - i - 56
88	---->	-1 - d - 14	252	---->	-1 - | - 56
89	---->	-1 - t - 14	253	---->	17 - i - 56
90	---->	-1 - f - 14	254	---->	-1 - ; - 56
91	---->	10 - i - 14	255	---->	-1 - p - 57
92	---->	-1 - (- 14	256	---->	16 - i - 57
93	---->	-1 - t - 14	257	---->	-1 - ; - 57
94	---->	11 - i - 14	258	---->	-1 - p – 58
95	---->	-1 -) - 14	259	---->	36 - l - 58
96	---->	-1 - ; - 14	260	---->	-1 - ; - 58
97	---->	-1 - d - 15	261	---->	16 - i - 59
98	---->	-1 - t - 15	262	---->	-1 - = - 59
99	---->	12 - i - 15	263	---->	12 - i - 59
100	---->	-1 - ; - 15	264	---->	-1 - ~ - 59
101	---->	-1 - d - 16	265	---->	19 - l - 59
102	---->	-1 - t - 16	266	---->	-1 - ; - 59
103	---->	13 - i - 16	267	---->	-1 - p - 60
104	---->	-1 - ; - 16	268	---->	16 - i - 60
105	---->	-1 - d - 17	269	---->	-1 - ; - 60
106	---->	-1 - t - 17	270	---->	-1 - p - 61
107	---->	14 - i - 17	271	---->	37 - l - 62
108	---->	-1 - ; - 17	272	---->	-1 - ; - 62
109	---->	-1 - d – 18	273	---->	-1 - d - 63

110	---->	-1 - t - 18	274	---->	-1 - t - 63
111	---->	15 - i - 18	275	---->	38 - i - 63
112	---->	-1 - ; - 18	276	---->	-1 - ; - 63
113	---->	-1 - d - 19	277	---->	38 - i - 64
114	---->	-1 - t - 19	278	---->	-1 - = - 64
115	---->	16 - i - 19	279	---->	4 - i - 64
116	---->	-1 - ; - 19	280	---->	-1 - (- 64
117	---->	-1 - d - 20	281	---->	24 - l - 64
118	---->	-1 - t - 20	282	---->	-1 - , - 64
119	---->	17 - i - 20	283	---->	20 - l - 64
120	---->	-1 - ; - 20	284	---->	-1 -) - 64
121	---->	-1 - d - 21	285	---->	-1 - ; - 64
122	---->	-1 - t - 21	286	---->	-1 - p - 65
123	---->	18 - i - 21	287	---->	38 - i - 65
124	---->	-1 - ; - 21	288	---->	-1 - ; - 65
125	---->	12 - i - 22	289	---->	-1 - d - 66
126	---->	-1 - = - 23	290	---->	-1 - t - 67
127	---->	19 - l - 23	291	---->	39 - i - 67
128	---->	-1 - ; - 23	292	---->	-1 - ; - 67
129	---->	13 - i - 24	293	---->	39 - i - 68
130	---->	-1 - = - 24	294	---->	-1 - = - 68
131	---->	20 - l - 24	295	---->	40 - l - 68
132	---->	-1 - ; - 24	296	---->	-1 - ; - 68
133	---->	17 - i - 25	297	---->	-1 - p - 69
134	---->	-1 - = - 25	298	---->	41 - l - 70
135	---->	21 - l - 25	299	---->	-1 - ; - 70
136	---->	-1 - ; - 25	300	---->	-1 - p - 71
137	---->	18 - i - 26	301	---->	39 - i - 71
138	---->	-1 - = - 26	302	---->	-1 - ; - 71
139	---->	22 - l - 26	303	---->	-1 - p - 72
140	---->	-1 - ; - 26	304	---->	42 - l - 73
141	---->	-1 - p - 27	305	---->	-1 - ; - 73
142	---->	23 - l - 28	306	---->	-1 - d - 74
143	---->	-1 - ; - 28	307	---->	-1 - t - 75
144	---->	14 - i - 29	308	---->	43 - i - 75
145	---->	-1 - = - 29	309	---->	-1 - ; - 75
146	---->	7 - i - 29	310	---->	43 - i - 76
147	---->	-1 - (- 29	311	---->	-1 - = - 76
148	---->	12 - i - 29	312	---->	44 - l - 76
149	---->	-1 - , - 29	313	---->	-1 - ; - 76
150	---->	24 - l - 29	314	---->	-1 - e - 77
151	---->	-1 -) - 29	315	---->	-1 - (- 77
152	---->	-1 - ; - 29	316	---->	43 - i - 77
153	---->	-1 - p - 30	317	---->	-1 -) - 77
154	---->	14 - i - 30	318	---->	-1 - [- 78
155	---->	-1 - ; - 30	319	---->	-1 - p - 79

```
156   ---->        -1 - p - 31
157   ---->        25 - l - 31
158   ---->        -1 - ; - 31
159   ---->        14 - i - 32
160   ---->        -1 - = - 32
161   ---->        12 - i - 32
162   ---->        -1 - + - 32
320   ---->        45 - l - 79
321   ---->        -1 - ; - 79
322   ---->        -1 - ] - 80
323   ---->        -1 - r - 81
324   ---->        22 - l - 81
325   ---->        -1 - ; - 81
326   ---->        -1 - } - 82
```

