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 HKV-2024»

Выполнил студент Хуторцов Кирилл Владимирович
(Ф.И.О.)

Руководитель проекта асс. Ромыш Александра Сергеевна

Заведующий кафедрой к.т.н., доц. Смелов Владимир Владиславович

Консультанты асс. Ромыш Александра Сергеевна

Нормоконтролер асс. Ромыш Александра Сергеевна

Курсовой проект защищен с оценкой

Минск 2024

Содержание

Введение

Транслятор – это комплекс отдельных программ, позволяющих преобразовывать исходный код на одном языке программирования в исходный код на другом языке программирования.

Классический транслятор состоит из следующих частей:

- лексический анализатор;
- синтаксический анализатор;
- семантический анализатор;
- генератор кода, или интерпретатор.

Все части транслятора, взаимодействуя между собой, обрабатывают входной текст и строят для него эквивалентный текст на понятном компьютеру языке программирования.

1. Спецификация языка программирования

1.1 Характеристика языка программирования

Язык программирования HKV-2024 является языком программирования высокого уровня. Он является компилируемым. В языке отсутствует преобразование типов. В языке поддерживается 4 типа данных: целочисленный (numb), строковый (stroke), символьный (symbol), логический (boolean). В стандартной библиотеке имеются функции для работы с целочисленным и строковыми типами данных: генерация случайных чисел, вычисление длины строки, остаток от деления.

1.2 Определение алфавита языка программирования

Символы, используемые на этапе выполнения: [a…z], [A…Z], [0…9], [а…я], [А…Я], символы пробела, перевода строки, спецсимволы: [] () , ; : + - / * % > < {}|&.

1.3 Применяемые сепараторы

Символы сепараторы служат в качестве разделителей цепочек языка во время обработки исходного текста программы с целью разделения на токены. Они представлены в таблице 1.1.

Таблица 1.1 – Символы-сепараторы

Символ(ы)	Назначение
Пробел	Разделитель цепочек. Допускается везде, кроме имен идентификаторов и ключевых слов
[…]	Блок функции или цикла
(…)	Блок параметров функции
,	Разделитель параметров функций
+ - * / %	Арифметические операции
> < } { & |	Логические операторы (Операции сравнения: больше, меньше, больше или равно, меньше или равно, логическое и, логическое или)
;	Разделитель программных конструкций
=	Оператор присваивания

Использование указанных символов-сепараторов обеспечивает корректное разбиение исходного текста программы на токены.

1.4 Применяемые кодировки

Для написания программ на языке HKV-2024 используется кодировка Windows – 1251, представленная на рис.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	J 0408	¤ 00A4	Ґ 0490	¦ 00A6	§ 00A7	Ё 0401	© 00A9	Є 0404	« 00AB	¬ 00AC	- 00AD	® 00AE	Ї 0407
B0	° 00B0	± 00B1	I 0406	і 0456	ґ 0491	µ 00B5	¶ 00B6	· 00B7	ё 0451	№ 2116	є 0454	» 00BB	j 0458	S 0405	s 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 Алфавит вводных символов

Windows-1251 — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии.

1.5 Типы данных

В языке HKV-2024 реализованы 4 фундаментальных типа данных: целочисленный, строковый, символьный, логический. Описание типов приведено в таблице 1.2.

Таблица 1.2 – Типы данных языка HKV-2024

Типы данных	Характеристика
Целочисленный тип данных **numb**	Является целочисленным типом данных. Этот тип данных занимает 4 байта. Предназначен для арифметических операций над числами. Инициализация по умолчанию: 0. Поддерживаемые операции: + (бинарный) – Оператор сложения; - (бинарный) – Оператор вычитания; * (бинарный) – Оператор умножения; / (бинарный) – Оператор деления;

Окончание таблицы 1.2

	= (бинарный) – Оператор присваивания %(бинарный) – Остаток от деления В качестве операторов условия или условия цикла можно использовать следующие операторы: > (бинарный) – Оператор “больше”; < (бинарный) – Оператор “меньше” } (бинарный) – Оператор “больше либо равно” { (бинарный) – Оператор “меньше либо равно”
Строковый тип данных **stroke**	Фундаментальный тип данных. Используется для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 255. Инициализация по умолчанию: строка нулевой длины ". Операции над данными строкового типа: =(бинарный)оператор присваивания
Символьный тип данных **symbol**	Фундаментральный тип данных. Используется для работы с символом, занимающим 1 байт. Инициализация по умолчанию: символ нулевой длины “”. Операции над данными символьного типа: =(бинарный)оператор присваивания.
Логический тип данный **boolean**	Является логическим типом данных. Переменные данного типа могу принимать 2 значения:true или false. Операции над данными логического типа: |(бинарный) – Логическое или &(бинарный) – Логическое И

Приведенные типы данных обеспечивают базовую функциональность языка HKV-2024 для работы с числами, строками, символами и логическими выражениями.

1.6 Преобразование типов данных

Преобразование типов данных в языке HKV-2024 не поддерживается, так как язык HKV-2024 является типизированным.

1.7 Идентификаторы

Общее количество идентификаторов ограниченно максимальным размером таблицы идентификаторов (4096). Идентификаторы могут содержать символы как нижнего регистра, так и верхнего. Максимальная длина идентификатора равна 10 символам. Идентификаторы, объявленные внутри функционального блока, получают область видимости , идентичную имени функции, внутри которой они объявлены. Данные правила действуют для всех идентификаторов. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не

должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, параметр функции.

Правило составления идентификатора:

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

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

<идентификатор> ::= <буква> {(<буква>| <цифра>)}

1.8 Литералы

С помощью литералов осуществляется инициализация переменных. Все литералы являются rvalue. Имеются литералы: целочисленные десятичного представления, строковые, логические, а также символьные. Подробное описание литералов языка HKV-2024 представлены в таблице 1.3.

Таблица 1.3 – Литералы

Литералы	Пояснение
Целочисленные литералы в десятичном представлении	Последовательность цифр 0…9 с предшествующим знаком минус или без него (знак минус не отделяется пробелом)
Строковые литералы	Набор символов алфавита языка, заключенных в одинарные кавычки
Логические литералы	Может принимать 2 значения: true или false
Символьные литералы	Символ алфавита языка, заключеный в двойные кавычки

Ограничения на целочисленные литералы: не могут начинаться с 0, если их значение не 0; если литерал отрицательный, после знака “-” не может идти 0.

1.9 Объявление данных

Для объявления переменной используется ключевое слово **new,** после которого указывается тип данных и имя идентификатора.

Пример объявления числового типа данных с инициализации:

new numb x = 52;

Пример объявления строкового типа данных с инициализацией:

new stroke str = 'здраствуй мир';

Пример объявления логического типа данные с инициализацией:

new boolean b = true;

new boolean c = false;

Пример объявления символьного типа данных с инициициализацией:

New symbol x = “X”;

Для объявления функций используется ключевое слово **func**, перед которым указывается тип функции. Далее обязателен список параметров и тело функции.

1.10 Инициализация данных

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть только идентификаторы и литералы. При объявлении переменные инициализируются значением по умолчанию. Для **numb** значение 0, для **stroke, symbol** строка нулевой длины (“”), для **boolean** – false.

1.11 Инструкции языка

Инструкции языка HKV-2024 представлены в таблице 1.4.

Таблица 1.4 – Инструкции языка

Инструкция	Реализация
Объявление переменной	**new** <тип данных>< идентификатор >;
Возврат значения из функции	**return** <идентификатор> | <литерал>;
Вывод данных	**write** <идентификатор> | <литерал>;
Вызов функции	<идентификатор функции>(<список параметров>);
Присваивание	<идентификатор> = <выражение>; Выражением может быть идентификатор, литерал, или вызов функции соответствующего типа. Для целочисленного типа выражение может быть дополнено арифметическими операциями.

Все инструкции языка HKV-2024 должны строго соответствовать указанным синтаксическим правилам.

1.12 Операции языка

Операции языка HKV-2024 и их приоритет представлен в таблице 1.5.

Таблица 1.5 – Операции языка HKV-2024

Тип оператора	Оператор
Арифметические	+ - сложение (приоритет 5) - - разность (приоритет 5) * - умножение (приоритет 4) / - деление (приоритет 4) % - остаток от деления (приоритет 4)
Логические	> - больше (приоритет 7) < - меньше (приоритет 7) } – больше или равно (приоритет 7) { - меньше или равно (приоритет 7) &-логическое и (приоритет 9)

Окончание таблицы 1.5

	| - логическое или (приоритет 10)

Для повышения приоритета выполнения операций используются круглые скобки “( )”.

1.13Выражения и их вычисления

Вычисление выражений – одна из важнейших задач языков программирования. Всякое выражение составляется согласно следующим правилам:

1. Допускается использовать скобки для смены приоритета операций;
2. Выражение записывается в строку без переносов;
3. Использование двух подряд идущих операторов не допускается;
4. 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. Преобразование выражений приведено в главе 5.

1.14 Конструкции языка

Программа на языке HKV-2024 оформляется в виде функций пользователя и главной функции. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для лучшей читаемости кода.

Использование осмысленных имен переменных и функций также улучшает восприятие и сопровождение программного кода.

