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 KDA-2024»

Выполнил студент Хомутов Денис Андреевич
(Ф.И.О. студента)

Руководитель проекта Волчек Дарья Ивановна
(учен. степень, звание, должность, подпись, Ф.И.О.)

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

Консультанты Волчек Дарья Ивановна
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____

Минск 2024

Содержание

Введение

Целью выполнения курсового проекта по дисциплине «Конструирование программного обеспечения» является написание спецификации и разработка компилятора для собственного языка программирования.

Название языка, для которого разрабатывается компилятор, – KDA-2024. Компиляция будет производиться в язык ассемблера.

Этапы разработки компилятора для языка KDA-2024:

- Написание спецификации языка программирования;
- Разработка лексического анализатора;
- Разработка синтаксического анализатора;
- Разработка семантического анализатора;
- Преобразование арифметических выражений;
- Генерация кода;
- Тестирование транслятора.

Информация о каждом этапе разработки компилятора приведена в соответствующих разделах пояснительной записки.

В первом разделе приведена спецификация языка – точное формализованное описание набора правил, определяющих лексику, синтаксис и семантику языка.

Во втором разделе описана структура компилятора.

В третьем разделе описаны принцип работы и этапы разработки лексического анализатора, определены разрешенные символы и ключевые слова языка программирования.

В четвертом разделе описан принцип работы синтаксического анализатора, формальная грамматика определена и приведена в нормальную форму Грейбах для выполнения синтаксического разбора.

В пятом разделе описаны принцип работы и основные функции семантического анализатора.

В шестом разделе описаны выражения, допускаемые языком, форма, принципы построения и вычисления выражений.

В седьмом разделе описан процесс генерации кода.

В восьмом разделе приведены примеры тестирования транслятора.

1 Спецификация языка программирования

1.1 Характеристика языка программирования

Язык KDA-2024 является компилируемым, строго типизированным, процедурным, поддерживающим парадигму структурного программирования.

1.2 Определение алфавита языка программирования

Алфавит языка KDA-2024 основан на кодировке Windows-1251, изображенной на рисунке 1.1.

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

Рисунок 1.1 – Алфавит входных символов языка KDA-2024

На рисунке представлены символы, соответствующие кодировке Windows-1251, включающей кириллицу, латиницу, цифры и специальные знаки. Это позволяет эффективно обрабатывать текстовые данные в русскоязычной среде и совместимых системах.

1.3 Применяемые сепараторы

Символы-сепараторы служат в качестве разделителей кострукций языка во время обработки исходного текста программы с целью разделения на токены.

Сепараторы, применяемые в языке KDA-2024, приведены в таблице 1.1.

Таблица 1.1 – Применяемые сепараторы

Разделители	Назначение
‘пробел’, ‘табуляция’, ‘переход на новую строку’	Разделяют входные лексемы
+, -, *, /, %	Арифметические операторы. Используются в арифметических операциях
=	Оператор присваивания. Используется для присваивания значения переменной
<, >, <=, >=, !=, ==	Условные операторы. Используются для сравнения переменных и литералов
( )	Блок параметров функции, так же указывает приоритет в арифметических операциях
{ }	Ограничивают программные конструкции

В таблице представлены разнообразные разделители, такие как пробел, табуляция и переход на новую строку, для отделения лексем. Среди операторов выделяются арифметические (+, -, *, /, %), оператор присваивания (=), а также условные операторы (<, >, <=, >=, !=, ==), обеспечивающие выполнение вычислений, присваиваний и логических сравнений.

1.4 Применяемые кодировки

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

1.5 Типы данных

В языке KDA-2024 поддерживается 3 типа данных: целочисленный, строковый и логический. Подробная описание типов данных приведено в таблице 1.2.