Листинг А.3 – Таблица лексем

```
#define FST_MAIN FST(5, LEX_MAIN,\
       NODE(1, RELATION('m', 1)),\
       NODE(1, RELATION('a', 2)),\
       NODE(1, RELATION('i', 3)),\
       NODE(1, RELATION('n', 4)),\
       NODE())

#define FST_PRINT FST(6, LEX_PRINT,\
       NODE(1, RELATION('w', 1)),\
       NODE(1, RELATION('r', 2)),\
       NODE(1, RELATION('i', 3)),\
       NODE(1, RELATION('t', 4)),\
       NODE(1, RELATION('e', 5)),\
       NODE())

#define FST_RETURN FST(4, LEX_RETURN,\
       NODE(1, RELATION('r', 1)),\
       NODE(1, RELATION('e', 2)),\
       NODE(1, RELATION('t', 3)),\
       NODE())

#define FST_DECLARE FST(4, LEX_DECLARE,\
       NODE(1, RELATION('s', 1)),\
       NODE(1, RELATION('e', 2)),\
       NODE(1, RELATION('t', 3)),\
       NODE())

#define FST_INTEGER FST(4, LEX_INTEGER,\
       NODE(1, RELATION('i', 1)),\
       NODE(1, RELATION('n', 2)),\
       NODE(1, RELATION('t', 3)),\
       NODE())

#define FST_CHAR FST(5, LEX_CHAR,\
       NODE(1, RELATION('s', 1)),\
       NODE(1, RELATION('y', 2)),\
       NODE(1, RELATION('m', 3)),\
       NODE(1, RELATION('b', 4)),\
       NODE())

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

```
       RELATION('B', 0), RELATION('B', 1),\
       RELATION('C', 0), RELATION('C', 1),\
       RELATION('D', 0), RELATION('D', 1),\
       RELATION('E', 0), RELATION('E', 1),\
       RELATION('F', 0), RELATION('F', 1),\
       RELATION('G', 0), RELATION('G', 1),\
       RELATION('H', 0), RELATION('H', 1),\
       RELATION('I', 0), RELATION('I', 1),\
       RELATION('J', 0), RELATION('J', 1),\
       RELATION('K', 0), RELATION('K', 1),\
       RELATION('L', 0), RELATION('L', 1),\
       RELATION('M', 0), RELATION('M', 1),\
       RELATION('N', 0), RELATION('N', 1),\
       RELATION('R', 0), RELATION('R', 1),\
       RELATION('S', 0), RELATION('S', 1),\
       RELATION('T', 0), RELATION('T', 1),\
       RELATION('U', 0), RELATION('U', 1),\
       RELATION('V', 0), RELATION('V', 1),\
       RELATION('W', 0), RELATION('W', 1),\
       RELATION('X', 0), RELATION('X', 1),\
       RELATION('Y', 0), RELATION('Y', 1),\
       RELATION('Z', 0), RELATION('Z', 1),\
       RELATION('1', 0), RELATION('1', 1),\
       RELATION('2', 0), RELATION('2', 1),\
       RELATION('3', 0), RELATION('3', 1),\
       RELATION('4', 0), RELATION('4', 1),\
       RELATION('5', 0), RELATION('5', 1),\
       RELATION('6', 0), RELATION('6', 1),\
       RELATION('7', 0), RELATION('7', 1),\
       RELATION('8', 0), RELATION('8', 1),\
       RELATION('9', 0), RELATION('9', 1),\
       RELATION('0', 0), RELATION('0', 1)\
),\
NODE())
#define FST_STRING_LITERAL FST(3,
LEX_LITERAL,\
              NODE(1, RELATION('\'', 1)),\
              NODE(135,\
              RELATION('A', 1),\
              RELATION('B', 1),\
              RELATION('C', 1),\
              RELATION('D', 1),\
              RELATION('E', 1),\
              RELATION('F', 1),\
              RELATION('G', 1),\
              RELATION('H', 1),\
              RELATION('I', 1),\
```

```
#define FST_IF FST(6, LEX_IF,\
       NODE(1, RELATION('w', 1)),\
       NODE(1, RELATION('h', 2)),\
       NODE(1, RELATION('i', 2)),\
       NODE(1, RELATION('l', 2)),\
       NODE(1, RELATION('e', 2)),\
       NODE())

#define FST_ELSE FST(5, LEX_ELSE,\
       NODE(1, RELATION('e', 1)),\
       NODE(1, RELATION('l', 2)),\
       NODE(1, RELATION('s', 3)),\
       NODE(1, RELATION('e', 4)),\
       NODE())

#define FST_LEFT_BRACKET FST(2,
LEX_LEFTHESIS,\
       NODE(1, RELATION('(', 1)),\
       NODE()\
)
#define FST_LEFTSQUARE FST(2,
LEX_LEFTSQUARE,\
       NODE(1, RELATION('[', 1)),\
       NODE()\
)
#define FST_RIGHTSQUARE FST(2,
LEX_RIGHTSQUARE,\
       NODE(1, RELATION(']', 1)),\
       NODE()\
)
#define FST_RIGHT_BRACKET FST(2,
LEX_RIGHTHESIS,\
       NODE(1, RELATION(')', 1)),\
       NODE()\
)
#define FST_LEFT_BRACE FST(2,
LEX_RIGHTBRACE,\
       NODE(1, RELATION('{', 1)),\
       NODE()\
)
#define FST_RIGHT_BRACE FST(2, LEX_BRACELET,\
       NODE(1, RELATION('}', 1)),\
       NODE()\
)
#define FST_PLUS FST(2, LEX_PLUS,\
       NODE(1, RELATION('+', 1)),\
       NODE()\
)
#define FST_MINUS FST(2, LEX_MINUS,\
       NODE(1, RELATION('-', 1)),\
       NODE()\
)
#define FST_MULTI FST(2, LEX_STAR,\
       NODE(1, RELATION('*', 1)),\
       NODE()\
)

