Содержание

Введение

Целью курсового проекта является разработка компилятора для своего языка программирования: GPA-2022. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над числами.

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.

Компилятор GPA-2022 состоит из следующих составных частей:

– лексический анализатор;

– синтаксический анализатор;

– семантический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода на язык ассемблер;

– тестирование транслятора.

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

Глава 1 Спецификация языка программирования

1.1 Характеристика языка программирования

Язык программирования GPA-2022 – процедурный, компилируемый, строго типизированный язык.

1.2 Определение алфавита языка программирования

В основе алфавита GPA-2022 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

Разрешены латинские символы, символы-сепараторы, символы операций и другие.

Алфавит языка GPA-2022 состоит из следующих множеств символов:

– латинские символы верхнего и нижнего регистра: {A, B, C, …, Z, a, b, c, …, z};

– цифры: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

– знаки пунктуации языка: {(), {}, [], ;, =}.

1.3 Применяемые сепараторы

Сепараторы необходимы для разделения операций языка. Сепараторы, используемые в языке программирования GPA-2022, приведены в таблице 1.1.

Таблица 1.1 - Применяемые сепараторы

Сепаратор	Назначение сепаратора
;	Разделитель инструкций
{ } [ ]	Программный блок
( )	Параметры, приоритетность операций
пробел	Служит для разделения программных конструкций. Допускается везде, кроме идентификаторов и ключевых слов
,	Разделитель параметров в функции

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 GPA- 2022 есть 3 типа данных: целочисленный, строковый и логический. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 - Фундаментальные типы данных

Тип данных	Описание
int	Фундаментальный тип данных. Автоматическая инициализация 0; Возможные операции: – Арифметические(+, -, =); – Побитовые(~, |, &); – Equal.
bool	Фундаментальный тип данных. Занимает 1 байт, значение 0 (false) или любое отличное от 0 (true), автоматическая инициализация 0 (false).
string	Фундаментальный тип данных. Предусмотрен для объявления строк. (1 символ – 1 байт). Автоматическая инициализация строкой нулевой длины.

1.6 Преобразование типов данных

В языке программирования GPA-2022 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.

Имя идентификатора составляется по следующим образом:

– состоит из символов латинского алфавита [a..z];

– допускается использование цифр(идентификатор не может начинаться с цифры).

1.8 Литералы

В языке существует 3 типа литералов: целого, булевского и строкового типов. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3 - Описание литералов

Тип литерала	Описание литерала
Строковые	Состоит из символов, заключенных в (одинарные кавычки), инициализируются пустой строкой, строковые переменные.
Логические	Распознаются при помощи ключевых слов “true” и “false”, соответственно значения от 0 (false) до 1 (true)
Целые	Целочисленные литералы, инициализируются 0. Могут быть представлены как в десятичном, так и в восьмеричном(первый символ: 0)представлении.

1.9 Объявления данных

В языке программирования GPA-2022 переменная должны быть объявлена до ее использования. Областью видимости переменной является блок функции, в которой она определена. Вне блока функции определении функции запрещено. Не допустимо объявление глобальных переменных. Конструкция для объявления переменных:

let <тип данных> <идентификатор>;

<идентификатор>=<литерал>|<идентификатор>;

1.10 Инициализация данных

При объявлении переменной допускается инициализация данных. Объектами-инициализаторами могут быть идентификаторы, литералы, выражения и вызов функции. Предусмотрены значения по умолчанию, если переменные не инициализированы: 0 – для целочисленных типов данных, пустая строка (строка размером 0) – для строкового типа данных, 0 (false) – для логического типа данных.

1.11 Инструкции языка

В языке программирования GPA-2022 применяются инструкции, представленные в таблице 1.4.

Таблица 1.4 - Инструкции языка

Инструкция языка	Синтаксис
Вызов функций	<идентификатор функции>(<идентификатор / литерал>,…)
Возврат из функции	return <идентификатор> | <литерал> | <выражение>;
Объявление переменной	let <тип данных> <идентификатор> ;
Присваивание	<идентификатор>=<литерал>|<идентификатор>;
Вывод данных	write <идентификатор> | <литерал>;
Объявление внешней функции	<тип данных> Func <идентификатор> (<тип данных> <идентификатор>, …) {…}

1.12 Операции языка

Язык программирования GPA-2022 может выполнять арифметические операции, представленные в таблице 1.5.

Таблица 1.5 – Приоритетности операций языка программирования GPA-2022

Операция	Приоритетность
(	1
~	5
|	2
,	-2
&	3
+/-	4
)	-1
;	-3

Операции языка применимы исключительно к целочисленному типу данных. Для строкового и логического типа операции языка не предусмотрены.

1.13 Выражения и их вычисления

Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих арифметических операций. Выражение может содержать вызов функции.

Булевские выражения могут содержать либо булевскую переменную, либо сравнение двух целочисленных значений. Вызов функции запрещён.

1.14 Конструкции языка

Ключевые программные конструкции языка программирования представлены в таблице 1.6.

Таблица 1.6 - Программные конструкции языка

Конструкция	Представление в языке

Главная функция	start
	{
	…
	};

Функция	<тип данных> Func <идентификатор> (<идентификатор>, …)
	{
	…
	return <идентификатор\литерал>;
	};

Условный оператор	if (<булевское выражение>) [<инструкции языка>]

1.15 Область видимости идентификаторов

Все идентификаторы обязаны быть объявленными внутри функции. Вне функции объявление идентификаторов недопустимы. Глобальных переменных нет, только локальные. Параметры видны только внутри функции, в которой объявлены. Объявление пользовательских областей видимости не предусмотрено.

1.16 Семантические проверки

Перечень семантических проверок, предусмотренных языком, приведен в таблице 1.7.

Таблица 1.7 - Перечень семантических проверок

Номер	Правило
1	Идентификаторы функций не должны повторяться

Продолжение таблицы 1.7

Номер	Правило
2	Операнды в операторах ветвления и выхода из функции должны быть целочисленного типа
3	Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении
4	Идентификатор должен быть объявлен до его использования.
5	Операнды в арифметическом выражении не могут быть различных типов
6	Тип возвращаемого функцией значения должен совпадать с типом функции

1.17 Распределение оперативной памяти на этапе выполнения

Все переменные хранятся в куче.

1.18 Стандартная библиотека и её состав

В языке GPA-2022 присутствует стандартная библиотека, которая автоматически подключается при трансляции исходного кода. Содержимое библиотеки и описание функций представлено в таблице 1.8.