Таблица 1.2 – Типы данных языка KDA-2024

Тип данных	Характеристика
Целочисленный (int)	В памяти занимает 2 байта. Максимальное значение: 32767. Минимальное значение: -32768. Принцип размещения в памяти: Последний бит числа отведен под знак, оставшиеся 15 бит предназначены для хранения значения числа. Значение по умолчанию: 0. В арифметических выражениях и условных конструкциях к целочисленным переменным и литералам применимы все арифметические и условные операции соответственно, поддерживаемые языком KDA-2024.
Строковый (str)	В памяти занимает n + 1 байт, где n – количество символов в строке + символ конца строки. Максимальное количество символов в строке: 255. Принцип размещения в памяти: Каждый символ строки занимает 1 байт. В конце строки располагается NULL символ (признак конца строки). К строковым переменным и литералам операции не применяются.
Логический (bool)	В памяти занимает 1 байт. Может принимать одно из двух значений: true или false. Принцип размещения в памяти: В зависимости от значения 1 бит числа установлен true: 1, false: 0. К булевым переменным и литералам применимы все условные операции, поддерживаемые языком KDA-2024.

В таблице представлены три основных типа данных: целочисленный (int), занимающий 2 байта с диапазоном значений от -32768 до 32767, строковый (str), ограниченный длиной в 255 символов, и логический (bool)**,** хранящий 1 байт для значений true или false. Каждый тип данных имеет свои ограничения и особенности, определяющие их размещение в памяти и применимость операций.

1.6 Преобразование типов данных

Преобразования типов данных в языке программирования KDA-2024 не поддерживаются.

1.7 Идентификаторы

Идентификатор – это имя, используемое для переменных, функций, параметров. Идентификаторы могут состоять как из одного, так и из нескольких символов. Первым символом должна быть маленькая буква латинского алфавита, а за ним могут стоять маленькие буквы латинского алфавита или цифры.
Идентификаторы не могут совпадать с ключевыми словами.

Пример корректных идентификаторов: str1, abc161 и т.п.

Пример некорректных идентификаторов: 14stroka, Stroka1, _stroka1 и т.п.

1.8 Литералы

Литерал – это запись в исходном коде программы, представляющая собой фиксированное значение. В языке программирования KDA-2024 предусмотрены следующие типы литералов: строковый, логический и целочисленный. Целочисленные литералы представлены в 4 системах счисления: двоичная, восьмеричная, десятеричная, шестнадцатиричная.

Описание литералов приведено в таблице 1.3.

Таблица 1.3 – Литералы языка KDA-2024

Тип литерала	Характеристика
Целочисленный	Десятичный: Последовательность десятичных цифр 0..9 с предшествующим знаком минус или без него. Двоичный: Последовательность двоичных цифр 0 и 1 с предшествующим знаком минус или без него, в конце которой стоит символ ‘B’ (признак двоичного целочисленного литерала). Восьмеричный: Последовательность восьмеричных цифр 0..7 с предшествующим знаком минус или без него, в конце которой стоит символ ‘O’ (признак восьмеричного целочисленного литерала). Шестнадцатеричный: Последовательность шестнадцатеричных чисел 0..F с предшествующим знаком минус или без него, в конце которой стоит символ ‘H’ (признак шестнадцатеричного целочисленного литерала). Допустимый диапазон значений: От -32768 до 32767 в десятичной системе исчисления.

Окончание таблицы 1.3

Строковый	Набор, состоящий из символов русского и латинского алфавитов, десятичных цифр и специальных символов, заключенный в двойные кавычки. Допустимый диапазон значений: От 0 до 255 символов.
Логический	Допустимые значения: true или false, где true: логическая единица, false: логический ноль.

Пример корректных литералов: 56, -3, 6D4H, -110B, 43O, true, “string” и т.п.
Примеры некорректных литералов: -7A, ‘stroka’, fals, 9O и т.п.

1.9 Объявление данных

Для объявления переменной используется ключевое слово new, после которого указывается тип переменной и имя идентификатора. new <тип> <имя идентификатора>; new <тип> <имя идентификатора> = <литерал>;

Переменную можно объявить в блоке main, в блоке функции или в условном блоке if-else. Область видимости идентификаторов определяется блоком кода, который заключен в { }.

1.10 Инициализация данных

В языке KDA-2024 присутствует 2 вида инициализации:

- Инициализация в месте объявления new <тип> <имя идентификатора> = <литерал>;
- Инициализация после объявления

<имя идентификатора> = <литерал>;

Так же в языке присутствует инициализация по умолчанию. Переменные целочисленного типа по умолчанию инициализируются значением 0. Переменные строкового типа по умолчанию инициализируются пустой строкой. Переменные логического типа по умолчанию инициализируются значением false.