Структурированное оформление функций упрощает отладку и тестирование программы, а четкое разделение логических блоков способствует более быстрому пониманию кода.

Программные конструкции языка HKV-2024 представлены в таблице 1.6.

Таблица 1.6 – Конструкции языка HKV-2024

Конструкция	Реализация
Главная функция	**main** **[** … **return** <идентификатор/литерал>; **]**

Окончание таблицы 1.6

Внешняя функция	<тип данных> **func** <идентификатор> (<тип> <идентификатор>, ...) **[** … **return** <идентификатор/литерал>; **]**
Цикл	**state:**<идентификатор1><логический оператор><идентификатор2>**$** **cycle[…]** **$**
Условное выражение	**state:** <идентификатор1> <логический оператор><идентификатор2>**$** **correctly:[**<идентификатор> = <литерал> \| <идентификатор>**]** **wrong:[**<идентификатор> = <литерал> \| <идентификатор>**]** **$**

Программные конструкции языка HKV-2024 обеспечивают гибкость и удобство при разработке алгоритмов различной сложности.

1.15 Область видимости идентификаторов

Область видимости: сверху вниз. Переменные, объявленные в одной функции не доступны в другой. Все операции и объявления происходят внутри какого-либо блока или тела функции. Каждая переменная или параметр функции получают область видимости – название функции, в которой они находятся.

Все идентификаторы являются локальными и обязаны быть объявленными внутри какой-либо функции. Глобальных переменных нет. Параметры видны только внутри функции, в которой объявлены.

1.16 Семантические проверки

В языке программирования HKV-2024 выполняются следующие семантические проверки:

1. Наличие функции main – точки входа в программу;
2. Единственность точки входа;
3. Переопределение идентификаторов;
4. Использование идентификаторов без их объявления;
5. Проверка соответствия типа функции и возвращаемого параметра;
6. Правильность передаваемых в функцию параметров: количество, типы, объявления;
7. Превышение размера строковых и числовых литералов;
8. Проверка совпадений типов в операциях;

1.17 Распределение оперативной памяти на этапе выполнения

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные и параметры функций. Локальная область видимости в исходном коде определяется за счет использования правил именования идентификаторов и регулируется их областью видимости, что и обуславливает их локальность на уровне исходного кода, несмотря на то, что в оттранслированным в язык ассемблера коде переменные имеют глобальную область видимости.

1.18 Стандартная библиотека и ее состав

В языке HKV-2024 присутствует стандартная библиотека, которая подключается автоматически на этапе трансляции исходного кода в язык ассемблера.

Содержимое стандартной библиотеки представлено в таблице 1.8.

Таблица 1.8 – Стандартная библиотека языка HKV-2024

Функция	Описание
numb rand(numb b)	Целочисленная функция, возвращает псевдослучайное число в определенном диапазоне b.
numb strlen(stroke str)	Целочисленная функция, возвращает размер строки
numb input()	Целочисленная функция, возвращает число, введенное пользователем.

Стандартная библиотека языка HKV-2024 обеспечивает базовый набор функций для работы со строками, псевдослучайными числами и пользовательским вводом.

1.19 Вывод и ввод данных

Вывод данных осуществляется с помощью оператора **write**. Допускается использование оператора **write** с литералами и идентификаторами.

1.20 Точка входа

В языке HKV-2024 каждая программа должна содержать главную функцию (точку входа) **main**, с первой инструкции которой начнётся последовательное выполнение команд программы.

Должна иметься только одна точка входа main.

1.21 Препроцессор

Команды препроцессора в языке HKV-2024 отсутствуют.

1.22 Соглашения о вызове

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

Язык HKV-2024 транслируется в язык ассемблера, а затем - в объектный код.

1.24 Классификация сообщений транслятора

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Сообщения об ошибках имеют специфический постфикс, зависящий от этапа, на котором обнаружена ошибка.

Список постфиксов приведен в таблице 1.9.

Таблица 1.9 – Список префиксов ошибок в языке HKV-2024

Постфикс	Пояснение
[SIN#]	Указывает, что ошибка была обнаружена на стадии синтаксического анализа.
[LEX#]	Указывает, что ошибка была обнаружена на стадии лексического анализа.
[SEM#]	Указывает, что ошибка была обнаружена на стадии семантического анализа.

Используемые постфиксы помогают быстро определить этап разработки, на котором была допущена ошибка, и ускорить её исправление.

1.25 Контрольный пример

Контрольный пример демонстрирует главные особенности языка HKV-2024: его фундаментальные типы, основные структуры, функции, использование функция стандартной библиотеки. Исходный код контрольного примера представлен в приложении А.

2. Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

В языке HKV-2024 исходный код транслируется в язык Assembler. Транслятор языка разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые им функции, которые представлены в пункте 2.1. Для того чтобы получить ассемблерный код, используется выходные данные работы лексического анализатора, а именно таблица лексем и таблица идентификаторов. Для указания выходных файлов используются входные параметры транслятора, которые описаны в таблице 2.1. Структура транслятора языка HKV-2024 приведена на рисунке 2.1.

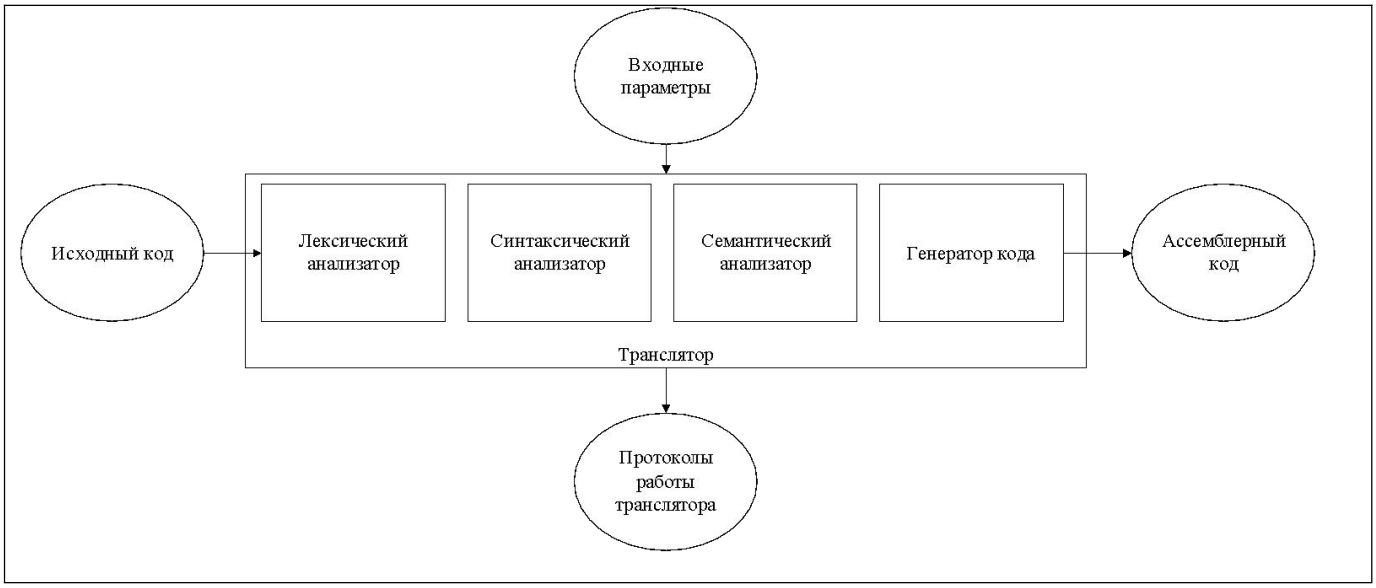

Рисунок 2.1 Структура транслятора языка программирования HKV-2024

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся последовательность символов входного языка. Он производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив отдельных слов (в теории компиляции вместо термина «слово» часто используют термин «токен»). Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация. Таблица лексем (ТЛ) и таблица идентификаторов (ТИ) являются входом для следующей фазы компилятора – синтаксического анализа (разбора, парсера).

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.

Синтаксический анализатор – часть компилятора, выполняющая синтаксический анализ, то есть проверку исходного кода на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией является дерево разбора.

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть проверку исходного кода на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

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

2.2 Перечень входных параметров транслятора

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

Таблица 2.1 – Входные параметры транслятора языка HKV-2024

Входной параметр	Описание параметра	Значение по умолчанию
-in:<путь к in-файлу>	Файл с исходным кодом на языке HKV-2024 , имеющий расширение .txt	Не предусмотрено
-log:<путь к log-файлу>	Файл журнала для вывода протоколов работы программы.	Значение по умолчанию: <имя in-файла>.log
-greibach:<путь к greibach-файлу>	Файл содержащий дерево разбора	Значение по умолчанию: <имя in-файла>.greibach
-LT:<путь к LT-файлу>	Файл содержащий таблицу лексем	Значение по умолчанию: <имя in-файла>.LT
-IT:<путь к IT-файлу>	Файл содержащий таблицу идентификаторов	Значение по умолчанию: <имя in-файла>.IT

Входные параметры транслятора позволяют гибко настроить процесс генерации файлов и диагностики работы анализаторов, обеспечивая точную настройку на различных этапах трансляции программы.