Таблица 1.8 - Стандартная библиотека языка GPA-2022

Функция	Описание
string strcat (string a, string b);	Строковая функция. Принимает в качестве параметров 2 строки. Копирует вторую строку к концу первой строки.
string strcopy (string a, string b);	Строковая функция. Принимает в качестве параметров 2 строки. Копирует содержимое второй строки в первую.
bool strcmp (string a, string b);	Логическая функция. Принимает в качестве параметров 2 строки. Если строки равны возвращает true, иначе false.
int strlen (string a)	Функция принимает в качестве параметра строку и возвращает длину.

1.19 Ввод и вывод данных

Ввод данных не поддерживается языком программирования GPA-2022.
Вывод данных осуществляется с помощью ключевого слова write.

1.20 Точка входа

В языке GPA-2022 точкой входа является функция start. В программе может быть только одна точка входа.

1.21 Препроцессор

В языке GPA-2022 препроцессоры не предусмотрены.

1.22 Соглашения о вызовах

При генерации кода используется соглашение __stdcall, в котором все параметры передаются в стек справа налево. Освобождением памяти занимается вызываемая подпрограмма, которая очищает стек.

1.23 Объектный код

Исходный код языка транслируется в язык ассемблера.

1.24 Классификация сообщений транслятора

Транслятор генерирует сообщения об ошибках пользователю. Классификация ошибок языка GPA-2022 представлена в таблице 1.9.

Таблица 1.9 – Классификация сообщений транслятора

Интервал	Описание ошибок
0-1	Системные ошибки
100-112	Ошибки параметров
113-119	Ошибки лексического анализа
600-607	Ошибки синтаксического анализа
120-129	Ошибки семантического анализа

1.25 Контрольный пример

Контрольный пример на языке GPA-2022 в приложении А.

Глава 2 Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор преобразует программу, написанную на языке GPA-2022 в программу на языке ассемблера. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.

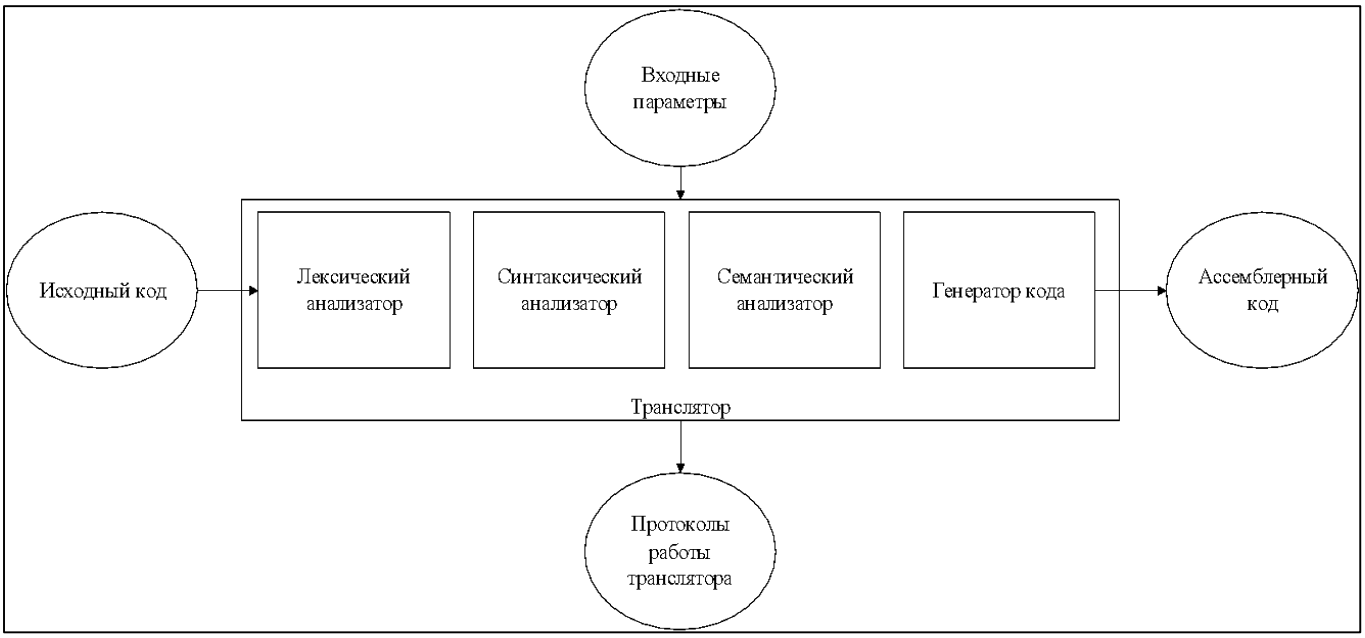

Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка и формирование таблицы лексем и таблицы идентификаторов. Подробнее описан в третьем разделе.

Семантический анализ в свою очередь является проверкой исходной программы на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики. Подробное описание представлено в пятом разделе.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода. Входным параметром для синтаксического анализа является таблица лексем. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора. Подробнее рассмотрен в четвертом разделе.

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 GPA-2022, прошедший все предыдущие этапы, в код на языке Ассемблера. Более полно описано в седьмом разделе.

2.2 Перечень входных параметров транслятора

В таблице 2.1 представлены входные параметры, которые могут использоваться для управления работой транслятора.

Таблица 2.1 – Входные параметры транслятора

Параметр	Назначение	Тип
-in:	Указывает на файл с исходным кодом. Исходный код содержится в файле с расширением *.txt	Обязательный
-out:	Указывает имя выходного файла. Если не указан явно, то имя протокола формируется конкатенацией имени файла исходного кода и постфикса «.asm»	Не обязательный
-log:	Указывает имя протокола. Если не указан явно, то имя протокола формируется конкатенацией имени файла исходного кода и постфикса «.log»	Не обязательный

2.3 Протоколы, формируемые транслятором

Таблица с перечнем протоколов, формируемых транслятором языка GPA-2022 и их назначением представлена в таблице 2.2

Таблица 2.2 – Протоколы, формируемые транслятором GPA-2022

Название протокола	Расширение	Описание
“Имя файла с исходным кодом”	.txt.log	В файле находится информация о входных параметрах, количестве символов исходного кода
IT	.txt.it	В файле находится таблица идентификаторов
LT	.txt.lt	В файле находится таблица лексем

Глава 3 Разработка лексического анализатора

3.1 Структура лексического анализатора

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Структура лексического анализатора представлена на рисунке 3.1