#define FST_AND FST(2, LEX_AND,\
       NODE(1, RELATION('&', 1)),\
       NODE()\
)
#define FST_OR FST(2, LEX_OR,\
       NODE(1, RELATION('|', 1)),\
       NODE()\
)
#define FST_INVERSE FST(2, LEX_INVERSE,\
                    RELATION('J', 1),\
                    RELATION('K', 1),\
                    RELATION('L', 1),\
                    RELATION('M', 1),\
                    RELATION('N', 1),\
                    RELATION('O', 1),\
                    RELATION('P', 1),\
                    RELATION('Q', 1),\
                    RELATION('R', 1),\
                    RELATION('S', 1),\
                    RELATION('T', 1),\
                    RELATION('U', 1),\
                    RELATION('V', 1),\
                    RELATION('W', 1),\
                    RELATION('X', 1),\
                    RELATION('Y', 1),\
                    RELATION('Z', 1),\
                    RELATION('a', 1),\
                    RELATION('b', 1),\
                    RELATION('c', 1),\
                    RELATION('d', 1),\
                    RELATION('e', 1),\
                    RELATION('f', 1),\
                    RELATION('g', 1),\
                    RELATION('h', 1),\
                    RELATION('i', 1),\
                    RELATION('j', 1),\
                    RELATION('k', 1),\
                    RELATION('l', 1),\
                    RELATION('m', 1),\
                    RELATION('n', 1),\
                    RELATION('o', 1),\
                    RELATION('p', 1),\
                    RELATION('q', 1),\
                    RELATION('r', 1),\
                    RELATION('s', 1),\
                    RELATION('t', 1),\
                    RELATION('u', 1),\
                    RELATION('v', 1),\
                    RELATION('w', 1),\
                    RELATION('x', 1),\
                    RELATION('y', 1),\
                    RELATION('z', 1),\
                    RELATION('A', 1),\
                    RELATION('Б', 1),\
                    RELATION('B', 1),\
                    RELATION('Г', 1),\
                    RELATION('Д', 1),\
                    RELATION('E', 1),\
                    RELATION('Ж', 1),\
                    RELATION('З', 1),\
                    RELATION('И', 1),\
                    RELATION('Й', 1),\
                    RELATION('K', 1),\
                    RELATION('Л', 1),\
                    RELATION('M', 1),\
                    RELATION('H', 1),\
                    RELATION('O', 1),\
                    RELATION('П', 1),\
                    RELATION('P', 1),\
                    RELATION('C', 1),\
                    RELATION('T', 1),\
                    RELATION('У', 1),\
                    RELATION('Ф', 1),\
                    RELATION('X', 1),\
                    RELATION('Ц', 1),\
```

```
	NODE(1, RELATION(':', 1)),\
	NODE()\
)
#define FST_COMMA FST(2, LEX_COMMA,\
	NODE(1, RELATION(',', 1)),\
	NODE()\
)
#define FST_SEMICOLON FST(2, LEX_SEMICOLON,\
	NODE(1, RELATION(';', 1)),\
	NODE()\
)
#define FST_EQUALLY FST(2, LEX_EQUAL,\
	NODE(1, RELATION('=', 1)),\
	NODE()\
)
#define FST_LOWER FST(2, LEX_LOWER,\
	NODE(1, RELATION('<', 1)),\
	NODE()\
)
#define FST_HIGHER FST(2, LEX_HIGHER,\
	NODE(1, RELATION('>', 1)),\
	NODE()\
)

#define FST_NUMBERS FST(1, LEX_LITERAL,\
	NODE(10, RELATION('0', 0),
RELATION('1', 0), RELATION('2', 0),
RELATION('3', 0), RELATION('4', 0),
RELATION('5', 0),\
	RELATION('6', 0), RELATION('7', 0),
RELATION('8', 0), RELATION('9', 0)),\
	NODE()\
)
#define FST_ID FST(2, LEX_ID, \
NODE(124,\
	RELATION('a', 0), RELATION('a', 1),\
	RELATION('b', 0), RELATION('b', 1),\
	RELATION('c', 0), RELATION('c', 1),\
	RELATION('d', 0), RELATION('d', 1),\
	RELATION('e', 0), RELATION('e', 1),\
	RELATION('f', 0), RELATION('f', 1),\
	RELATION('g', 0), RELATION('g', 1),\
	RELATION('h', 0), RELATION('h', 1),\
	RELATION('i', 0), RELATION('i', 1),\
	RELATION('j', 0), RELATION('j', 1),\
	RELATION('k', 0), RELATION('k', 1),\
	RELATION('l', 0), RELATION('l', 1),\
	RELATION('m', 0), RELATION('m', 1),\
	RELATION('n', 0), RELATION('n', 1),\
	RELATION('o', 0), RELATION('o', 1),\
	RELATION('s', 0), RELATION('s', 1),\
	RELATION('t', 0), RELATION('t', 1),\
	RELATION('u', 0), RELATION('u', 1),\
	RELATION('v', 0), RELATION('v', 1),\
	RELATION('w', 0), RELATION('w', 1),\
	RELATION('x', 0), RELATION('x', 1),\
	RELATION('y', 0), RELATION('y', 1),\
	RELATION('z', 0), RELATION('z', 1),\
	RELATION('A', 0), RELATION('A', 1),\
	RELATION('Щ', 1),\
	RELATION('ъ', 1),\
	RELATION('Ы', 1),\
	RELATION('ь', 1),\
	RELATION('Э', 1),\
	RELATION('Ю', 1),\
	RELATION('Я', 1),\
	RELATION('а', 1),\
	RELATION('б', 1),\
	RELATION('в', 1),\
	RELATION('г', 1),\
	RELATION('д', 1),\
	RELATION('е', 1),\
	RELATION('ж', 1),\
	RELATION('з', 1),\
	RELATION('и', 1),\
	RELATION('й', 1),\
	RELATION('к', 1),\
	RELATION('л', 1),\
	RELATION('м', 1),\
	RELATION('н', 1),\
	RELATION('о', 1),\
	RELATION('п', 1),\
	RELATION('р', 1),\
	RELATION('с', 1),\
	RELATION('т', 1),\
	RELATION('у', 1),\
	RELATION('ф', 1),\
	RELATION('х', 1),\
	RELATION('ц', 1),\
	RELATION('ч', 1),\
	RELATION('ш', 1),\
	RELATION('щ', 1),\
	RELATION('ъ', 1),\
	RELATION('ы', 1),\
	RELATION('ь', 1),\
	RELATION('э', 1),\
	RELATION('ю', 1),\
	RELATION('я', 1),\
	RELATION('0', 1),\
	RELATION('1', 1),\
	RELATION('2', 1),\
	RELATION('3', 1),\
	RELATION('4', 1),\
	RELATION('5', 1),\
	RELATION('6', 1),\
	RELATION('7', 1),\
	RELATION('8', 1),\
	RELATION('9', 1),\
	RELATION(',', 1),\
	RELATION('.', 1),\
	RELATION('!', 1),\
	RELATION('?', 1),\
	RELATION('"', 1),\
	RELATION('(', 1),\
	RELATION(')', 1),\
	RELATION(' ', 1),\
	RELATION('\'', 2)\
),\NODE())
```

Листинг А.4 – Конечные автоматы

ПРИЛОЖЕНИЕ Б