2.3 Перечень протоколов, формируемых транслятором и их содержимое

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

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

Таблица 2.2 – Протоколы, формируемые транслятором языка HKV-2024

Формируемый протокол	Описание выходного протокола
Файл журнала, заданный параметром "-log:"	Файл с протоколом работы транслятора языка программирования HKV-2024 .
Файл таблицы лексем, заданный параметром "-LT:"	Файл работы лексического анализатора. Содержит таблицу лексем
Файл таблицы идентификаторов, заданный параметром "-IT:"	Файл работы лексического анализатора. Содержит таблицу идентификаторов
Файл таблицы идентификаторов, заданный параметром "-greibach:"	Файл работы синтаксического анализатора. Содержит дерево разбора и протокол работы
Выходной файл, с расширением ".asm"	Результат работы программы – файл, содержащий исходный код на языке ассемблера.

Протоколы работы транслятора предоставляют подробную информацию о процессе анализа и трансляции программы, позволяя отслеживать этапы работы каждого анализатора и получать результат в виде исходного кода на языке ассемблера.

3. Разработка лексического анализатора

3.1 Структура лексического анализатора

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка, производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив токенов.

Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

Функции лексического анализатора:

- удаление «пустых» символов. Если «пустые» символы (пробелы, знаки табуляции и перехода на новую строку) будут удалены лексическим анализатором, синтаксический анализатор никогда не столкнется с ними (альтернативный способ, состоящий в модификации грамматики для включения «пустых» символов и комментариев в синтаксис, достаточно сложен для реализации);
- распознавание идентификаторов и ключевых слов;
- распознавание констант;
- распознавание разделителей и знаков операций.

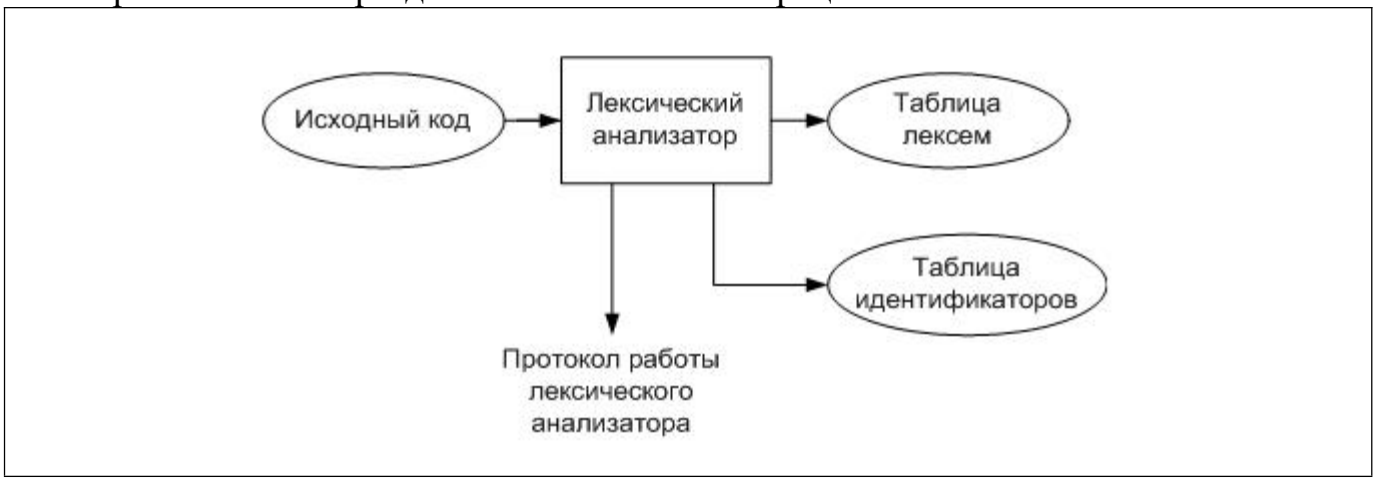

Рисунок 3.1 Структура лексического анализатора

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

3.2. Контроль входных символов

Таблица контроля входных символов представлена в приложении Б

Принцип работы таблицы заключается в соответствии значения каждого элемента значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, I – игнорируемый символ, S – символ-разделитель.

3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

- посимвольно считываем файл с исходным кодом программы;
- в отличие от других символов-разделителей, не записываем пробелы и символы табуляции в таблицу лексем;
- продолжаем считывание файла с исходным кодом программы до встречи с лексемой, отличной от пробела или символа табуляции.

3.4 Перечень ключевых слов

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие токенов и лексем приведено в таблице 3.1.

Таблица 3.1 – Соответствие токенов и лексем в языке HKV-2024

Токен	Лексема	Пояснение
numb,stroke,boolean, symbol	t	Названия типов данных языка.
Идентификатор	i	Содержит информацию о идентификаторе
Литерал	l	Литерал любого доступного типа.
func	f	Объявление функции.
return	r	Выход из функции/процедуры.
main	m	Главная функция.
new	n	Объявление переменной.
write	p	Вывод данных.
state	?	Указывает начало цикла/условного оператора.
cycle	v	Указывает на начало тела цикла.
$	$	Разделение конструкций в цикле/условном операторе.
;	;	Разделение выражений.
,	,	Разделение параметров функций.
[	[	Начало блока/тела функции.

Окончание таблицы 3.1

]	]	Закрытие блока/тела функции.
(	(	Передача параметров в функцию, приоритет операций.
)	)	Закрытие блока для передачи параметров, приоритет операций.
=	=	Знак присваивания.

+
-
*
/
}
{
>
<
%
&
	+
	-
	*
	/
	}
	{
	>
	<
	%
	&
		Знаки операций.

correctly	c	Указывает на достоверность условного выражения
wrong	w	Указывает на недостоверность условного выражения

В приложении Б находится пример конечного автомата, используемый для разбора цепочки символов.

3.5 Основные структуры данных

Структуры таблиц лексем и идентификаторов данных языка HKV-2024, используемых для хранения, представлены в приложении Б.

В таблице лексем содержатся сами лексемы, строка для каждой лексемы, в которой она была замечена. Так же размер самой таблицы лексем. В таблице идентификаторов содержится имя идентификатора, его номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение, а также имя родительской функции.

3.6 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами.

В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением.

3.7 Структура и перечень сообщений лексического анализа

Перечень сообщений представлен в приложении Б.

Сообщения об ошибках данной стадии имеют префикс [LEX#] что с легкостью дает пользователю понять, на каком этапе возникла ошибка.

3.8 Параметры лексического анализатора

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся в файл с таблицей лексем, файл с таблицей идентификаторов , а также в командную строку.

3.9 Алгоритм лексического анализа

Последовательность выполнения алгоритма работы лексического анализатора представлен ниже.

1) Разделение текста на отдельные лексемы.
2) Распознавание каждой строки в двумерном массиве с помощью автоматов.
3) При удачном прохождении информация заносится в таблицу лексем и идентификаторов. Возврат к шагу 2).
4) Формирование протокола работы
5) При невозможности обработать строку двумерного массива выводится сообщение об ошибке.
6) Конец работы лексического анализатора

3.10 Контрольный пример

Результат работы лексического анализатора – таблицы лексем и идентификаторов – представлен в приложении Б.

4. Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

Синтаксический анализатор: часть компилятора, выполняющая синтаксический анализ, то есть исходный код проверяется на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией– дерево разбора.

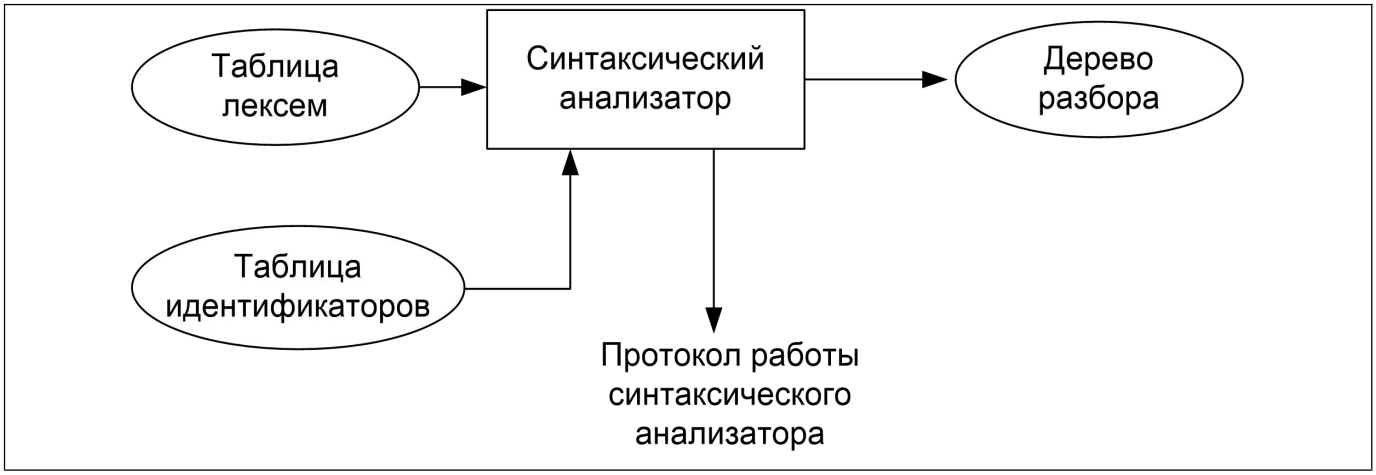

Рисунок 4.1 – Структура синтаксического анализатора

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

4.2 Контекстно-свободная грамматика, описывающая синтаксис

В синтаксическом анализаторе транслятора языка HKV-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 $a \quad T, \alpha \quad (T \quad N) \quad \{\lambda\}$; (или $\alpha \quad (T \quad N)^*$, или $\alpha \quad V^*$);

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

Описание нетерминальных символов содержится в таблице 4.1.