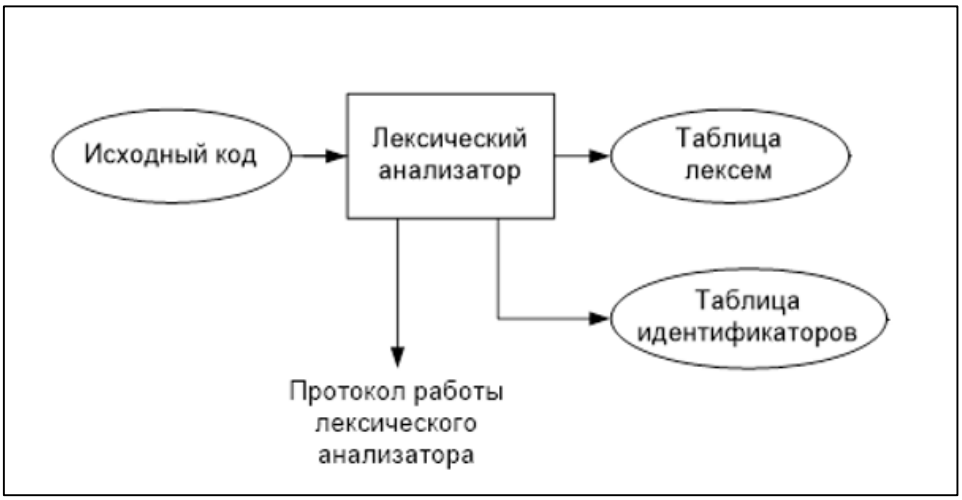

Рисунок 3.1 Структура лексического анализатора GPA-2022

Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке GPA-2022. На выходе формируется таблица лексем и таблица идентификаторов.

3.2 Контроль входных символов

Таблица для контроля входных символов представлена на рисунке 3.2

```
#define IN_CODE_TABLE {\
/*15*/      IN::P, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::P, IN::P, IN::T, IN::T, IN::T, IN::T, IN::T, \
/*31*/      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, \
/*47*/      IN::P, IN::T, IN::T, IN::T, IN::T, IN::S, IN::T, IN::Q, IN::S, IN::S, IN::S, IN::S, IN::S, IN::S, IN::F, IN::S, \
/*63*/      IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::S, IN::T, IN::S, IN::T, IN::T, \
/*79*/      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, \
/*95*/      IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::S, IN::T, IN::S, IN::F, IN::T, \
/*111*/     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, \
/*127*/     IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::S, IN::F, IN::S, IN::T, IN::T, \
            \
/*143*/     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, \
/*159*/     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, \
/*175*/     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, \
/*191*/     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, \
/*207*/     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, \
/*223*/     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, \
/*239*/     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, \
/*255*/     IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \
}
```

Рисунок 3.2. Таблица контроля входных символов

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 – сепаратор, P – пробелы, табуляция и переход на новую строку, Q – одинарная кавычка.

3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы. Избыточные символы удаляются на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом программы.
2. Встреча пробела или знака табуляции является своего рода встречей символа-сепаратора.
3. В отличие от других символов-сепараторов не записываем в слово эти символы, только этот символ не является частью строкового литерала.

3.4 Перечень ключевых слов

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

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

Тип цепочки	Цепочка	Лексема
Ключевые слова	let	v
	int	t
	string	t
	bool	t
	Func	f
	return	r
	write	w
	start	s
	if	?
	false	t
	true	t
	equal	e
Иное	Идентификатор	i
	Литерал	l
Сепараторы	;	;
	,	,
	{	{
	}	}
	(	(
	)	)

Продолжение таблицы 3.1

Тип цепочки	Цепочка	Лексема
Сепараторы	[	[
	]	]
Операторы	+	o
	-	o
	&	o
	|	o
	~	o
	=	=

Пример реализации таблицы лексем представлен в приложении А.

Также в приложении А находятся конечные автоматы, соответствующие лексемам языка GPA-2022.

3.5 Основные структуры данных

Основные структуры таблиц лексем и идентификаторов данных языка GPA- 2022, используемых для хранения, представлены в приложении А. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде, для арифметических операторов хранится их значение. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение.

3.6 Принцип обработки ошибок

При возникновении ошибки работа транслятора прекращается и диагностическое сообщение будет выведено в log файл.

3.7 Структура и перечень сообщений лексического анализатора

Перечень сообщений лексического анализатора представлен в таблице 3.2.

Таблица 3.2 – Перечень сообщений лексического анализатора

Код	Сообщение
113	Элемент не распознан
114	Ошибка при создании файла IT
115	Ошибка при создании файла LT
116	Ошибка при создании лексической таблицы (превышен максимальный размер)
117	Ошибка при создании таблицы идентификаторов (превышен максимальный размер)
118	Ошибка при получении строки лексической таблицы (нет элемента)
119	Ошибка при получении строки таблицы идентификаторов (нет элемента)

3.8 Параметры лексического анализатора и режимы его работы

Входным параметром лексического анализа является структура, полученная на этапе проверки исходного кода на допустимость символов.

3.9 Алгоритм лексического анализа

Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором. Цель лексического анализа — выделение и классификация лексем в тексте исходной программы. Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. 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. И ещё она производит проверку на использование переменной перед объявлением и проверку на повторное объявление переменных.

3.10 Контрольный пример

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

Глава 4 Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

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

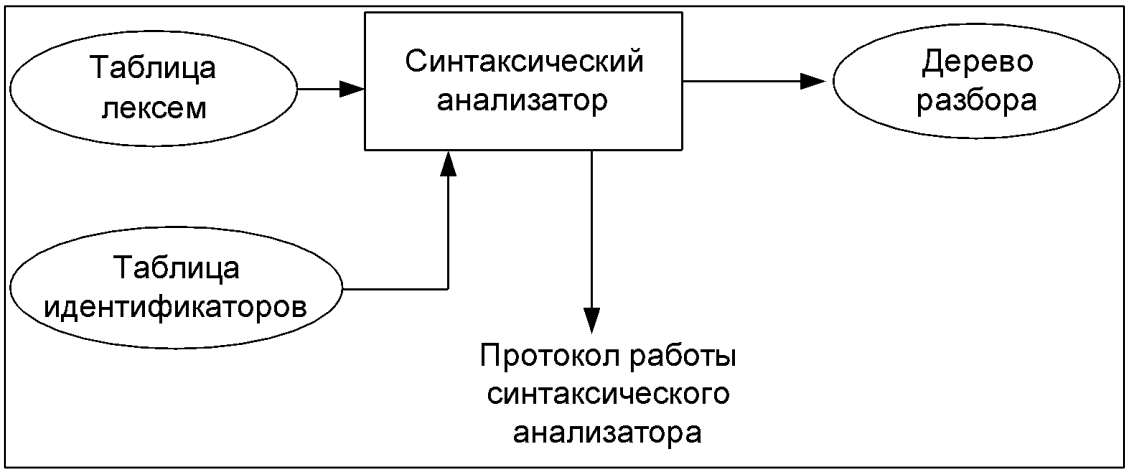

Рисунок 4.1 – Структура синтаксического анализатора

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 GPA-2022 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.