```
Greibach greibach(NS('S'), TS('$'),
                    7,
                    Rule(NS('S'), GRB_ERROR_SERIES + 0,
                           3, //m{NrE;};  m{rE;};     - main функция       tfi(F){NrE;};
tfi(F){NrE;};S       tfi(){NrE;}; tfi(){NrE;};S - обычные функции(с параметрами и без)
                           Rule::Chain(8, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'),
TS(';'), TS('}')),
                           Rule::Chain(7, TS('m'), TS('{'), 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(NS('N'), GRB_ERROR_SERIES + 1,
                           19, //dti;   rE;   i=E;  i=E;N   dtfi(F);   dti;N   rE;N   i=E;N
dtfi(F);N
                           Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),
                           Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),
                           Rule::Chain(8, TS('d'), TS('t'), TS('f'), TS('i'), TS('('),
NS('F'), TS(')'), TS(';')),
                           Rule::Chain(9, TS('d'), TS('t'), TS('f'), TS('i'), TS('('),
NS('F'), TS(')'), TS(';'), NS('N')),
                           Rule::Chain(7, TS('d'), TS('t'), TS('f'), TS('i'), TS('('),
TS(')'), TS(';')),
                           Rule::Chain(8, TS('d'), TS('t'), TS('f'), TS('i'), TS('('),
TS(')'), 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(3, TS('p'), TS('i'), TS(';')),
                           Rule::Chain(4, TS('p'), TS('i'), TS(';'), NS('N')),
                           Rule::Chain(3, TS('p'), TS('l'), TS(';')),
                           Rule::Chain(4, TS('p'), TS('l'), TS(';'), NS('N')),
                           Rule::Chain(6, TS('p'), TS('i'), TS('('), NS('W'), TS(')'),
TS(';')),
                           Rule::Chain(7, TS('p'), TS('i'), TS('('), NS('W'), TS(')'),
TS(';'), NS('N')),

                           Rule::Chain(7, TS('T'), TS('('), NS('C'), TS(')'), TS('['),
NS('N'), TS(']')),
                           Rule::Chain(8, TS('T'), TS('('), NS('C'), TS(')'), TS('['),
NS('N'), TS(']'), NS('N')),
                           Rule::Chain(7, TS('e'), TS('('), NS('C'), TS(')'), TS('['),
NS('N'), TS(']')),
                           Rule::Chain(8, TS('e'), TS('('), NS('C'), TS(')'), TS('['),
NS('N'), TS(']'), NS('N')),
                           Rule::Chain(3, TS('r'), NS('E'), TS(';'))
                    ),
                    Rule(NS('E'), GRB_ERROR_SERIES + 2,
                           11, //i   l   (E)   i(W)  s(W)  R(W)  c(W)  c()   iM   lM  Mi  Ml
(E)M   i(W)M
                           Rule::Chain(1, TS('i')),
                           Rule::Chain(1, TS('l')),
                           Rule::Chain(3, TS('('), NS('E'), TS(')')),
                           Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),
                           Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),
                           Rule::Chain(3, TS('i'), TS('('), TS(')')),
                           Rule::Chain(4, TS('i'), TS('('), TS(')'), NS('E')),
                           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('M'), GRB_ERROR_SERIES + 3,
                35, //vE    vEM   v(E) v(E)M
                Rule::Chain(2, TS('+'), NS('E')),
                Rule::Chain(4, TS('+'), TS('('), NS('E'), TS(')')),
                Rule::Chain(5, TS('+'), TS('('), NS('E'), TS(')'), NS('M')),
                Rule::Chain(3, TS('+'), NS('E'), NS('M')),
                Rule::Chain(2, TS('-'), NS('E')),
                Rule::Chain(4, TS('-'), TS('('), NS('E'), TS(')')),
                Rule::Chain(5, TS('-'), TS('('), NS('E'), TS(')'), NS('M')),
                Rule::Chain(3, TS('-'), NS('E'), NS('M')),
                Rule::Chain(2, TS('*'), NS('E')),
                Rule::Chain(4, TS('*'), TS('('), NS('E'), TS(')')),
                Rule::Chain(5, TS('*'), TS('('), NS('E'), TS(')'), NS('M')),
                Rule::Chain(3, TS('*'), NS('E'), NS('M')),
                Rule::Chain(2, TS('/'), NS('E')),
                Rule::Chain(4, TS('/'), TS('('), NS('E'), TS(')')),
                Rule::Chain(5, TS('/'), TS('('), NS('E'), TS(')'), NS('M')),
                Rule::Chain(3, TS('/'), NS('E'), NS('M')),
                Rule::Chain(1, TS(':')),
                Rule::Chain(1, TS('&')),
                Rule::Chain(1, TS('|')),
                Rule::Chain(2, TS('|'), NS('E')),
                Rule::Chain(4, TS('|'), TS('('), NS('E'), TS(')')),
                Rule::Chain(5, TS('|'), TS('('), NS('E'), TS(')'), NS('M')),
                Rule::Chain(3, TS('|'), NS('E'), NS('M')),