Таблица 4.1 – Таблица правил переходов нетерминальных символов

символ	Правила	Какие правила порождает
S	S–>tZ[N];S S–>tZ[N]S S–>m[N]	Стартовые правила, описывающее общую структуру программы
G	G–>ti G–>ti,G G–>i,G G–>l,G G–>i G–>l	Правила для параметров объявляемых функций; Правила для параметров вызываемой функции
Z	Z –>fiG	Правила для объявления функции
K	K –>:EA K –>:E$$ K –>:E$$N	Правила определяющие структуру условного выражения
A	A–>c:Y A–>w:Y A–>vY	Правила построения структуры условного выражения/цикла
Y	Y–>[N]A Y–>[N]	Правила построения тела условного выражения/цикла
W	W –>i=M C –>i=E C –>i=L C –>i=EVE	Правила вызова функции
V	V –>+ V –>- V –>* V –>/ V –>%	Правила построения арифметических операторов
N	N->nti; N->nti;N N->nD;N N->nD; N->i=E;N N->i=E; N->pE; N->pE;N N->?KN N->?K N->rM; N->pi;rM	Правила объявления переменных

Окончание таблицы 4.1

O	O –>| O –>& O –>> O –>< O –>} O –>{	Правила построения логических операторов
Q	Q–>(M) Q–>(M)VE	Правила передачи параметров в функцию статической библиотеки
L	L –>RQ L –>SQ L –>UQ	Правила вызова функций статической библиотеки
D	D –>tW	Правило инициализации переменной
M	M –>i M –>l	Правила вывода идентификатора/литерала
E	E –>i E –>l E –>i; E –>l; E –>iVE E –>lVE E –>iOE E –>lOE E–>SQ E –>UQ E –>RQ Ew –>i(G)	Правила построения выражений

Данная контекстно-свободная грамматика описывает основные конструкции языка HKV-2024, включая правила для объявления переменных, работы с функциями и операциями, что позволяет эффективно анализировать синтаксическую структуру программ."

4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку $M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$.

Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

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 представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблица 3.1 и 4.1.
Z	Алфавит специальных магазинных символов	Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $)
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
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

Конечный магазинный автомат, описанный в данной таблице, служит основой для синтаксического анализа языка HKV-2024, обеспечивая эффективное выполнение переходов и управление памятью с помощью стека.

4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора представляются в

виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка. Данные структуры 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. Иначе генерируется исключение.

4.6 Структура и перечень сообщений синтаксического анализатора

Сообщения генерируемые синтаксическим анализатором представлены в приложении В.

4.7. Параметры синтаксического анализатора и режимы его работы

Входной информацией для синтаксического анализатора является таблица лексем и идентификаторов. Кроме того используется описание грамматики в форме Грейбах. Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью выводятся в журнал работы синтаксического анализатора.

4.8. Принцип обработки ошибок

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке (если она найдена).

4.9. Контрольный пример

Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью приведены в приложении В.

5. Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа.

Рисунок 5.1. Структура семантического анализатора

Общая структура обособленно работающего семантического анализатора представлена на рисунке 5.1.

5.2 Функции семантического анализатора

Семантический анализатор проверяет правильность составления программных конструкций. При невозможности подобрать правило перехода будет выведен код ошибки, а так же код этой ошибки. Информация об ошибках выводится в консоль, а так же в протокол работы.

5.3 Структура и перечень семантических ошибок

Сообщения, формируемые семантическим анализатором, представлены в приложении Г.

5.4 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением.

5.5 Контрольный пример

Исходный код	Текст сообщения
main [new Numb b; return 0;];	Ошибка 317: [SEM]# Ошибка в объявление идентификатора (указан неправильный тип) Строка 3
main [numb b; return 0;];	Ошибка 302: [SEM]# В объявлении отсутствует ключевое слово new Строка 3
main [new numb b; new numb b; return 0;];	Ошибка 311: [SEM]# Повторное объявление идентификатора Строка 4
main [new numb b; new numb b; return 0;]; main[new stroke b; new numb e; return 0;]	Ошибка 308: [SEM]# Обнаружено несколько точек входа в main Строка 6

Контрольный пример демонстрирует различные типы семантических ошибок, которые могут возникнуть при объявлении идентификаторов и неправильном использовании ключевых слов в языке HKV-2024.

6. Вычисление выражений

6.1 Выражения, допускаемые языком

В языке HKV-2024 допускаются вычисления выражений целочисленного, а также логического типов данных с поддержкой вызова функций внутри целочисленных выражений.

Таблица 6.1. - приоритет операций

Операция	Значение приоритета
( )	0
*	4
/	4
+	5
-	5
%	4
|	10
&	9

Данная таблица отражает приоритет операций, используемых в выражениях языка HKV-2024, что позволяет правильно интерпретировать и вычислять выражения с учетом порядка выполнения операций.

6.2 Польская запись и принцип ее построения

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

Польская запись - это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: прямая и обратная, также известные как префиксная и постфиксная. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а, соответственно, до или после аргументов. Алгоритм построения польской записи:

- исходная строка: выражение;
- результирующая строка: польская запись;
- стек: пустой;
- исходная строка просматривается слева направо;
- операнды переносятся в результирующую строку;
- операция записывается в стек, если стек пуст;
- операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
- отрывающая скобка помещается в стек;
- закрывающая скобка выталкивает все операции;

6.3 Программная реализация обработки выражений

Программная реализация алгоритма преобразования выражений к польской записи представлена в приложении Г.

6.4 Контрольный пример

Пример преобразования выражения к польской записи представлен в таблице 6.2.

Преобразование выражений в формат польской записи в нашем случае необходимо для построения более простых алгоритмов при последующей обработки таблицы лексем.

Таблица 6.2 – Преобразование выражений к ПОЛИЗ

Исходное выражение	Стек	Результирующая строка
l*((l-l)/l)+l		
*((l-l)/l)+l		l
((l-l)/l)+l	*	l
(l-l)/l)+l	*,(	l
l-l)/l)+l	*,(,(	l
-l)/l)+1	*,(,(	l,l
l)/l)+l	*,(,(,-	l,l
)/l)+l	*,(,(,-	l,l,l
/l)+l	*,(	l,l,l,-
l)+l	*,(,/	l,l,l,-
)+l	*,(,/	l,l,l,-,l
+l	*	l,l,l,-,l,/
l	+	l,l,l,-,l,/,*
	+	l,l,l,-,l,/,*,l
		l,l,l,-,l,/,*,l,+

В приложении Г приведена изменённая таблица лексем, отображающая результаты преобразования выражений в польский формат.

7. Генерация кода

7.1 Структура генератора кода

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

Рисунок 7.1 – Структура генератора кода

Структура генератора кода обеспечивает правильное преобразование промежуточных данных в ассемблерный код, при этом гарантируя выявление и предотвращение ошибок в процессе трансляции.

7.2 Представление типов данных в оперативной памяти

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

Таблица 7.1 – Соответствия типов идентификаторов языка и языка ассемблера

Тип идентификатора на языке HKV-2024	Тип идентификатора на языке ассемблера	Пояснение
number	sdword	Хранит целочисленный тип данных.
stroke	dword	Хранит указатель на начало строки. Строка должна завешаться нулевым символом.
boolean	dword	Хранит логический тип данных
symbol	dword	Хранит указатель на символ, оканчивается нулевым символом

Эта таблица обеспечивает правильное отображение типов идентификаторов и ассемблером, что необходимо для корректной работы программы в процессе трансляции и выполнения.

7.3 Статическая библиотека

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

Стандартная библиотека находится в директории языка и при генерации кода подключается автоматически. Путь к библиотеке генерируется автоматически на стадии генерации кода.

7.4 Особенности алгоритма генерации кода

В процессе генерации используются векторы и строки. Отдельные сегменты сначала записываются в строки, а затем отправляются в вектор. В конце работы весь вектор последовательно выводится в файл.

7.5 Входные параметры генератора кода

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке HKV-2024 . Результаты 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 Д. Результат работы контрольного примера приведён в приложении Д.

8. Тестирование транслятора

8.1 Общие положения

В языке HKV-2024, при возникновении ошибки на одном из этапов, генерируется исключение, которое обрабатывается в главной функции. Затем код ошибки и сообщение выводится в консольное окно, а так же записывается в протокол работы.

8.2 Результаты тестирования

В таблице 8.1 приведены ошибки возникающие при считывании из файла, а так же на стадии лексического, синтаксического и семантического анализа.