1.11 Инструкции языка

Инструкции языка KDA-2024 приведены в таблице 1.4.

Таблица 1.4 – Инструкции языка KDA-2024

Инструкция	Форма записи
Объявление переменной	**new** <тип данных> <идентификатор>.
Объявление переменной с явной инициализацией	**new** <тип данных> <идентификатор> = <значение>|<выражение>; Значение – литерал, идентификатор, вызов функции соответствующего типа данных

Объявление функции	<тип данных> **function** <идентификатор> (<тип данных> <идентификатор>, …)
	{
	/ тело функции / **return** <идентификатор/литерал>.
	};

Вызов функции	<идентификатор> (<идентификатор>, ...)
Присвоение значения	<идентификатор> = <значение>;
Вывод данных	**write** <идентификатор/литерал>;

В таблице представлены инструкции языка KDA-2024.

1.12 Операции языка

В языке KDA-2024 существует два типа операций: арифметические и логические.

Наибольшая приоритетность у операций умножения, деления и деления с остатком, затем идут операции сложения и вычитания. Можно задать самый высокий приоритет, поместив операции в скобки.

Все логические операции имеют равный приоритет.

Описание операций языка KDA-2024 приведено в таблице 1.5.

Таблица 1.5 – Операции языка KDA-2024

Тип оператора	Оператор
Арифметические операции	+ – сложение - – вычитание * – умножение / – деление % – остаток от деления

Окончание таблицы 1.5

Логические операции	> – больше < – меньше >= – больше или равно <= – меньше или равно == – проверка на равенство != – проверка на неравенство

В таблице представлены все операции языка KDA-2024.

1.13 Выражения и их вычисления

Выражение языка программирования KDA-2024 представляет собой совокупность переменных, литералов, вызовов функций, знаков операций, скобок, которая может быть вычислена в соответствии с синтаксисом языка.

Правила составления выражений:

- Выражения записываются в одну строку;
- В выражении могут присутствовать только операнды одинакового типа;
- В выражении могут использоваться функции. Как стандартные, так и пользовательские;
- В выражении не могут идти подряд два оператора;
- Допускается использование круглых скобок для смены приоритета операций.

В арифметических выражениях допускаются только операнды целочисленного типа. В выражениях сравнения допускаются операнды булевого и целочисленного типов.

Перед генерацией кода выражения приводятся к ПОЛИЗ для более удобного вычисления на языке ассемблера.

1.14 Конструкции языка

Конструкции языка KDA-2024 приведены в таблице 1.6.

Таблица 1.6 – Конструкции языка KDA-2024

Конструкция	Описание

Главная функция	main {
	…
	};

Окончание таблицы 1.6

Пользовательская функция	<тип возвращаемого значения> function(<тип параметра> <имя параметра>, …)
	{
	…
	return <имя переменной/литерал>;
	};
	Максимальное количество параметров: 8.

Условная конструкция	if(<имя переменной/литерал><условный оператор><имя переменной/литерал>)
	{
	…
	} else {
	...
	}
	(блок else необязателен)

В таблице представлены конструкции языка KDA-2024.

1.15 Область видимости идентификатора

Каждой конструкции языка KDA-2024 соответствует своя область видимости. Причем функции имеют глобальную область видимости.

Глобальные переменные отсутствуют, поэтому объявление переменных вне функций невозможно.

1.16 Семантические проверки

Семантическим анализатором языка KDA-2024 предусмотрены следующие проверки:

- Наличие блока main, точки входа в программу;
- Единственная точка входа в программу;
- Использование идентификаторов до их объявления;
- Переопределение идентификаторов;
- Соответствие параметров, передаваемых в функцию, с параметрами в объявлении функции;
- Соответствие типа возвращаемого значения с типом функции;
- Соответствие типов в выражениях;
- Превышение размера целочисленных и строковых литералов;

−Соответствие операторов типам данных, для работы с которыми они предназначены.

1.17 Распределение оперативной памяти на этапе выполнения

Для запоминания промежуточных результатов в вычислении выражения используется стек. В сегмент констант записываются все литералы языка.