Правила языка GPA-2022 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 GPA-2022

Нетерминал	Цепочки правил	Описание
S	tfi(F){NrR;};S tfi(F){rR;};S s{N};	Порождает правила, описывающее общую структуру программы

N	vti;
	i=E;
	vti;N
	i=E;N
	w(R);
	w(R);N
	?(I)[N];N
	?(I)[N];
		Порождает правила, описывающие инструкции языка

E	i l (E) i() i(W) iM lM (E)M i(W)M	Порождает правила, описывающие выражения
M	oE voM	Порождает правила, описывающие арифметические действия
F	ti ti,F	Порождает правила, описывающие параметры шаблона функции
W	i l i,W l,W	Порождает правила, описывающие параметры функции
R	l i	Порождает правила, описывающие возвращаемый тип и операнды вывода в консоль
I	ioi lol loi iol i l	Порождает правила, описывающие условную конструкцию

4.3 Построение конечного магазинного автомата

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

Таблица 4.2 – Описание компонентов магазинного автомата

Компонента	Определение	Описание
Q	Множество состояний автомата	Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата
V	Алфавит входных символов	Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1.
Z	Алфавит специальных магазинных символов	Алфавит магазинных символов содержит стартовый символ и маркер дна стека
δ	Функция переходов автомата	Функция представляет из себя множество правил грамматики, описанных в таблице 4.1.
q_0	Начальное состояние автомата	Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ S)
z_0	Начальное состояние магазина автомата	Символ маркера дна стека ($)
F	Множество конечных состояний	Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты

4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающей правила языка GPA-2022. Данные структуры представлены в приложении В.

4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата следующий:

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

4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен в таблице 4.3.

Таблица 4.3 – Перечень сообщений синтаксического анализатора

Код	Сообщение
600	Неверная структура программы
601	Ошибочный оператор
602	Ошибка в выражениях
603	Ошибка в операторах выражения
604	Ошибка в параметрах функции
605	Ошибка в параметрах функции
606	Значением данного оператора может быть только переменная или литерал
607	Ошибка в условии условного оператора

4.7 Параметры синтаксического анализатора и режимы его работы

Входным параметром синтаксического анализатора является таблица лексем, полученная на этапе лексического анализа, а также правила контекстно-свободной грамматики в форме Грейбах.

Выходными параметрами являются трассировка прохода таблицы лексем (при наличии разрешающего ключа) и правила разбора, которые выводятся в консоль.

4.8 Принцип обработки ошибок

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.

2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
3. Все ошибки записываются в общую структуру ошибок.
4. В случае нахождения ошибки, после всей процедуры трассировки в протокол будет выведено диагностическое сообщение.

4.9 Контрольный пример

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

Глава 5 Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализ происходит после выполнения фазы синтаксического анализа и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях и заполнении таблицы идентификаторов.

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

Сообщения, формируемые семантическим анализатором, представлены в таблице 4.3.

Таблица 4.3 – Перечень сообщений семантического анализатора

Код	Сообщение
120	Невозможно применить оператор к данному типу операнда
121	Тип возвращаемого значения не соответствует типу функции
122	Использована необъявленная переменная
123	Повторное объявление имени
124	Функция должна возвращать значение
125	Ошибка в условии условного оператора
126	Длина строкового литерала превышает допустимое значение
127	Ошибка в условии условного оператора
128	Несовпадение фактических и формальных параметров функции

5.4 Принцип обработки ошибок

Принцип обработки ошибок идентичен принципу обработки ошибок на этапе лексического анализа (раздел 3.6).

5.5 Контрольный пример

Демонстрации ошибок, диагностируемых семантическим анализатором на разных этапах трансляции приведена в разделе 8.2

Глава 6 Преобразование выражений

6.1 Выражения, допускаемые языком

В языке GPA-2022 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, &, ~, | и (), и вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке GPA-2022

Приоритет	Операция
-3	;
-2	,
-1	)
1	(
2	|
3	&
4	+
4	-
5	~

6.2 Польская запись и принцип ее построения

Выражения в языке GPA-2022 преобразовываются к обратной польской записи.

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

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

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– результирующая строка: польская запись;

– исходная строка просматривается слева направо;

– операнды переносятся в результирующую строку в порядке их следования;

– операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;

– операция выталкивает все операции с большим или равным приоритетом в результирующую строку;

– запятая не помещается в стек, если в стеке операции, то все выбираются в строку;

– отрывающая скобка помещается в стек;

– закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;

– закрывающая скобка, если поднят флаг функции, выталкивает все до открывающей скобки в обратном порядке и добавляет идентификатор функции в конце;

– по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

Исходная строка	Результирующая строка	Стек
(a+b) - f(i)		
+b) - f(i)	a	(
b) - f(i)	a	(+
) - f(i)	ab	(+
- f(i)	ab+	
f(i)	ab+c*	-
	ab+i@-	

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

Заключительным этапом трансляции языка GPA-2022 является генерация кода. Генератор принимает на вход таблицу лексем, таблицу идентификаторов и дерево разбора. На выходе получается файл с исходным кодом на ассемблере, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода представлена на рисунке 7.1.

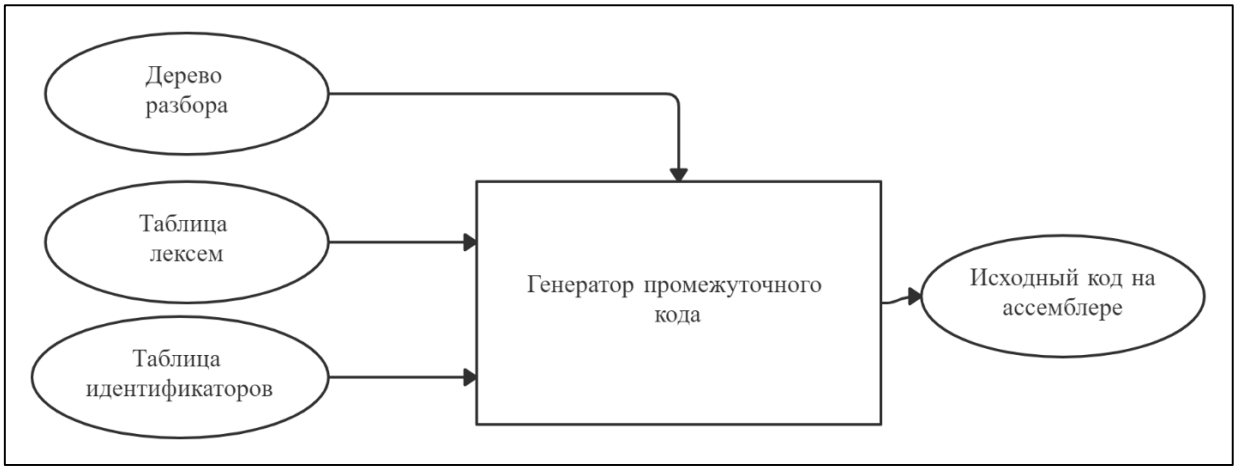

Рисунок 7.1 – Структура генератора кода

7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка GPA-2022 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке GPA-2022 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка GPA-2022 и языка Ассемблера

Тип идентификатора на языке GPA-2022	Тип идентификатора на языке ассемблера	Пояснение
int bool	DWORD	Хранит целочисленный тип данных(для bool: 0 или 1).
string	DWORD	Хранит указатель на начало строки.
l	BYTE DWORD	Литералы: символьные(строковые), целочисленные

7.3 Статическая библиотека

В языке GPA-2022 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++ и автоматически подключается на начальном этапе генерации кода. Все функции статической библиотеки описаны в разделе 1.18.