Таблица 8.1 – Результаты тестирования транслятора

Исходный код	Диагностическое сообщение
main [new numb a=10; new str b='qwe'; new numb res ; state:a>b$ correctly:[res =10;] wrong:[res =12;] $ return 0;];	Ошибка 317: [SEM]# Ошибка в объявление идентификатора (указан неправильный тип) Строка 4
main [new numb a=10; new stroke b='qwe'; new numb res ; state:a>b$ correctly:[res =10;] wrong:[res =12;] $ return 0;];	Ошибка 304: [SEM]# Ошибка в условии условного выражения Строка 5

[
new numb a=10;
new numb b=12;
new numb res ;
];
	Ошибка 300: [LEX]# Отсутствует точка входа main
	Строка -1

Эта таблица демонстрирует типичные ошибки, возникающие при разных этапах трансляции, и помогает анализировать проблемы в исходном коде для корректного выполнения программы.Она является важным инструментом для анализа работы транслятора языка HKV-2024. Также она позволяет не только выявлять типичные ошибки, возникающие на различных этапах трансляции, но и систематизировать их для дальнейшего улучшения качества программного кода. Приведенные диагностические сообщения помогают разработчикам быстро находить и исправлять проблемные участки, что особенно важно при работе с крупными проектами.

Кроме того, данные таблицы служат основой для разработки дополнительных тестовых случаев, направленных на проверку устойчивости транслятора к различным сценариям использования. Это способствует не только улучшению пользовательского опыта, но и повышению надежности конечного программного обеспечения.

Заключение

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования HKV-2024 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка HKV-2024;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. Осуществлена программная реализация синтаксического анализатора;
6. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
7. Разработан транслятор кода на язык ассемблера;
8. Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка HKV-2024 включает:

1. 4 типа данных;
2. Поддержка операторов ввода и вывода строки;
3. Наличие 5 арифметических операторов для вычисления выражений
4. Наличие 6 логических операторов для использования в условиях цикла и условной конструкции
5. Поддержка функций; Операторов цикла и условия;
6. Наличие библиотеки стандартных функций языка
7. Структурированная и классифицированная система для обработки ошибок пользователя.

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении трансляторов, а также основные различия и преимущества тех или иных средств трансляции.

Список использованных источников

1. Курс лекций по КПО Наркевич А.С.

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

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

4. Прата, С. Язык программирования C++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 с.

5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с

Приложение А

```
extra numb func rand(numb ra)
extra numb func strlen(stroke s)
extra numb func input()
numb func saul (numb q, numb v, numb z)[
 new numb result;
 state:q<v$
 correctly:[
result = 5 * 4 + 10;
]
wrong:[
result = 5 + z;
]
$
 return result;
];
numb func goodman(numb start , numb end)[
 new numb soprano = 2;
 state: start<end$
 cycle[
 start = start+4;
 soprano = soprano*3;
]$
 return soprano;
]
main
[
 new numb a = 4;
 new numb b =input();
 new numb c = strlen('xutrikk');
 new boolean b1 =false;
 write 'Результат выполнения функции saul:';
 new numb result = saul(a,b,c);
 write result;
```

```
 write 'Результат деления с остатком:';
 new numb ostatok = 10 % 4;
 write ostatok;
 write 'Результат выполнения функции goodman:';
 new numb k = goodman(a,b);
 write k;
 new numb ran = rand(1000);
 write 'Случайное число:';
 write ran;
 return 0;
]
```

Листинг 1 – Исходный код программы на языке HKV-2024

Приложение Б

```
#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::S, 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::S, IN::S, IN::T, IN::F, IN::S, IN::S, IN::S, IN::T, IN::S, IN::S, IN::S, IN::S, IN::S, IN::S, IN::F, IN::S, \
    IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::S, IN::S, IN::S, IN::S, IN::S, 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::F, IN::F, IN::I, IN::F, IN::F, IN::S, IN::F, IN::S, 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::S, IN::S, IN::S, 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::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::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::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \
}
```

Рисунок 1 - Таблица контроля входных символов

```
struct Entry          //стро
{
    char lexema;
    int sn;
    int idxTI;
    int priority;
};

struct LexTable
{
    char scope[MAX_SCOPE];
    int maxsize;
    int size;
    Entry* table;
};
```

Рисунок 2 – Структура таблицы лексем

```
#define FST_CORRECTLY 10,\
      FST::NODE(1, FST::RELATION('c', 1)),\
      FST::NODE(1, FST::RELATION('o', 2)),\
      FST::NODE(1, FST::RELATION('r', 3)),\
      FST::NODE(1, FST::RELATION('r', 4)),\
      FST::NODE(1, FST::RELATION('e', 5)),\
      FST::NODE(1, FST::RELATION('c', 6)),\
      FST::NODE(1, FST::RELATION('t', 7)),\
      FST::NODE(1, FST::RELATION('l', 8)),\
      FST::NODE(1, FST::RELATION('y', 9)),\
       FST::NODE()
```

Листинг 1 – Пример конечного автомата

```
struct Entry
{
    int         idxfirstLE;
    char        id[ID_MAXSIZE];
    IDDATATYPE  iddatatype=UNDEF;
    IDTYPE      idtype;
    char        scope[10]="";
    int         numberOfParam;
    union {
        int vint;
        struct
        {
            char len;
            char str[TI_STR_MAXSIZE - 1];
        }vstr;
    }value;
};
struct IdTable
{
    int maxsize;
    int size;
    Entry* table;
};
```

Рисунок 3 – Структура таблицы идентификаторов

```
ERROR_ENTRY(120,"[LEX]# Превышен максимльный размер таблицы лексем"),
ERROR_ENTRY(121,"[LEX]# Таблица лексем переполнена"),
ERROR_ENTRY_NODEF(122), ERROR_ENTRY_NODEF(123), ERROR_ENTRY_NODEF(124),
ERROR_ENTRY_NODEF(125), ERROR_ENTRY_NODEF(126), ERROR_ENTRY_NODEF(127), ERROR_ENTRY_NODEF(12
ERROR_ENTRY_NODEF(129),  ERROR_ENTRY_NODEF10(130), ERROR_ENTRY_NODEF10(140), ERROR_ENTRY_NOD
ERROR_ENTRY(160,"[LEX]# Превышено максимальное количество строк в таблице идентификаторов"),
ERROR_ENTRY(161,"[LEX]# Таблица идентификаторов переполнена"),
ERROR_ENTRY_NODEF(162),
ERROR_ENTRY(163,"[LEX]# Не удалось распознать цепочку"),
ERROR_ENTRY(164,"[LEX]# Размер литерала больше максимально допустимого"),
ERROR_ENTRY(165,"[LEX]# Отсутствует точка входа main"),
```

Рисунок 4 – Сообщения об ошибках стадии лексического анализа

№	Scope	Идентификатор	Тип данных	Тип идентификатора	Индекс в ТЛ	Значение
0000	rand	rand	integer	функция	3	-
0001	rand	ra	integer	параметр	6	0
0002	strlen	strlen	integer	функция	11	-
0003	strlen	s	string	параметр	14	""
0004	input	input	integer	функция	19	-
0005	saul	saul	integer	функция	24	-
0006	saul	q	integer	параметр	27	0
0007	saul	v	integer	параметр	30	0
0008	saul	z	integer	параметр	33	0
0009	saul	result	integer	переменная	38	0
0010	saul	L0	integer	литерал	52	5
0011	saul	L1	integer	литерал	54	4
0012	saul	L2	integer	литерал	56	10
0013	goodman	goodman	integer	функция	76	-
0014	goodman	start	integer	параметр	79	0
0015	goodman	end	integer	параметр	82	0
0016	goodman	soprano	integer	переменная	87	0
0017	goodman	L3	integer	литерал	90	2
0018	goodman	L4	integer	литерал	110	3
0019	main	a	integer	переменная	121	0
0020	main	b	integer	переменная	127	0
0021	main	c	integer	переменная	135	0
0022	main	L5	string	литерал	140	'xutrikk'
0023	main	b1	Boolean	переменная	144	-
0024	main	L6	Boolean	литерал	147	false
0025	main	L7	string	литерал	150	'Результат выполнения
0027	main	L8	string	литерал	169	'Результат деления с о
0028	main	ostatok	integer	переменная	172	0
0029	main	L9	string	литерал	183	'Результат выполнения
0030	main	k	integer	переменная	186	0
0031	main	ran	integer	переменная	200	0
0032	main	L10	integer	литерал	205	1000
0033	main	L11	string	литерал	209	'Случайное число:'