                Rule::Chain(2, TS('&'), NS('E')),
                Rule::Chain(4, TS('&'), TS('('), NS('E'), TS(')')),
                Rule::Chain(5, TS('&'), TS('('), NS('E'), TS(')'), NS('M')),
                Rule::Chain(3, TS('&'), NS('E'), NS('M'))
        ),
        Rule(NS('F'), GRB_ERROR_SERIES + 4,
                2, //ti   ti,F
                Rule::Chain(2, TS('t'), TS('i')),
                Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))
        ),
        Rule(NS('W'), GRB_ERROR_SERIES + 5,
                4, // i   l   i,W   l,W
                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('C'), GRB_ERROR_SERIES + 6,
                13, // для условий if
                Rule::Chain(3, TS('i'), TS('<'), TS('i')),
                Rule::Chain(3, TS('l'), TS('<'), TS('l')),
                Rule::Chain(3, TS('i'), TS('<'), TS('l')),
                Rule::Chain(3, TS('l'), TS('<'), TS('i')),
                Rule::Chain(3, TS('i'), TS('>'), TS('i')),
                Rule::Chain(3, TS('l'), TS('>'), TS('l')),
                Rule::Chain(3, TS('i'), TS('>'), TS('l')),
                Rule::Chain(3, TS('l'), TS('>'), TS('i')),
                Rule::Chain(3, TS('i'), TS('&'), TS('i'))));
```

Листинг Б.1 – Правила, описывающие грамматику языка

ПРИЛОЖЕНИЕ В

```
namespace MFST {
	struct MfstState {
		short lenta_position, nrule, nrulechain;
		MFSTSTSTACK st;
		MfstState();
		MfstState(short pposition, MFSTSTSTACK pst, short pnrulechain);
		MfstState(short pposition, MFSTSTSTACK pst, short pnrule, short pnrulechain);
	};

	struct Mfst {
		enum class RC_STEP { //! Код возврата функции step
			NS_OK,			//! Найдено правило и цепочка, цепочка 
записана в стек
			NS_NORULE,		//! Не найдено правило грамматики (ошибки в 
грамматике)
			NS_NORULECHAIN,		//! Не найдена подходящая цепочка правила 
(ошибка в исходном коде)
			NS_ERROR,		//! Неизвестный нетерминальный символ 
грамматики
			TS_OK,			//! Текущий символ ленты == вершине стека, 
продвинулась лента, pop стека
			TS_NOK,				//! Текущий символ ленты != вершине 
стека, восстановлено состояние
			LENTA_END,		//! Текущая позиция ленты >= lenta_size
			SURPRISE		//! Неожиданный код возврата ( ошибка в step)
		};
		struct MfstDiagnosis {
			short lenta_position, nrule, nrule_chain;
			RC_STEP rc_step;
			MfstDiagnosis();
			MfstDiagnosis(short plenta_position, RC_STEP prc_step, short pnrule, 
short pnrule_chain);
		}diagnosis[MFST_DIAGN_NUMBER];
		std::vector<GRBALPHABET> tape;
		short lenta_position, nrule, nrulechain, lenta_size;
		GRB::Greibach grebach;
		LT::LexTable lex;
		MFSTSTSTACK st;
		use_container<std::stack<MfstState>> storestate;
		Mfst();
		Mfst(LT::LexTable plex, GRB::Greibach pgrebach);
		std::string getCSt();
		std::string getCLenta(short pos, short n = 25);
		std::string getDiagnosis(short n);
		bool savestate(const Log::LOG& log);
		bool reststate(const Log::LOG& log);
		bool push_chain(GRB::Rule::Chain chain);
		RC_STEP step(const Log::LOG& log);
		bool start(const Log::LOG& log);
		bool savediagnois(RC_STEP pprc_step);
		void printrules(const Log::LOG& log);
		static void clearGreibach(Mfst&);
		struct Deducation {
			short size;
			std::vector<short> nrules, nrulechains;
			Deducation() : size(0), nrules(0), nrulechains(0) { };
		}deducation;
		bool savededucation();};
```

Листинг В.1 – Структура магазинного конечного автомата

ПРИЛОЖЕНИЕ Г

```
0   : S->tfi(F){NrE;}S              tfi(ti,ti){dti;i=i+i;ri;}     S$
1   : SAVESTATE:          1
1   :                               tfi(ti,ti){dti;i=i+i;ri;}     tfi(F){NrE;}S$
2   :                               fi(ti,ti){dti;i=i+i;ri;}t     fi(F){NrE;}S$
3   :                               i(ti,ti){dti;i=i+i;ri;}tf     i(F){NrE;}S$
4   :                               (ti,ti){dti;i=i+i;ri;}tfi     (F){NrE;}S$
5   :                               ti,ti){dti;i=i+i;ri;}tfi(     F){NrE;}S$
6   : F->ti                         ti,ti){dti;i=i+i;ri;}tfi(     F){NrE;}S$
7   : SAVESTATE:          2
7   :                               ti,ti){dti;i=i+i;ri;}tfi(     ti){NrE;}S$
8   :                               i,ti){dti;i=i+i;ri;}tfi(t     i){NrE;}S$
9   :                               ,ti){dti;i=i+i;ri;}tfi(ti     ){NrE;}S$
```

Рисунок Г.1 – Работа синтаксического анализатора (начало разбора)