7.4 Особенности алгоритма генерации кода

В языке GPA-2022 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2.

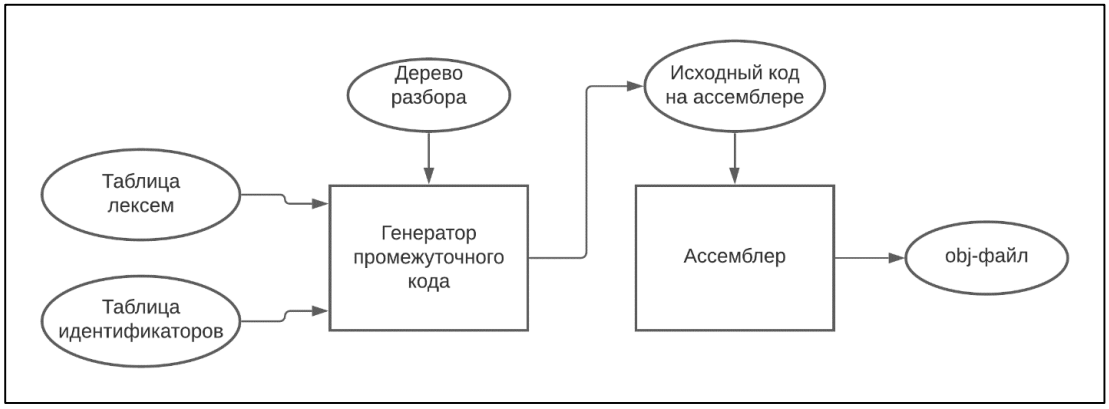

Рисунок 7.2 - Структура генератора кода

7.5 Входные параметры, управляющие генерацией кода

На вход генератору кода поступают таблицы лексем и идентификаторов исходного кода программы на языке GPA-2022. Результат работы генератора кода выводятся в файл с расширением .asm.

7.6 Контрольный пример

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

Глава 8 Тестирование транслятора

8.1 Общие положения

При возникновении ошибки на каком-либо этапе трансляции, она обрабатывается в главном файле программ: ошибка выводится на консоль и записывается в протокол работы.

8.2 Результаты тестирования

В таблице 8.1 приведены ошибки, генерируемые в процессе считывания входного файла, а также в процессе лексического, синтаксического и семантического анализов.

Таблица 8.1 – Результаты тестирования транслятора

Исходный код	Генерируемая ошибка
int Func fi1(int value) return value; }; …	Ошибка 600: Синтаксическая ошибка. Неверная структура программы
int Func fi1(int value){ var int new; new = восемь; return new; }; …	Ошибка 113: Лексическая ошибка. Слово не распознано.
int Func fi1(int value){ return new; }; …	Ошибка 122: Семантическая ошибка. Использована необъявленная переменная

int Func fi1(int value){
var int new;
new = 8;
var int new;
return new;
}; …
	Ошибка 123: Семантическая ошибка. Повторное объявление имени

int Func fi1(int value){ var string new; new = ‘Hello Word’; return new;	Ошибка 121: Семантическая ошибка. Тип возвращаемого значения не соответствует типу функции

}; …	

int Func fi1(int value){
  var string new;
  new =value;
  return 0;
}; …
	Ошибка 127: Семантическая ошибка. Несоответствие типов в выражении

int Func fi1(int value){
  return value;
} ;
 start
{
  var int x;
  x = fi1(‘Hello’);
};
	Ошибка 128: Семантическая ошибка. Несовпадение фактических и формальных параметров функции

int Func fi1(int value){
 	
  if(value equal ‘str’)
  [
  ]
  return 0;
}; …
	Ошибка 125: Семантическая ошибка. Ошибка в условии условного оператора

Заключение

В ходе проделанной работы был разработан транслятор для языка программирования GPA-2022.

В пояснительной записке описана реализация поставленных в рамках курсового проекта ряда задач:

– реализованы 2 арифметических оператора для вычисления выражений;

– реализованы три типа данных;

– поддерживается оператор вывода;

– присутствует подключаемая стандартная библиотека;

– более 2000 строк кода;

– реализована функция лексического сравнения строк;

– реализована условная конструкция;

– реализована функция вывода длины строки;

– присутствует арифметика целочисленных литералов.

Таким образом была достигнута поставленная цель по разработке компилятора GPA-2022, были учтены все требования, все задачи курсового проекта выполнены.

ПРИЛОЖЕНИЕ А

```
int Func increase2(int value)
{
     let int newvalue;
     newvalue = value + value;
     return newvalue;
};
int Func increase3(int value)
{
     let int newvalue;
     newvalue = value + increase2(value);
     return newvalue;
};
start
{
     let int y;
     y = 15;
     let int x;
     x = (increase3(20) + 2)-(15 | 16);
     write(x);
     let string str1;
     let string str2;
     str1 = 'This my string';
     str2 = 'This string';
     let bool z;
     z = strcmp(str1, str2);
     if(z)
     [
          let int len;
          len = strlen(str1);
          if( len equal 10)
          [
               write(len);
          ]

          write(str2);
     ]

};
```

Листинг – Контрольный пример

LEXEME	№ IN IT	№ IN CODE	OPERATOR VALUE
t		1	i
f		1	F
i	4	1	p
(		1	(
t		1	i
i	5	1	v
)		1	)
{		2	{
v		3	l
t		3	i
i	6	3	n
;		3	;
i	6	4	n
=		4	=
i	5	4	v
i	5	4	v
o		4	+
;		4	;
r		5	r
i	6	5	n
;		5	;
}		6	}
;		6	;
t		8	i
f		8	F
i	7	8	p
(		8	(