Листинг 2 – Таблица идентификаторов контрольного примера

```
-------------------------Таблица лексем-----------------------
0000 | etfi(ti)                                          | 000 - 007 |
0001 | etfi(ti)                                          | 007 - 015 |
0002 | etfi()                                            | 015 - 021 |
0003 | tfi(ti,ti,ti)[                                    | 021 - 035 |
0004 | nti;                                              | 035 - 039 |
0005 | ?:i<i$                                            | 039 - 045 |
0006 | c:[                                               | 045 - 048 |
0007 | i=l*l+l;                                          | 048 - 056 |
0008 | ]                                                 | 056 - 057 |
0009 | w:[                                               | 057 - 060 |
0010 | i=l+i;                                            | 060 - 066 |
0011 | ]                                                 | 066 - 067 |
0012 | $                                                 | 067 - 068 |
0013 | ri;                                               | 068 - 071 |
0014 | ];                                                | 071 - 073 |
0015 | tfi(ti,ti)[                                       | 073 - 084 |
0016 | nti=l;                                            | 084 - 090 |
0017 | ?:i<i$                                            | 090 - 096 |
0018 | v[                                                | 096 - 098 |
0019 | i=i+l;                                            | 098 - 104 |
0020 | i=i*l;                                            | 104 - 110 |
0021 | ]$                                                | 110 - 112 |
0022 | ri;                                               | 112 - 115 |
0023 | ]                                                 | 115 - 116 |
0024 | m                                                 | 116 - 117 |
0025 | [                                                 | 117 - 118 |
0026 | nti=l;                                            | 118 - 124 |
0027 | nti=i();                                          | 124 - 132 |
0028 | nti=i(l);                                         | 132 - 141 |
0029 | nti=l;                                            | 141 - 147 |
0030 | pl;                                               | 147 - 150 |
0031 | nti=i(i,i,i);                                     | 150 - 163 |
0032 | pi;                                               | 163 - 166 |
0033 | pl;                                               | 166 - 169 |
0034 | nti=l%l;                                          | 169 - 177 |
0035 | pi;                                               | 177 - 180 |
0036 | pl;                                               | 180 - 183 |
0037 | nti=i(i,i);                                       | 183 - 194 |
0038 | pi;                                               | 194 - 197 |
0039 | nti=i(l);                                         | 197 - 206 |
0040 | pl;                                               | 206 - 209 |
0041 | pi;                                               | 209 - 212 |
0042 | rl;                                               | 212 - 215 |
0043 | ------------------------------------------------------------
```

Листинг 3 – Таблица лексем контрольного примера

Приложение В

```
      Greibach greibach(NS('S'), TS('$'),
            15,
            Rule(NS('Z'), GRB_ERROR_SERIES + 1,                //неверная структура
функции 1
                  2,
                  Rule::Chain(5, TS('f'), TS('i'), TS('('), NS('G'), TS(')')),
                  Rule::Chain(4, TS('f'), TS('i'), TS('('), TS(')'))
            ),
            Rule(NS('S'), GRB_ERROR_SERIES + 0,                //неверная структура
программы 1
                  5,
                  Rule::Chain(7, TS('t'), NS('Z'), TS('['), NS('N'), TS(']'), TS(';'),
NS('S')),
                  Rule::Chain(6, TS('t'), NS('Z'), TS('['), NS('N'), TS(']'), NS('S')),
                  Rule::Chain(4, TS('m'), TS('['), NS('N'), TS(']')),
                  Rule::Chain(3, TS('t'), NS('Z'), NS('S')),
                  Rule::Chain(4, TS('e'), TS('t'), NS('Z'), NS('S'))
            ),
            Rule(NS('G'), GRB_ERROR_SERIES + 2,          //ошибка в параметрах функции
1
                  6,
                  Rule::Chain(1, TS('i')),
                  Rule::Chain(1, TS('l')),
                  Rule::Chain(2, TS('t'), TS('i')),
                  Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('G')),
                  Rule::Chain(3, TS('i'), TS(','), NS('G')),
                  Rule::Chain(3, TS('l'), TS(','), NS('G'))
            ),
            Rule(NS('V'), GRB_ERROR_SERIES + 3,          //ошибка в арифметической
операции 1
                  4,
                  Rule::Chain(1, TS('+')),
                  Rule::Chain(1, TS('-')),
                  Rule::Chain(1, TS('*')),
```

```
                Rule::Chain(1, TS('/'))
        ),
        Rule(NS('O'), GRB_ERROR_SERIES + 4,                //логические операции 1
                8,
                Rule::Chain(1, TS('|')),
                Rule::Chain(1, TS('&')),
                Rule::Chain(1, TS('!')),
                Rule::Chain(1, TS('%')),
                Rule::Chain(1, TS('>')),
                Rule::Chain(1, TS('<')),
                Rule::Chain(1, TS('}')),
                Rule::Chain(1, TS('{'))
        ),
        Rule(NS('Q'), GRB_ERROR_SERIES + 5,                //ошибка в параметрах
функции статической библиотеки 1
                2,
                Rule::Chain(3, TS('('), NS('M'), TS(')')),
                Rule::Chain(5, TS('('), NS('M'), TS(')'), NS('V'), NS('E'))
        ),
        Rule(NS('L'), GRB_ERROR_SERIES + 6, //ошибка в вызове стандартной функции 1
                3,
                Rule::Chain(2, TS('R'), NS('Q')),
                Rule::Chain(2, TS('S'), NS('Q')),
                Rule::Chain(2, TS('U'), NS('Q'))
        ),
        Rule(NS('D'), GRB_ERROR_SERIES + 7,                //инициализация
переменной 1
                1,
                Rule::Chain(2, TS('t'), NS('W'))
        ),
        Rule(NS('W'), GRB_ERROR_SERIES + 8,                //присвоение значения 1
                5,
                Rule::Chain(3, TS('i'), TS('='), NS('M')),
```

```
                Rule::Chain(3, TS('i'), TS('='), NS('E')),
                Rule::Chain(3, TS('i'), TS('='), NS('L')),
                Rule::Chain(5, TS('i'), TS('='), NS('E'), NS('V'), NS('E')),
                Rule::Chain(4, TS('i'), TS('='), TS('n'), NS('M'))
        ),

        Rule(NS('E'), GRB_ERROR_SERIES + 9,          //ошибка в выражении 1
                17,
                Rule::Chain(1, TS('i')),
                Rule::Chain(1, TS('l')),
                Rule::Chain(3, TS('i'), NS('V'), NS('E')),
                Rule::Chain(2, TS('l'), NS('V')),
                Rule::Chain(3, TS('l'), NS('V'), NS('E')),
                Rule::Chain(3, TS('i'), NS('O'), NS('E')),
                Rule::Chain(3, TS('l'), NS('O'), NS('E')),
                Rule::Chain(2, NS('S'), NS('Q')),
                Rule::Chain(2, NS('U'), NS('Q')),
                Rule::Chain(2, NS('R'), NS('Q')),
                Rule::Chain(2, NS('V'), NS('E')),
                Rule::Chain(4, TS('i'), TS('('), NS('G'), TS(')')),
                Rule::Chain(6, TS('i'), TS('('), NS('G'), TS(')'), NS('V'),
NS('E')),
                Rule::Chain(3, TS('i'), TS('('), TS(')')),
                Rule::Chain(5, TS('i'), TS('('), TS(')'), NS('V'), NS('E')),
                Rule::Chain(3, TS('('), NS('E'), TS(')')),
                Rule::Chain(5, TS('('), NS('E'), TS(')'), NS('V'), NS('E'))
        ),
        Rule(NS('M'), GRB_ERROR_SERIES + 10,              //ожидается
идентификатор или литерал 1
                2,
                Rule::Chain(1, TS('l')),
                Rule::Chain(1, TS('i'))
        ),Rule(NS('K'), GRB_ERROR_SERIES + 11,//ошибка в структуре условного
выражения 1
```

```
      3,                Rule::Chain(5, TS(':'), NS('E'), TS('$'), NS('A'), TS('$')),
                  Rule::Chain(4, TS(':'), NS('E'), TS('$'), TS('$')),
                  Rule::Chain(5, TS(':'), NS('E'), TS('$'), TS('$'), NS('N'))
            ),
            Rule(NS('A'), GRB_ERROR_SERIES + 12,              //Ошибка построения
условного выражения 1
                  3,
                  Rule::Chain(3, TS('c'), TS(':'), NS('Y')),
                  Rule::Chain(3, TS('w'), TS(':'), NS('Y')),
                  Rule::Chain(2, TS('v'), NS('Y'))
            ),
            Rule(NS('Y'), GRB_ERROR_SERIES + 13,  //ошибка тела выражения/цикла 1
                  2,
                  Rule::Chain(4, TS('['), NS('N'), TS(']'), NS('A')),
                  Rule::Chain(3, TS('['), NS('N'), TS(']'))
            ),
            Rule(NS('N'), GRB_ERROR_SERIES + 14,        //ошибка в объявлении переменной
1
                  12,
                  Rule::Chain(4, TS('n'), TS('t'), TS('i'), TS(';')),
                  Rule::Chain(5, TS('n'), TS('t'), TS('i'), TS(';'), NS('N')),
                  Rule::Chain(4, TS('n'), NS('D'), TS(';'), NS('N')),
                  Rule::Chain(3, TS('n'), NS('D'), TS(';')),
                  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),
                  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),
                  Rule::Chain(3, TS('p'), NS('E'), TS(';')),
                  Rule::Chain(4, TS('p'), NS('E'), TS(';'), NS('N')),
                  Rule::Chain(3, TS('?'), NS('K'), NS('N')),
                  Rule::Chain(2, TS('?'), NS('K')),
                  Rule::Chain(3, TS('r'), NS('M'), TS(';')),
                  Rule::Chain(4, NS('N'), TS(';'), TS('r'), NS('i'))
            )
      );}
```