```
4748: TS_NOK/NS_NORULECHAIN
4748: RESSTATE
4748:                                   pl;]rl;}                      N]rE;}$
4749: N->pl;                            pl;]rl;}                      N]rE;}$
4750: SAVESTATE:          134
4750:                                   pl;]rl;}                      pl;]rE;}$
4751:                                   l;]rl;}                       l;]rE;}$
4752:                                   ;]rl;}                        ;]rE;}$
4753:                                   ]rl;}                         ]rE;}$
4754:                                   rl;}                          rE;}$
4755:                                   l;}                           E;}$
4756: E->l                             l;}                           E;}$
4757: SAVESTATE:          135
4757:                                   l;}                           l;}$
4758:                                   ;}                            ;}$
4759:                                   }                             }$
4760:                                                                 $
4761: LENTA_END
4762: ------>LENTA_END
```

Рисунок Г.2 – Работа синтаксического анализатора (конец разбора)

```
0   : S->tfi(F){NrE;}S
4   : F->ti,F
7   : F->ti
11  : N->dti;N
15  : N->i=E;
17  : E->iM
18  : M->+E
19  : E->i
22  : E->i
25  : S->tfi(F){NrE;}S
29  : F->ti,F
32  : F->ti
36  : N->dti;N
40  : N->i=E;
42  : E->iM
```

```
43  : M->+E
44  : E->(E)M
45  : E->i(W)M
47  : W->i,W
49  : W->i
51  : M->+E
52  : E->i(W)M
54  : W->i,W
56  : W->i
58  : M->+E
59  : E->(E)
60  : E->iM
61  : M->-E
62  : E->i
65  : M->-E
66  : E->iM
67  : M->*E
68  : E->i
71  : E->i
74  : S->m{NrE;}
76  : N->dtfi(F);N
81  : F->ti,F
84  : F->ti
88  : N->dtfi(F);N
93  : F->ti
97  : N->dti;N
101 : N->dti;N
105 : N->dti;N
109 : N->dti;N
113 : N->dti;N
117 : N->dti;N
121 : N->dti;N
125 : N->i=E;N
127 : E->l
129 : N->i=E;N
131 : E->l
133 : N->i=E;N
135 : E->l
137 : N->i=E;N
139 : E->l
141 : N->pl;N
144 : N->i=E;N
146 : E->i(W)
148 : W->i,W
150 : W->l
153 : N->pi;N
156 : N->pl;N
159 : N->i=E;N
161 : E->iM
162 : M->+E
163 : E->i
165 : N->pi;N
```

```
168 : N->pl;N
171 : N->i=E;N
173 : E->iM
174 : M->-E
175 : E->i
177 : N->pi;N
180 : N->pl;N
183 : N->i=E;N
185 : E->i(W)
187 : W->i
190 : N->pl;N
193 : N->pi;N
196 : N->pl;N
199 : N->pi;N
202 : N->pl;N
205 : N->i=E;N
207 : E->iM
208 : M->*E
209 : E->i
211 : N->pi;N
214 : N->pl;N
217 : N->i=E;N
219 : E->l
221 : N->i=E;N
223 : E->l
225 : N->i=E;N
227 : E->iM
228 : M->+E
229 : E->i
231 : N->pi;N
234 : N->pl;N
237 : N->i=E;N
239 : E->iM
240 : M->&E
241 : E->i
243 : N->pi;N
246 : N->pl;N
249 : N->i=E;N
251 : E->iM
252 : M->|E
253 : E->i
255 : N->pi;N
258 : N->pl;N
261 : N->i=E;N
263 : E->iM
264 : M->~E
265 : E->l
267 : N->pi;N
270 : N->pl;N
273 : N->dti;N
277 : N->i=E;N
279 : E->i(W)
```

```
281 : W->l,W
283 : W->l
286 : N->pi;N
289 : N->dti;N
293 : N->i=E;N
295 : E->l
297 : N->pl;N
300 : N->pi;N
303 : N->pl;N
306 : N->dti;N
310 : N->i=E;N
312 : E->l
314 : N->e(C)[N]
316 : C->i
319 : N->pl;
324 : E->l
```

Рисунок Г.3 – Результат работы синтаксического анализатора

ПРИЛОЖЕНИЕ Д

```
bool PN::find_elem(std::stack<char> stack, size_t size, char elem) {
	for (size_t i = 0; i < size; i++)
		if (stack.top() == elem)
			return true;
		else
			stack.pop();
	return false;
}

int PN::get_priority(char lexem) {
	std::vector<std::pair<int, char>> priority = { {0, LEX_LEFTHESIS}, {0,
LEX_RIGHTHESIS},

	{1,LEX_COMMA},

	{2,LEX_PLUS}, {2, LEX_MINUS},

	{3,LEX_STAR}, {3, LEX_OR}, {3, LEX_AND}, {3, LEX_INVERSE}};
	for (size_t i = 0; i < priority.size(); i++)
		if (lexem == priority[i].second)
			return priority[i].first;
	return 0;
}

void PN::fix_lextable(LT::LexTable& lextable, const std::string& str, size_t
length, size_t pos, const std::vector<int>& ids) {
	for (size_t i = 0, q = 0; i < str.size(); i++) {
		lextable.table[pos + i].lexema = str[i];
		if (lextable.table[pos + i].lexema == LEX_ID || lextable.table[pos
+ i].lexema == LEX_LITERAL) {
			lextable.table[pos + i].idxTI = ids[q];
			q++;
		}
		else
			lextable.table[pos + i].idxTI = LT_TI_NULLIDX;
	}
	int temp = str.size() + pos;
	for (size_t i = 0; i < length - str.size(); i++) {
		lextable.table[temp + i].idxTI = LT_TI_NULLIDX;
		lextable.table[temp + i].lexema = '!';
		lextable.table[temp + i].sn = -1;
	}
}

void PN::PolishNotation(LT::LexTable& lextable, IT::IdTable& idtable) {
	for (int i = 0; i < lextable.size; i++)
		if (lextable.table[i].lexema == LEX_EQUAL)
			if (!Convertation(i + 1, lextable, idtable))
				throw ERROR_THROW(130);
```