Рисунок – Часть таблицы лексем

~ ~ ~TABLE OF IDENTIFIERS~ ~ ~

TYPE:	ID:	DATA TYPE:	№ IN LT	VALUE(PARM):
FUNCTION	strlen	INT	-1	
FUNCTION	strcmp	BOOL	-1	
FUNCTION	strcpy	STR	-1	
FUNCTION	strcat	STR	-1	
FUNCTION	pow2	INT	2	
PARAMETER	value_pow2	INT	5	
VARIABLE	newvalue_pow2	INT	10	
FUNCTION	pow3	INT	25	
PARAMETER	value_pow3	INT	28	
VARIABLE	newvalue_pow3	INT	33	
VARIABLE	y_start	INT	53	
LITERAL	L1	INT	57	15
VARIABLE	x_start	INT	61	
LITERAL	L2	INT	68	20
LITERAL	L3	INT	71	2
LITERAL	L4	INT	77	16
VARIABLE	str1_start	STR	87	
VARIABLE	str2_start	STR	91	
LITERAL	L5	STR	95	'This my string'
LITERAL	L6	STR	99	'This string'
VARIABLE	z_start	BOOL	103	
VARIABLE	len_start	INT	121	
LITERAL	L7	INT	134	10

Рисунок – Таблица идентификаторов

ПРИЛОЖЕНИЕ Б

```
#define NS(n)      GRB::Rule::Chain::N(n)
#define TS(n)      GRB::Rule::Chain::T(n)
      Greibach greibach(NS('S'), TS('$'),
            8,

            Rule(NS('S'), GRB_ERROR_SERIES + 0,                                    //
Неверная структура программы
                  3,
                  Rule::Chain(5, TS('s'), TS('{'), NS('N'), TS('}'), TS(';')),
                  Rule::Chain(14, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'),
TS('{'), NS('N'), TS('r'), NS('R'), TS(';'), TS('}'), TS(';'), NS('S')),
                  Rule::Chain(13, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'),
TS('{'), TS('r'), NS('R'), TS(';'), TS('}'), TS(';'), NS('S'))

            ),

            Rule(NS('N'), GRB_ERROR_SERIES + 1,                                //
Ошибочный оператор
                  10,
                  Rule::Chain(4, TS('v'), TS('t'), TS('i'), TS(';')),
                  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),
                  Rule::Chain(5, TS('v'), TS('t'), TS('i'), TS(';'), NS('N')),
                  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),
                  Rule::Chain(9, TS('v'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'),
TS(')'), TS(';'), NS('N')),
                  Rule::Chain(5, TS('w'), TS('('), NS('R'), TS(')'), TS(';')),
                  Rule::Chain(6, TS('w'), TS('('), NS('R'), TS(')'), TS(';'), NS('N')),
                  Rule::Chain(8, TS('?'), TS('('), NS('I'), TS(')'), TS('['), NS('N'),
TS(']'), NS('N')),
                  Rule::Chain(7, TS('?'), TS('('), NS('I'), TS(')'), TS('['), NS('N'),
TS(']')),
                  Rule::Chain(9, TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';'),
NS('N'), TS('}'), TS(';'))
                  //Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), TS(';'),
NS('N'))
            ),

            Rule(NS('E'), GRB_ERROR_SERIES + 2,                                // Ошибка в
выражении
                  10,
                  Rule::Chain(1, TS('i')),
                  Rule::Chain(1, TS('l')),
                  Rule::Chain(3, TS('('), NS('E'), TS(')')),
                  Rule::Chain(3, TS('i'), TS('('), TS(')')),
                  Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),
                  Rule::Chain(2, TS('i'), NS('M')),
                  Rule::Chain(2, TS('l'), NS('M')),
                  Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),
                  Rule::Chain(4, TS('i'), TS('('), TS(')'), NS('M')),
                  Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M'))


            ),

            Rule(NS('M'), GRB_ERROR_SERIES + 3,
                  2,

                  Rule::Chain(2, TS('o'), NS('E')),
                  Rule::Chain(3, TS('o'), NS('E'), NS('M'))
```

```
		),

		Rule(NS('F'), GRB_ERROR_SERIES + 4, // ошибка в параметрах
			2,
			Rule::Chain(2, TS('t'), TS('i')),
			Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))
		),

		Rule(NS('W'), GRB_ERROR_SERIES + 5, // ошибка в параметрах вызываемой ф-ции
			4,
			Rule::Chain(1, TS('i')),
			Rule::Chain(1, TS('l')),
			Rule::Chain(3, TS('i'), TS(','), NS('W')),
			Rule::Chain(3, TS('l'), TS(','), NS('W'))
		),

		Rule(NS('R'), GRB_ERROR_SERIES + 6, // знач только перемнная или литерал
			2,
			Rule::Chain(1, TS('i')),
			Rule::Chain(1, TS('l'))
		),

		Rule(NS('I'), GRB_ERROR_SERIES + 7, // ошибка в услов операторе
			6,
			Rule::Chain(3, TS('i'), TS('o'), TS('i')),
			Rule::Chain(1, TS('l')),
			Rule::Chain(1, TS('i')),
			Rule::Chain(3, TS('i'), TS('o'), TS('l')),
			Rule::Chain(3, TS('l'), TS('o'), TS('i')),
			Rule::Chain(3, TS('l'), TS('o'), TS('l'))
		)


                                        );
```