Листнинг 1 – Грамматика языка HKV-2024

```
struct Mfst // магазинный автомат
	{
		enum RC_STEP //код возврата 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 походящая цепочка правила (ошибка в
исходном коде)
			NS_ERROR,// неизвесный нетерминальный символ грамматики
			TS_OK,// тек. символ ленты == вершине стека, продвинулась лента, pop
стека
			TS_NOK,// тек. символ ленты != вершине стека, восстановленно
состояние
			LENTA_END,// теущая 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, RC_STEP prc_step, short pnrule,
short pnrule_chain);
		} 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
		LT::LexTable lexTable;
		MFSTSTSTACK st;// стек автомата
		vector<MfstState> storestate;// стек для сохранения состояний

		Mfst();
		Mfst(LT::LexTable& plexTable, GRB::Greibach pgrebach, wchar_t parsfile[]);

		char* getCSt(char* buf);//получить содержиое стека
		char* getCLenta(char* buf, short pos, short n = 25);//лента: n символов,
начиная с pos
		char* getDiagnosis(short n, char* buf);//получить n-ую строку диагностики
или '\0'

		bool savestate();//сохранить состояние автомата
		bool resetstate();//восстановить состояние автомата
		bool push_chain(GRB::Rule::Chain chain);

		RC_STEP step();//выполнить шаг автомата
		bool start();//запустить автомат
		bool savedDiagnosis(RC_STEP prc_step);

		void printRules();//вывести последовательность правил

		struct Deducation
		{
```

```
                short size;
                short* nrules;
                short* nrulechains;

                Deducation()
                {
                        this->size = 0;
                        this->nrules = 0;
                        this->nrulechains = 0;
                }
        } deducation;

        bool savededucation();

        ofstream* pars;

};
```

Листнинг 2 – Структура магазинного автомата

```
Шаг : Правило              Входная лента                  Стек
0   : S->etZS              etfi(ti)etfi(ti)tfi(ti,ti      S$
0   : SAVESTATE:           1
0   :                      etfi(ti)etfi(ti)tfi(ti,ti      etZS$
1   :                      tfi(ti)etfi(ti)tfi(ti,ti,      tZS$
2   :                      fi(ti)etfi(ti)tfi(ti,ti,t      ZS$
Шаг : Правило              Входная лента                  Стек
3   : Z->fi(G)             fi(ti)etfi(ti)tfi(ti,ti,t      ZS$
3   : SAVESTATE:           2
3   :                      fi(ti)etfi(ti)tfi(ti,ti,t      fi(G)S$
4   :                      i(ti)etfi(ti)tfi(ti,ti,ti      i(G)S$
Шаг : Правило             Входная лента                   Стек
3294: M->l                l;]                             M;]$
3294: SAVESTATE:          120
3294:                     l;]                             l;]$
3295:                     ;]                              ;]$
3296:                     ]                               ]$
3297:                                                     $
3298: LENTA_END
3299: ------>LENTA_END
217 всего строк, синтаксический анализ выполнен без ошибок
0   : S->etZS
2   : Z->fi(G)
5   : G->ti
8   : S->etZS
10  : Z->fi(G)
```

Листинг 3 – Разбор исходного кода синтаксическим анализатором

```
struct Greibach
{
    short size;
    GRBALPHABET startN;
    GRBALPHABET stbottomT;
    Rule* rules;

    Greibach()
    {
        this->size = 0;
        this->startN = 0;
        this->stbottomT = 0;
        this->rules = 0;
    };
    Greibach(GRBALPHABET pstartN, GRBALPHABET pstbottomT, short psize, Rule r, ...);

    short getRule(GRBALPHABET pnn, Rule& prule);
    Rule getRule(short n);
};

Greibach getGreibach();
```

Рисунок 1 – Структура грамматики Грейбах

```
ERROR_ENTRY(600,"[SIN]# Неверная структура программы"),
ERROR_ENTRY(601,"[SIN]# Ошибка в объявлении переменной"),
ERROR_ENTRY(602,"[SIN]# Ошибка в выражении"),
ERROR_ENTRY(603,"[SIN]# Ошибка в параметрах функции"),
ERROR_ENTRY(604,"[SIN]# Ошибка в параметрах вызываемой функции"),
ERROR_ENTRY(606,"[SIN]# Ошибка в теле функции"),
ERROR_ENTRY(607,"[SIN]# Ошибка при констуировании условного выражения"),
ERROR_ENTRY(608,"[SIN]# Ошибка в теле условного выражения"),
ERROR_ENTRY(609,"[SIN]# Не найден список параметров функции"),
ERROR_ENTRY(610,"[SIN]# Ошибка в вызове функции"),
ERROR_ENTRY(611,"[SIN]# Ошибка в арифметической операции"),
ERROR_ENTRY(612,"[SIN]# Ожидаются только лексемы и идентификаторы"),
```

Рисунок 2 – Сообщения об ошибках стадии синтаксического анализа

Приложение Г

```
ERROR_ENTRY(301,"[SEM]# В объявлении не указан тип идентификатора"),
ERROR_ENTRY(302,"[SEM]# В объявлении отсутствует ключевое слово new"),
ERROR_ENTRY(303,"[SEM]# Возвращаемый функцией тип не соответствует типу функции"),
ERROR_ENTRY(304,"[SEM]# Ошибка в условии условного выражения"),
ERROR_ENTRY(305,"[SEM]# Типы данных в выражении не совпадают"),
ERROR_ENTRY(306,"[SEM]# Деление на ноль"),
ERROR_ENTRY(307,"[SEM]# Необъявленный идентификатор"),
ERROR_ENTRY(308,"[SEM]# Обнаружено несколько точек входа в main"),
ERROR_ENTRY(309,"[SEM]# Обнаружен \' . Возможно, не закрыт строковый литерал"),
ERROR_ENTRY(310,"[SEM]# В Symbol типе не может быть более 1 символа"),
ERROR_ENTRY(311,"[SEM]# Повторное объявление идентификатора"),
ERROR_ENTRY(312,"[SEM]# Ошибка в вызове Стандартной функции "),
ERROR_ENTRY(313,"[SEM]# Эта операция не применима к типу symbol/stroke"),
ERROR_ENTRY(314,"[SEM]# В стандартную функцию передан необъявленный идентификатор"),
ERROR_ENTRY(315,"[SEM]# Ожидается тип stroke/symbol"),
ERROR_ENTRY(316,"[SEM]# Тип возвращаемого функцией значения не соответствует типу идентификатора"),
ERROR_ENTRY(317,"[SEM]# Ошибка в объявление идентификатора (указан неправильный тип)"),
ERROR_ENTRY(318,"[SEM]# Ожидается ;"),
```