1.18 Стандартная библиотека и её состав

В языке KDA-2024 предусмотрена стандартная библиотека, которая включает в себя набор стандартных функций, а также функций вывода в консоль. Функции, входящие в состав стандартной библиотеки приведены в таблице 1.7.

Таблица 1.7 – Функции стандартной библиотеки языка KDA-2024

Прототип функции	Описание
_pow(int a, int b);	Возводит число a в степень b и возвращает результат.
_abs(int a);	Берет абсолютное значение числа a и возвращает результат.
noutl(int value);	Выводит целочисленный идентификатор или литерал на консоль.
soutl(str value);	Выводит строковый идентификатор или литерал на консоль.
compare (str value1, str value1);	Выводит результат сравнения строк. Если первая строка больше возвращает 2, если строки равны, возвращает 1, если первая строка меньше, возвращает 0.

Стандартная библиотека написана на языке C++, подключается на этапе компоновки. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций.

1.19 Ввод и вывод данных

В языке программирования KDA-2024 ввод данных не поддерживается.

Вывод данных на консоль осуществляется за счет оператора **write**. Использование данного оператора допускается только с идентификаторами или литералами.

Функции, управляющие выводом данных на консоль, реализованы на языке C++. На этапе генерации кода операторы вывода языка KDA-2024 заменяются на встроенные функции, находящиеся в стандартной библиотеке.

1.20 Точка входа

В языке KDA-2024 точкой входа в программу является функция **main.**

1.21 Препроцессор

Препроцессор в языке программирования KDA-2024 не предусмотрен.

1.22 Соглашение о вызовах

В языке KDA-2024 вызов функций происходит по стандартному соглашению о вызовах stdcall. Данное соглашение имеет следующие особенности:

−Все параметры функции передаются через стек;

−Освобождением памяти занимается вызываемый код;

−Параметры в стек заносятся справа налево.

1.23 Объектный код

Язык программирования KDA-2024 транслируется в язык ассемблера.

1.24 Классификация сообщений транслятора

Описание и классификация сообщений компилятора об ошибках приведено в таблице 1.8.

Таблица 1.8 – Описание ошибок транслятора языка KDA-2024

Интервал кодов	Описание
0-9	Системные ошибки.
10-19	Ошибки параметров.
20-29	Ошибки файлов.
110-129	Ошибки лексического анализатора.
130-149	Ошибки семантического анализатора.
600-609	Ошибки синтаксического анализатора.

В таблице представлены ошибки транслятора языка KDA-2024.

1.25 Контрольный пример

Контрольный пример языка KDA-2024 представлен в приложении А.

2 Структура транслятора

2.1Компоненты транслятора их назначение и принципы взаимодействия

Транслятор – это программа преобразующая исходный код на одном языке программирования в исходный код на другом языке.

Схема, поясняющая принцип работы транслятора, изображена на рисунке 2.1.