```
}

bool PN::Convertation(int lextable_pos, LT::LexTable& lextable, IT::IdTable&
idtable)
{
	container<std::stack<char>> stack;
	std::string PolishString;
	std::vector<char> operators = { LEX_MINUS, LEX_PLUS, LEX_STAR, LEX_AND,
LEX_OR, LEX_INVERSE };
	std::vector<int> ids;
	int operators_count = 0, operands_count = 0, iterator = 0, right_counter
= 0, left_counter = 0, params_counter = 0;

	for (int i = lextable_pos; i < lextable.size; i++, iterator++) {
		char lexem = lextable.table[i].lexema;
		size_t stack_size = stack.size();
		if (idtable.table[lextable.table[i].idxTI].idtype == IT::IDTYPE::F)
{
			stack.push('@');
			operands_count--;
		}
		if (std::find(operators.begin(), operators.end(), lexem) !=
operators.end()) {
			if (!stack.empty() && stack.top() != LEX_LEFTHESIS) {
				while (!stack.empty() && PN::get_priority(lexem) <=
PN::get_priority(stack.top())) {
					PolishString += stack.top();
					stack.pop();
				}
			}
			stack.push(lexem);
			operators_count++;
		}
		else if (lexem == LEX_COMMA) {
			while (!stack.empty()) {
				if (stack.top() == LEX_LEFTHESIS)
					break;
				PolishString += stack.top();
				stack.pop();
			}
			operands_count--;
		}
		else if (lexem == LEX_LEFTHESIS) {
			left_counter++;
			stack.push(lexem);
		}
		else if (lexem == LEX_RIGHTHESIS) {
			right_counter++;
			if (!PN::find_elem(stack, stack_size, LEX_LEFTHESIS))
				return false;
			while (stack.top() != LEX_LEFTHESIS) {
				PolishString += stack.top();
```

```
                    stack.pop();
                }
                stack.pop();
                if (!stack.empty() && stack.top() == '@') {
                    PolishString += stack.top() + To_string(params_counter -
1);
                    params_counter = 0;
                    stack.pop();
                }
            }
            else if (lexem == LEX_SEMICOLON) {
                if (operators_count != 0 && operands_count != 0)
                    if ((!stack.empty() && (stack.top() == LEX_RIGHTHESIS ||
stack.top() == LEX_LEFTHESIS))
                        || right_counter != left_counter || operands_count
- operators_count != 1)
                        return false;
                while (!stack.empty()) {
                    PolishString += stack.top();
                    stack.pop();
                }
                PN::fix_lextable(lextable, PolishString, iterator,
lextable_pos, ids);
                break;
            }
            else if (lexem == LEX_ID || lexem == LEX_LITERAL) {
                if (std::find(stack.c.begin(), stack.c.begin(), '@') !=
stack.c.end())
                    params_counter++;
                PolishString += lexem;
                if (lextable.table[i].idxTI != LT_TI_NULLIDX)
                    ids.push_back(lextable.table[i].idxTI);
                operands_count++;
            }
        }
        return true;
}
```

Листинг Д.1 – Алгоритма преобразования выражений к польской записи

ПРИЛОЖЕНИЕ Е

```
.586
.model flat, stdcall
includelib libucrt.lib
includelib kernel32.lib
includelib ../Debug/GED_Lib.lib
ExitProcess PROTO : DWORD
_ConsoleWriteInt PROTO : SDWORD
_ConsoleWriteHex PROTO : SDWORD
iabs PROTO : SDWORD
step PROTO : SDWORD, : SDWORD
output PROTO : SDWORD


.stack 4096
.const
	l0 SDWORD 4
	l1 SDWORD 6
	l2 SDWORD 1
	l3 SDWORD 0
	l4 BYTE '... func step ...', 0
	l5 SDWORD 2
	l6 BYTE '... summ 4 and 6 ...', 0
	l7 BYTE '... sub 4 and 6 ...', 0
	l8 BYTE '... func iabs ...', 0
	l9 BYTE '.. begin ..', 0
	l10 BYTE '.. later ..', 0
	l11 BYTE '... mul 4 and 6 ...', 0
	l12 BYTE '... summ hex ...', 0
	l13 SDWORD 10
	l14 SDWORD 31
	l15 BYTE '... AND 1 and 0 ...', 0
	l16 BYTE '... OR 1 and 0 ...', 0
	l17 BYTE '... NOT a ...', 0
	l18 BYTE ' ... func expression ...', 0
	l19 BYTE 'W', 0
	l20 BYTE '... Out symb ...', 0
	l21 BYTE '... If and else ...', 0
	l22 SDWORD 3
	l23 BYTE '10', 0
.data
	expresssummab		SDWORD 0
	expressionsm		SDWORD 0
	maina		SDWORD 0
	mainb		SDWORD 0
	mainc		SDWORD 0
	maind		SDWORD 0
	mainz		SDWORD 0
	mainone		SDWORD 0
	mainzero		SDWORD 0
	mainresult		SDWORD 0
```