Листинг 1 – Правила, описывающие грамматику языка

ПРИЛОЖЕНИЕ В

```
struct MfstState                    // состояние автомата (для сохранения)
      {
            short lenta_position;               // позиция на ленте
            short nrule;                              // номер текущего правила
            short nrulechain;                   // номер текущей цепочки, текущего
правила
            MFSTSTACK st;                             // стек автомата
            MfstState();
            MfstState(
                  short pposition,              // позиция на ленте
                  MFSTSTACK pst,                    // стек автомата
                  short pnrulechain             // номер текущей цепочки, текущего
правила
            );
            MfstState(
                  short pposition,              // позиция на ленте
                  MFSTSTACK pst,                    // стек автомата
                  short pnrule,                     // номер текущего правила
                  short pnrulechain             // номер текущей цепочки, текущего
правила
            );
      };

      struct Mfst                   // магазинный автомат
      {
            enum RC_STEP {                // код возвтара функции step
                  NS_OK,                        // найдено правило и цепочка,
цепочка записана в стек
                  NS_NORULE,              // не найдено правило грамматики (ошибка
в грамматике)
                  NS_NORULECHAIN,         // не найдена подходящая цепочка правила
(ошибка в исходном коде)
                  NS_ERROR,               // неизвестный нетерминальный символ
грамматики
                  TS_OK,                        // тек. символ ленты == вершине
стека, продвинулась лента, pop стека
                  TS_NOK,                       // тек. символ ленты != вершине
стека, восстановлено состояние
                  LENTA_END,              // текущая позиция ленты >= lenta_size
                  SURPRISE                // неожиданный код возврата (ошибка в
step)
            };

            struct MfstDiagnosis    // диагностика
            {
                  short lenta_position;         // позиция на ленте
                  RC_STEP rc_step;              // код завершения шага
                  short nrule;                        // номер правила
                  short nrule_chain;                  // номер цепочки правила
                  MfstDiagnosis();
                  MfstDiagnosis(
                        short plenta_position,  // позиция на ленте
                        RC_STEP prt_step,       // код завершения шага
                        short pnrule,                 // номер правила
                        short pnrule_chain            // номер цепочки правила
                  );
            } diagnosis[MFST_DIAGN_NUMBER];           // последние самые глубокие
сообщения

            GRBALPHABET* lenta;                       // перекодированная (TS/NS)
лента (из LEX)
```

```
            short lenta_position;               // текущая позиция на ленте
            short nrule;                              // номер текущего правила
            short nrulechain;                  // номер текущей цепочки, текущего
правила
            short lenta_size;                  // размер ленты
            GRB::Greibach grebach;             // грамматика Грейбах
            LA::Tables lex;                           // результат работы
лексического анализатора
            MFSTSTACK st;                             // стек автомата
            std::stack<MfstState> storestate;  // стек для сохранения состояний
            Mfst();
            Mfst(
                  LA::Tables pt,                      // результат работы
лексического анализатора
                  GRB::Greibach pgrebach       // грамматика Грейбах
            );
            char* getCSt(char* buf);           // получить содержимое стека
            char* getCLenta(char* buf, short pos, short n = 25); // лента: n
символов с pos
            char* getDiagnosis(short n, char* buf);        // получить n-ю строку
диагностики или 0x00
            bool savestate();                  // сохранить состояние автомата
            bool reststate();                  // восстановить состояние автомата
            bool push_chain(                   // поместить цепочку правила в
стек
                  GRB::Rule::Chain chain       // цепочка правила
            );
            RC_STEP step();                           // выполнить шаг автомата
            bool start();                             // запустить автомат
            bool savediagnosis(
                  RC_STEP pprc_step            // код завершения шага
            );
            void printrules();                        // вывести
последовательность правил

            struct Deducation       // вывод
            {
                  short size;             // количество шагов в выводе
                  short* nrules;          // номера правил грамматики
                  short* nrulechains;     // номера цепочек правик грамматики
(nrules)
                  Deducation() { size = 0; nrules = 0; nrulechains = 0; };
            } deducation;

            bool savededucation();  // сохранить дерево вывода
      };
```

Листинг – Структура магазинного конечного автомата

ПРИЛОЖЕНИЕ Г

```
Шаг : Правило                Входная лента                  Стек
0   : S->tfi(F){NrR;};S      tfi(ti){vti;i=ioi;ri;};tf      S$
0   : SAVESTATE:             1
0   :                        tfi(ti){vti;i=ioi;ri;};tf      tfi(F){NrR;};S$
1   :                        fi(ti){vti;i=ioi;ri;};tfi      fi(F){NrR;};S$
2   :                        i(ti){vti;i=ioi;ri;};tfi(      i(F){NrR;};S$
3   :                        (ti){vti;i=ioi;ri;};tfi(t      (F){NrR;};S$
4   :                        ti){vti;i=ioi;ri;};tfi(ti      F){NrR;};S$
5   : F->ti                  ti){vti;i=ioi;ri;};tfi(ti      F){NrR;};S$
5   : SAVESTATE:             2
5   :                        ti){vti;i=ioi;ri;};tfi(ti      ti){NrR;};S$
6   :                        i){vti;i=ioi;ri;};tfi(ti)      i){NrR;};S$
7   :                        ){vti;i=ioi;ri;};tfi(ti){      ){NrR;};S$
8   :                        {vti;i=ioi;ri;};tfi(ti){v      {NrR;};S$
9   :                        vti;i=ioi;ri;};tfi(ti){vt      NrR;};S$
10  : N->vti;                vti;i=ioi;ri;};tfi(ti){vt      NrR;};S$
10  : SAVESTATE:             3
10  :                        vti;i=ioi;ri;};tfi(ti){vt      vti;rR;};S$
11  :                        ti;i=ioi;ri;};tfi(ti){vti      ti;rR;};S$
12  :                        i;i=ioi;ri;};tfi(ti){vti;      i;rR;};S$
13  :                        ;i=ioi;ri;};tfi(ti){vti;i      ;rR;};S$
14  :                        i=ioi;ri;};tfi(ti){vti;i=      rR;};S$
15  : TS_NOK/NS_NORULECHAIN
15  : RESSTATE
15  :                        vti;i=ioi;ri;};tfi(ti){vt      NrR;};S$
16  : N->vti;N               vti;i=ioi;ri;};tfi(ti){vt      NrR;};S$
16  : SAVESTATE:             3
16  :                        vti;i=ioi;ri;};tfi(ti){vt      vti;NrR;};S$
17  :                        ti;i=ioi;ri;};tfi(ti){vti      ti;NrR;};S$
18  :                        i;i=ioi;ri;};tfi(ti){vti;      i;NrR;};S$
19  :                        ;i=ioi;ri;};tfi(ti){vti;i      ;NrR;};S$
20  :                        i=ioi;ri;};tfi(ti){vti;i=      NrR;};S$
21  : N->i=E;                i=ioi;ri;};tfi(ti){vti;i=      NrR;};S$
21  : SAVESTATE:             4
21  :                        i=ioi;ri;};tfi(ti){vti;i=      i=E;rR;};S$
22  :                        =ioi;ri;};tfi(ti){vti;i=i      =E;rR;};S$
23  :                        ioi;ri;};tfi(ti){vti;i=io      E;rR;};S$
24  : E->i                   ioi;ri;};tfi(ti){vti;i=io      E;rR;};S$
24  : SAVESTATE:             5
24  :                        ioi;ri;};tfi(ti){vti;i=io      i;rR;};S$
25  :                        oi;ri;};tfi(ti){vti;i=ioi      ;rR;};S$
26  : TS_NOK/NS_NORULECHAIN
26  : RESSTATE
26  :                        ioi;ri;};tfi(ti){vti;i=io      E;rR;};S$
27  : E->i()                 ioi;ri;};tfi(ti){vti;i=io      E;rR;};S$
27  : SAVESTATE:             5
27  :                        ioi;ri;};tfi(ti){vti;i=io      i();rR;};S$
28  :                        oi;ri;};tfi(ti){vti;i=ioi      ();rR;};S$
29  : TS_NOK/NS_NORULECHAIN
29  : RESSTATE
```