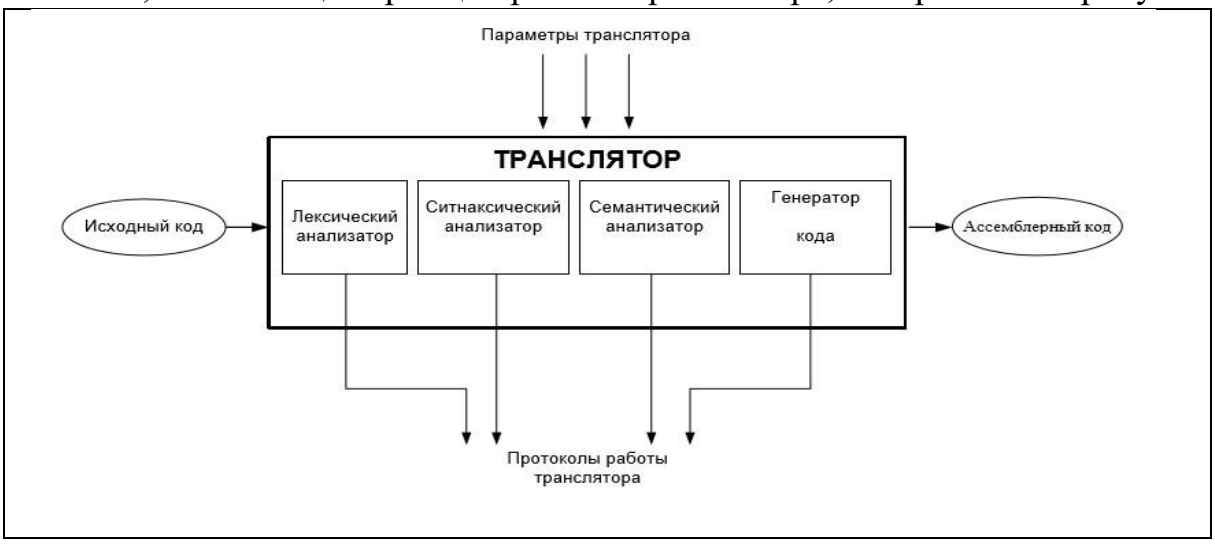

Рисунок 2.1 – Структура транслятора языка KDA-2024

Трансляция исходного кода в язык ассемблера разделена на 4 этапа:

- Лексический анализ
- Синтаксический анализ
- Семантический анализ
- Генерация кода

Этапы выполняются последовательно. У каждого этапа есть входные и выходные данные, которые последовательно передаются следующему компоненту транслятора.

Первой частью трансляции является лексический анализ. На вход лексического анализатора подается исходный код программы. В свою очередь лексический анализатор производит деление исходного кода программы на токены, которые затем идентифицируются и заменяются на лексемы.

На выходе лексического анализатора мы имеем две таблицы: таблицу лексем и таблицу идентификаторов.

Синтаксический анализ является второй частью работы транслятора. Синтаксический анализатор выполняет синтаксический анализ. Входом для синтаксического анализатора является таблица лексем и таблица идентификаторов. Выходом – дерево разбора.

Затем выполняется семантический анализ. Задача семантического анализатора: проверка соблюдения в исходной программе семантических правил

входного языка. Входом для семантического анализатора является таблица идентификаторов, таблица лексем и дерево разбора.

Последним этапом трансляции является генерация кода. На вход генератора подаются таблица лексем и таблица идентификаторов, на основе которых генерируется файл с кодом на языке ассемблера.

2.2 Перечень входных параметров транслятора

Входные параметры необходимы для формирования файлов с результатами работы транслятора. Входные параметры транслятора языка программирования KDA-2024 приведены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка KDA-2024

Ключ и входной параметр	Описание	Значение по умолчанию
-in:<путь к файлу>	Текстовый файл с исходным кодом на языке KDA-2024.	Отсутствует
-out:<путь к файлу>	Выходной файл, содержащий исходный код на языке ассемблера.	Отсутствует
-log:<путь к файлу>	Файл с протоколом работы транслятора.	<имя файла in>.log
-an:<путь к файлу>	Файл с таблицей лексем, таблицей идентификаторов, деревом разбора синтаксического анализатора.	<имя файла>.an.txt

В таблице представлены параметры транслятора KDA-2024.

2.3 Протоколы, формируемые транслятором

Протоколы, формируемые транслятором языка KDA-2024 приведены в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка KDA-2024

Протокол	Описание
Файл, заданный параметром “-log:”	Содержит общую информацию о ходе выполнения трансляции: перечисление входных параметров, количество символов и строк, успех или ошибку по каждому этапу трансляции. В случае возникновения ошибки, в файл будет выведена информация об ошибке.
Файл, заданный параметром “-an:”	Содержит таблицу лексем, итог работы лексического анализа; таблицу идентификаторов, итог работы лексического анализа; дерево разбора, итог работы синтаксического анализатора.

3 Разработка лексического анализатора

3.1 Структура лексического анализатора

Лексический анализатор – это программа, преобразующая исходный текст программы, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Структурная схема лексического анализатора изображена на рисунке 3.1.