```
      mainsa                SDWORD 0
      mainy           SDWORD 0

.code
express PROC b: SDWORD, a: SDWORD
      push       a
      push       b
      ;\/(ADD)+\/
      pop        eax
      pop        ebx
      ADD        eax, ebx
      push       eax
      ;/\(ADD)+/\
      pop              expresssummab

      mov        eax, expresssummab
      ret
express ENDP

expression PROC b: SDWORD, a: SDWORD
      push       a
      push       a
      push       b
      call       express
      push       eax
      push       a
      push       b
      call       express
      push       eax
      ;\/(ADD)+\/
      pop        eax
      pop        ebx
      ADD        eax, ebx
      push       eax
      ;/\(ADD)+/\
      push       b
      push       a
      ;\/(SUB)-\/
      pop        ebx
      pop        eax
      SUB        eax, ebx
      push       eax
      ;/\SUB(-)/\
      ;\/(ADD)+\/
      pop        eax
      pop        ebx
      ADD        eax, ebx
      push       eax
      ;/\(ADD)+/\
      ;\/(ADD)+\/
      pop        eax
      pop        ebx
```

```
      ADD         eax, ebx
      push        eax
      ;/\(ADD)+/\
      push        b
      push        a
      ;\/(MUL)*\/
      pop         eax
      pop         ebx
      MUL         ebx
      push        eax
      ;/\(MUL)*/\
      ;\/(SUB)-\/
      pop         ebx
      pop         eax
      SUB         eax, ebx
      push        eax
      ;/\SUB(-)/\
      pop               expressionsm

      mov         eax, expressionsm
      ret
expression ENDP

main PROC
      push        l0
      pop               maina

      push        l1
      pop               mainb

      push        l2
      pop               mainone

      push        l3
      pop               mainzero

      push        offset l4
      call        output

      push        maina
      push        l5
      call        step
      push        eax
      pop               mainc

      push        mainc
      call        _ConsoleWriteInt
      push        offset l6
      call        output

      push        maina
      push        mainb
```

```
    ;\/(ADD)+\/
    pop         eax
    pop         ebx
    ADD         eax, ebx
    push        eax
    ;/\(ADD)+/\
    pop              mainc

    push        mainc
    call        _ConsoleWriteInt
    push        offset l7
    call        output

    push        mainb
    push        maina
    ;\/(SUB)-\/
    pop         ebx
    pop         eax
    SUB         eax, ebx
    push        eax
    ;/\SUB(-)/\
    pop              mainc

    push        mainc
    call        _ConsoleWriteInt
    push        offset l8
    call        output

    push        mainc
    call        iabs
    push        eax
    pop              maind

    push        offset l9
    call        output

    push        mainc
    call        _ConsoleWriteInt
    push        offset l10
    call        output

    push        maind
    call        _ConsoleWriteInt
    push        offset l11
    call        output

    push        mainb
    push        maina
    ;\/(MUL)*\/
    pop         eax
    pop         ebx
    MUL         ebx
```

```
    push       eax
    ;/\(MUL)*/\
    pop             mainc

    push       mainc
    call       _ConsoleWriteInt
    push       offset l12
    call       output

    push       l13
    pop             maina

    push       l14
    pop             mainb

    push       mainb
    push       maina
    ;\/(ADD)+\/
    pop        eax
    pop        ebx
    ADD        eax, ebx
    push       eax
    ;/\(ADD)+/\
    pop             mainc

    push       mainc
    call       _ConsoleWriteInt
    push       offset l15
    call       output

    push       mainzero
    push       mainone
    ;\/AND(&)\/
    pop        eax
    pop        edx
    AND        eax, edx
    push       eax
    ;/\AND(&)/\
    pop             mainz

    push       mainz
    call       _ConsoleWriteInt
    push       offset l16
    call       output

    push       mainzero
    push       mainone
    ;\/(OR)|\/
    pop        eax
    pop        edx
    OR         eax, edx
    push       eax
```

```
      ;/\(OR)|/\
      pop             mainz

      push       mainz
      call       _ConsoleWriteInt
      push       offset l17
      call       output

      push       maina
      push       l0
      ;\/(INVERSE(NOT)):\/
      pop        eax
      pop        eax
      NOT             eax
      push       eax
      ;/\(INVERSE(NOT)):/\
      pop             mainz

      push       mainz
      call       _ConsoleWriteInt
      push       offset l18
      call       output

      push       l5
      push       l1
      call       expression
      push       eax
      pop             mainresult

      push       mainresult
      call       _ConsoleWriteInt
      push       offset l19
      pop             mainsa

      push       offset l20
      call       output

      push       mainsa
      call       output

      push       offset l21
      call       output

      push       l22
      pop             mainy

while_start0:
      push mainy
      pop eax
      test eax, eax
      jz while_start0
      pop ebx
```

```
      pop eax
      loop while_start0
      push       l3
      call       ExitProcess
main ENDP
end main
```

Листинг Е.1 – Результат генерации ассемблерного кода

ПРИЛОЖЕНИЕ Ж

```
#define IN_CODE_TABLE {\
      IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::I, IN::T,
IN::F, IN::F, IN::I, IN::F, IN::F, \
      IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, \
      IN::S, IN::T, IN::F, IN::F, IN::O, IN::O, IN::O, IN::Q, IN::O, IN::O, IN::O,
IN::O, IN::O, IN::O, IN::O, IN::O, \
      IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::O, IN::T, IN::O, IN::T, IN::T, \
      IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, \
      IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::O, IN::F, IN::O, IN::F, IN::F, \
      IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, \
      IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::O, IN::O, IN::O, IN::O, IN::F, \


                  \
      IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, \
      IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, \
      IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, \
      IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, \
      IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, \
      IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, \
      IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, \
      IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, \
                                        }
```

Листинг Ж.1 – Таблица допустимых символов