Рисунок – Работа синтаксического анализатора

```
0   : S->tfi(F){NrR;};S
4   : F->ti
8   : N->vti;N
12  : N->i=E;
14  : E->iM
15  : M->oE
16  : E->i
19  : R->i
23  : S->tfi(F){NrR;};S
27  : F->ti
31  : N->vti;N
35  : N->i=E;
37  : E->iM
38  : M->oE
39  : E->i(W)
41  : W->i
45  : R->i
49  : S->s{N};
51  : N->vti;N
55  : N->i=E;N
57  : E->l
59  : N->vti;N
63  : N->i=E;N
65  : E->(E)M
66  : E->i(W)M
68  : W->l
70  : M->oE
71  : E->l
73  : M->oE
74  : E->(E)
75  : E->lM
76  : M->oE
77  : E->l
80  : N->w(R);N
82  : R->i
85  : N->vti;N
89  : N->vti;N
93  : N->i=E;N
95  : E->l
97  : N->i=E;N
99  : E->l
101 : N->vti;N
105 : N->i=E;N
107 : E->i(W)
109 : W->i,W
111 : W->i
114 : N->?(I)[N]
116 : I->i
119 : N->vti;N
123 : N->i=E;N
125 : E->i(W)
127 : W->i
130 : N->?(I)[N]N
132 : I->iol
137 : N->w(R);
139 : R->i
143 : N->w(R);
145 : R->i
```

Рисунок – Результат работы синтаксического анализатора

ПРИЛОЖЕНИЕ Д

```
bool PN::PolishNotation(int n, LT::LexTable& lextable, IT::IdTable&
idtable)
{

	std::queue<LT::Entry> current;
	std::stack<LT::Entry> stack;

	int i = n;
	int priority;
	int parCount = 0;
	int indID = -1;

	while (lextable.table[i].lexema != ';')
	{
		priority = getP(lextable.table[i]);

		if (lextable.table[i].lexema == 'i' && lextable.table[i +
1].lexema == '(') // попали на вызов функции.
		{
			indID = lextable.table[i].indID;
			i++;
			priority = getP(lextable.table[i]);
			while (priority < 2)
			{
				if (priority == 0)
				{
					current.push(lextable.table[i]);
					parCount++;
				}
				else if (priority ==
1)stack.push(lextable.table[i]);
				else if (priority == -2);
				else if (priority == -1)
				{
					while (getP(stack.top()) != 1)		//
выталкиваем стек пока не встретим открывашку.
					{
						current.push(stack.top());
						stack.pop();
					}
					stack.pop();
					current.push({ '@', lextable.table[i].sn,
lextable.table[i].idxLT, indID });
					i++;
					priority = getP(lextable.table[i]);
					break;
				}

				i++;
				priority = getP(lextable.table[i]);
			}
		}

		if (priority == 0) current.push(lextable.table[i]);
		else if (priority == 1) stack.push(lextable.table[i]);
```

```
            else if (priority == 2 || priority == 3 || priority == 4 ||
priority == 5)
            {
                  while (!stack.empty())
                  {
                        if (getP(stack.top()) >= priority)
                        {
                              current.push(stack.top());
                              stack.pop();
                        }
                        else break;
                  }
                  stack.push(lextable.table[i]);
            }
            else if (priority == -1)
            {
                  while (getP(stack.top()) != 1)              // выталкиваем
стек пока не встретим открывашку.
                  {
                        current.push(stack.top());
                        stack.pop();
                  }
                  stack.pop();
            }
            else if (priority == -3) break;
            i++;
      }
      while (!stack.empty())
      {
            current.push(stack.top());
            stack.pop();
      }
      current.push(lextable.table[i]);
      //обратная польская запись построена.


      int posLast = i; // позиция последнего символа выражения до польской
записи
      i = n;

      for (i; i <= posLast; i++) {
            lextable.table[i] = { '#' , -1 , i , -1 };
            while (!current.empty())
            {
                  lextable.table[i] = current.front();
                  lextable.table[i].idxLT = i;
                  current.pop();
                  i++;
            }

      }
      return true;
}

int PN::getP(LT::Entry table)
{

      char token = table.lexema;
```

```
      if (token == 'o') token = table.operatorValue;

      if (token == '~') return 5;
      else if (token == '+' || token == '-') return 4;
      else if (token == '&') return 3;
      else if (token == '|') return 2;
      else if (token == '(') return 1;
      else if (token == ')') return -1;
      else if (token == ',') return -2;
      else if (token == ';') return -3;
      else return 0;

}
```

Листинг – Алгоритма преобразования выражений к польской записи

ПРИЛОЖЕНИЕ Е

```
.586P
.MODEL FLAT, STDCALL
includelib kernel32.lib
includelib libucrt.lib
includelib ..\Debug\Static_Library.lib
EXTRN Strcmp: PROC
EXTRN Strcat: PROC
EXTRN Strcpy: PROC
EXTRN Strlen: PROC
EXTRN ConsoleInt: PROC
EXTRN ConsoleStr: PROC
EXTRN ConsoleBool: PROC
EXTRN consolpause: proc
ExitProcess PROTO : DWORD

.STACK 4096
.CONST

      L1 dd 15
      L2 dd 20
      L3 dd 2
      L4 dd 16
      L5 db 'This my string' , 0
      L6 db 'This string' , 0
      L7 dd 10

.DATA

      buffer BYTE 256 dup(0)
      newvalue_pow2 DD 0
      newvalue_pow3 DD 0
      y_start DD 0
      x_start DD 0
      str1_start DD ?
      str2_start DD ?
      z_start DD 0
      len_start DD 0

.CODE

pow2 PROC value_pow2 : DWORD

push value_pow2
push value_pow2
pop EAX
pop EBX
add EAX, EBX
push EAX
pop newvalue_pow2

mov eax, newvalue_pow2
ret
pow2 ENDP

pow3 PROC value_pow3 : DWORD

push value_pow3
push value_pow3
call pow2
push EAX
pop EAX
```

```
pop EBX
add EAX, EBX
push EAX
pop newvalue_pow3

mov eax, newvalue_pow3
ret
pow3 ENDP


main PROC
START:

push L1
pop y_start

push L2
call pow3
push EAX
push L3
pop EAX
pop EBX
add EAX, EBX
push EAX
push L1
push L4
pop EAX
pop EBX
OR EAX, EBX
push EAX
pop EBX
pop EAX
sub EAX, EBX
push EAX
pop x_start

mov EAX, x_start
push EAX
call ConsoleInt

push offset L5
pop str1_start

push offset L6
pop str2_start

push str1_start
push str2_start
call strcmp
push EAX
pop z_start


mov EAX,z_start
mov EBX, 1
sub EAX, EBX
je true1
jmp exit1

true1:
push str1_start
call strlen
push EAX
pop len_start
```

```
mov EAX,len_start
mov EBX, L7
sub EAX, EBX
je true2
jmp exit2

true2:
mov EAX, len_start
push EAX
call ConsoleInt


exit2:
mov EAX, str2_start
push EAX
call ConsoleStr


exit1:
push 0
call consolpause
call ExitProcess
main ENDP
end main
```

Листинг – Результат генерации ассемблерного кода