Рисунок 1 – Сообщения об ошибках стадии семантического анализа

```
bool PolishNotation(int i, Lex::Tables& table) {
        LT::LexTable lex = table.lextable;
        stack<LT::Entry> st;
        queue<LT::Entry> q;
        LT::Entry temp;
        temp.lexema = ' ';
        temp.sn = -1;
        temp.idxTI = -1;
        int pos;
        LT::Entry func;
        func.lexema = '@';
        int countOfLex = 0;
        int position = i;
        bool funcFlag = false;
        int param = 0;
        int hesises = 0;
        int comma = 0;
        bool findLibFunc = false;

        for (i; lex.table[i - 1].lexema != LEX_SEMICOLON; i++, countOfLex++)
        {
```

```
switch (lex.table[i].lexema)
                    {
                    case LEX_ID: {
                          if (funcFlag) {
                                param++;
                                q.push(lex.table[i]);
                          }
                          else if (table.idtable.table[lex.table[i].idxTI].idtype !=
IT::F) {
                                q.push(lex.table[i]);
                          }
                          else if (table.lextable.table[i - 2].lexema == LEX_PRINT) {
                                countOfLex += 2;
                                position -= 2;
                                q.push(lex.table[i - 2]);
                          }
                          continue;}
                    case LEX_LITERAL: {
                          if (funcFlag) {
                                param++;
                                q.push(lex.table[i]);
                          }
                          else if (table.idtable.table[lex.table[i].idxTI].idtype !=
IT::F) {
                                q.push(lex.table[i]);
                          }
                          else if (table.lextable.table[i - 2].lexema == LEX_PRINT) {
                                countOfLex += 2;
                                position -= 2;
                                q.push(lex.table[i - 2]);
                          }
                          continue;}
                    case LEX_COMMA: {
continue;
                    }
```

```
                    case LEX_LEFTHESIS: {
                          hesises++;
                          if (lex.table[i - 1].lexema == LEX_ID) {
                                if (table.idtable.table[lex.table[i - 1].idxTI].idtype
== IT::F) {
                                      pos = i - 1;
                                      funcFlag = true;
                                }
                          }
                          if (funcFlag) {
                                func.idxTI = lex.table[i - 1].idxTI;
                                func.sn = lex.table[i - 1].sn;

                          }
                          else
                                st.push(lex.table[i]);
                          if (lex.table[i - 2].lexema == LEX_PRINT) {
                                countOfLex += 2;
                                position -= 2;
                                q.push(lex.table[i - 2]);
                                continue;
                          }
                          continue;
                    }
                    case LEX_RIGHTHESIS: {
                          hesises++;
                          if (funcFlag) {
                                if (param > MAX_NUMBER_OF_PARAM) {
                                      throw ERROR_THROW_IN(331, func.sn, pos);
                                }
                                q.push(func);
char buf[10];
                                itoa(param, buf, 10);
                                func.lexema = buf[0];
                                q.push(func);
```

```
                                funcFlag = false;}
                        else if (findLibFunc) {
                                q.push(func);
                                findLibFunc = false;
                                st.pop();
                                continue;
                        }
                        else {
                                while (st.top().lexema != LEX_LEFTHESIS) {
                                        q.push(st.top());
                                        st.pop();
                                        if (st.empty())
                                                return false;
                                }
                                st.pop();
                        }
                        continue;
                }
                case LEX_PLUS:
                case LEX_MINUS:
                case LEX_DIRSLASH:
                case LEX_STAR:
                case LEX_REMAINDER:
                {
                        while (!st.empty() && st.top().priority <=
lex.table[i].priority && st.top().priority > 1)
                        {
                                q.push(st.top());
                                st.pop();}
char buf[10];
                                itoa(param, buf, 10);
                                func.lexema = buf[0];
                                q.push(func);
                                funcFlag = false;
                        }
```

```
                        else if (findLibFunc) {
                                q.push(func);
                                findLibFunc = false;
                                st.pop();
                                continue;
                        }
                        else {
                                while (st.top().lexema != LEX_LEFTHESIS) {
                                        q.push(st.top());
                                        st.pop();
                                        if (st.empty())
                                                return false;
                                }
                                st.pop();
                        }
                        continue;}
                case LEX_PLUS:
                case LEX_MINUS:
                case LEX_DIRSLASH:
                case LEX_STAR:
                case LEX_REMAINDER:
                {
                        while (!st.empty() && st.top().priority <= lex.table[i].priority
&& st.top().priority > 1)
                        {
                                q.push(st.top());
                                st.pop();}
st.push(lex.table[i]);
                        continue;
                }
                case LEX_SEMICOLON: {
```

```
                        temp.lexema = lex.table[i].lexema;
                        temp.sn = lex.table[i].sn;
                        temp.idxTI = TI_NULLIDX;
                        continue;
                    }
                    }
                }
                while (!st.empty()) {
                    if (st.top().lexema == LEX_LEFTHESIS || st.top().lexema ==
LEX_RIGHTHESIS)
                        return false;
                    q.push(st.top());
                    st.pop();
                }
                q.push(temp);
                int size = q.size();
                int t = countOfLex - size;
                while (countOfLex != 0) {
                    if (!q.empty()) {
                        lex.table[position++] = q.front();
                        q.pop();
                    }
                    else {
                        countOfLex--;
                    }}
                for (int i = 0; i < position; i++)
                {
                    if (lex.table[i].lexema == LEX_PLUS || lex.table[i].lexema == LEX_MINUS
|| lex.table[i].lexema == LEX_STAR || lex.table[i].lexema == LEX_DIRSLASH ||
lex.table[i].lexema == LEX_LITERAL) {
                        table.idtable.table[lex.table[i].idxTI].idxfirstLE = i;
                    }
                }
```

```
            for (int i = position + t; i < table.lextable.size; i++)
            {
                  lex.table[position++] = lex.table[i];
            }
            lex.size = lex.size - t;
            table.lextable = lex;
            return true;
      }
}
```

Листинг 1 – Программная реализация механизма преобразования в Обратную польскую запись

```
-------------------------Таблица лексем-----------------------
0000 | etfi(ti)                                                | 000 - 007 |
0001 | etfi(ti)                                                | 007 - 015 |
0002 | etfi()                                                  | 015 - 021 |
0003 | tfi(ti,ti,ti)[                                          | 021 - 035 |
0004 | nti;                                                    | 035 - 039 |
0005 | ?:i<i$                                                  | 039 - 045 |
0006 | c:[                                                     | 045 - 048 |
0007 | i=ll*l+;                                                | 048 - 056 |
0008 | ]                                                       | 056 - 057 |
0009 | w:[                                                     | 057 - 060 |
0010 | i=li+;                                                  | 060 - 066 |
0011 | ]                                                       | 066 - 067 |
0012 | $                                                       | 067 - 068 |
0013 | ri;                                                     | 068 - 071 |
0014 | ];                                                      | 071 - 073 |
0015 | tfi(ti,ti)[                                             | 073 - 084 |
0016 | nti=l;                                                  | 084 - 090 |
0017 | ?:i<i$                                                  | 090 - 096 |
0018 | v[                                                      | 096 - 098 |
0019 | i=il+;                                                  | 098 - 104 |
0020 | i=il*;                                                  | 104 - 110 |
0021 | ]$                                                      | 110 - 112 |
0022 | ri;                                                     | 112 - 115 |
0023 | ]                                                       | 115 - 116 |
0024 | m                                                       | 116 - 117 |
0025 | [                                                       | 117 - 118 |
0026 | nti=l;                                                  | 118 - 124 |
0027 | nti=@0;                                                 | 124 - 131 |
0028 | nti=l@1;                                                | 131 - 139 |
0029 | nti=l;                                                  | 139 - 145 |
0030 | pl;                                                     | 145 - 148 |
0031 | nti=iii@3;                                              | 148 - 158 |
0032 | pi;                                                     | 158 - 161 |
0033 | pl;                                                     | 161 - 164 |
0034 | nti=ll%;                                                | 164 - 172 |
0035 | pi;                                                     | 172 - 175 |
0036 | pl;                                                     | 175 - 178 |
0037 | nti=ii@2;                                               | 178 - 187 |
0038 | pi;                                                     | 187 - 190 |
0039 | nti=l@1;                                                | 190 - 198 |
0040 | pl;                                                     | 198 - 201 |
0041 | pi;                                                     | 201 - 204 |
0042 | rl;                                                     | 204 - 207 |
0043 | ------------------------------------------------------------------
```

Листинг 2 – измененная таблица лексем после преобразования выражений

Приложение Д

```
.586
.model flat, stdcall
includelib libucrt.lib
includelib kernel32.lib
includelib ../Debug/GenLib.lib
ExitProcess        PROTO:DWORD

Remainder          PROTO : DWORD, :DWORD
PrintStroke        PROTO : DWORD
PrintBoolean       PROTO : DWORD
PrintNumb          PROTO : DWORD
.stack 4096
.const
            L0 SDWORD 5
            L1 SDWORD 4
            L2 SDWORD 10
            L3 SDWORD 2
            L4 SDWORD 3
            L5 byte 'xutrikk', 0
            TRUE equ 1
            FALSE equ 0
            L6 word 0
            L7 byte 'Результат выполнения
функции saul:', 0
            L8 byte 'Результат деления с
остатком:', 0
            L9 byte 'Результат выполнения
функции goodman:', 0
            L10 SDWORD 1000
            L11 byte 'Случайное число:',
0
.data
            saulresult sdword 0
            goodmansoprano sdword 0
            maina sdword 0
      mainb sdword 0
            mainc sdword 0
            mainb1 word ?
            mainresult sdword 0
            mainostatok sdword 0
            maink sdword 0
            mainran sdword 0
.code
Rand PROTO   : DWORD
Strlen PROTO : DWORD
Input PROTO

;----------- saul ------------

saul PROC,
      saulq : sdword, saulv : sdword,
saulz : sdword
push ebx
push edx

mov edx, saulq
cmp edx, saulv

jl right1
jg wrong1
      right1:
push L0
push L1
pop ebx
pop eax
imul eax, ebx
push eax
push L2
```

```
pop ebx
pop eax
add eax, ebx
push eax
pop ebx
mov saulresult, ebx

jmp next1
      wrong1:
push L0
push saulz
pop ebx
pop eax
add eax, ebx
push eax
pop ebx
mov saulresult, ebx

next1:

pop ebx
pop edx
mov eax, saulresult
ret
saul ENDP
;------------------------------
;----------- goodman ------------

goodman PROC,
      goodmanstart : sdword, goodmanend :
sdword
push ebx
push edx
```

```
push L3
pop ebx
mov goodmansoprano, ebx

mov edx, goodmanstart
cmp edx, goodmanend
jl repeat2
jmp repeatnext2
repeat2:

push goodmanstart
push L1
pop ebx
pop eax
add eax, ebx
push eax
pop ebx
mov goodmanstart, ebx

push goodmansoprano
push L4
pop ebx
pop eax
imul eax, ebx
push eax
pop ebx
mov goodmansoprano, ebx

mov edx, goodmanstart
cmp edx, goodmanend

jl repeat2
repeatnext2:
```

```
pop ebx
pop edx
mov eax, goodmansoprano
ret
goodman ENDP
;-----------------------------
;----------- MAIN ------------
main PROC
push L1
pop ebx
mov maina, ebx
call input
push eax
mov mainb, eax
push offset L5
call strlen
push eax
mov mainc, eax
mov cx, L6
mov mainb1, cx
push offset L7
call PrintStroke
push mainc
push mainb
push maina
call saul
push eax
mov mainresult, ea
push mainresult
call PrintNumb
push offset L8
call PrintStroke
push L2
push L1
call Remainder
mov mainostatok, eax
push mainostatok
call PrintNumb
push offset L9
call PrintStroke
push mainb
push maina
call goodman
push eax
mov maink, eax
push maink
call PrintNumb

push L10
call rand
push eax
mov mainran, eax
push offset L11
call PrintStroke
push mainran
call PrintNumb
;-----------------------------
push 0
call ExitProcess
main ENDP
end main
```

Листинг 1 – Результат генерации кода контрольного примера в Ассемблер