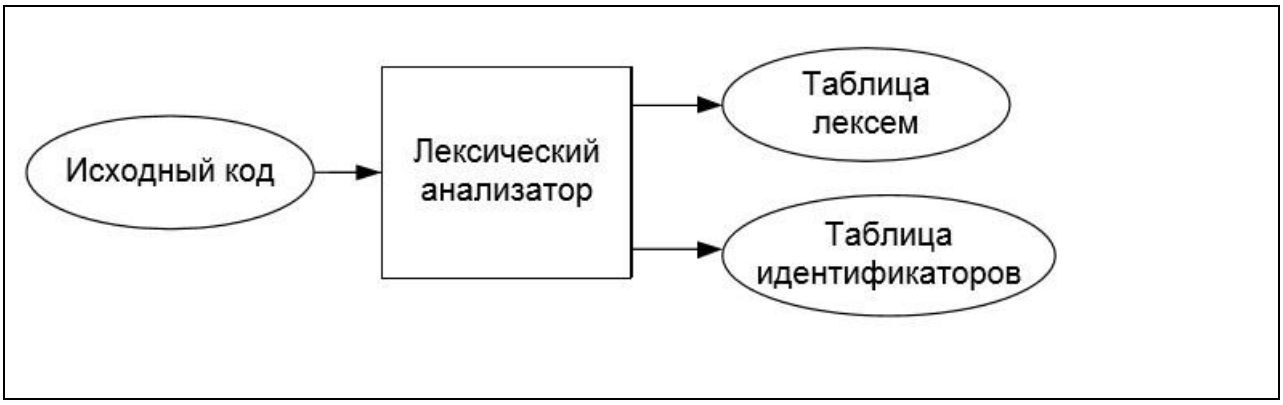

Рисунок 3.1 – Структурная схема лексического анализатора

Лексический анализ в языке KDA-2024 происходит в два этапа:

−Разбиение исходного кода программы на токены.

−Идентификация токенов и последующая их замена на лексемы.

Заполнение таблиц лексем и идентификаторов.

Входные данные: исходный код.

Результат работы: Таблица лексем и таблица идентификаторов.

3.2 Входные и выходные данные лексического анализатора

На вход лексического анализатора поступает исходный код программы на языке KDA-2024. Результатом работы лексического анализатора является таблица лексем и таблица идентификаторов.

3.3 Параметры лексического анализатора

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

3.4 Алгоритм лексического анализа

Алгоритм лексического анализа языка KDA-2024 заключается в следующем:

–Считываем исходный текст программы и делим его на токены, формируя структуру таблицы токенов;

- Слова из таблицы токенов пропускаем через графы, определяя тип лексем;
- Составляем таблицу лексем и идентификаторов.

Программный код, реализующий данный алгоритм, представлен в Листинге 3.1.

```
      struct LEX
    {
         IT::IdTable idtable;
         LT::LexTable lextable;
         LEX();
         LEX(int lexTableSize, int idTableSize);
    };
  // Определяем тип лексемы  char
 LexType(Tokens::Token token);
    // Заполнение таблиц
    LEX FillingInTables(Tokens::TokenTable tokenTable);
      // Поиск id в таблице int SearchID(stack<int>
      areaOfVisibility, string id, IT::IdTable&
      idTable);
       // Поиск id функции в таблице int SearchGlobalFunctionID(int
      globalAreaOfVisibility, string id, IT::IdTable& idTable);
```

Листинг 3.1 – Программный код, реализующий лексический анализ

3.5 Контроль входных символов

Для обеспечения корректности обработки входных данных в языке KDA-2024 была разработана таблица входных символов, которая полностью соответствует кодировке Windows-1251. Эта таблица служит для проверки допустимости каждого символа, используемого в исходном коде. Разрешенные символы в таблице обозначены специальным символом T, что подтверждает их корректность, тогда как запрещенные символы отмечены символом F, указывая на их недопустимость.

Таблица контрольных символов была тщательно разработана, чтобы минимизировать ошибки на этапе ввода данных и гарантировать соблюдение

лексических правил языка. Более подробное представление таблицы контрольных символов языка KDA-2024 можно найти в листинге приложения Б.

Избыточный символ – это символ, отсутствие которого никак не влияет на исходный текст программы. В языке KDA-2024 символ пробела и табуляции являются избыточными символами. Их удаление предусмотрено на этапе разбиения исходного кода программы на токены. Алгоритм удаления избыточных символов пока есть символы для чтения:

−Читаем очередной символ;

−Если символ является пробелом или табуляцией:

−Если идёт запись слова, пробел или табуляция являются символом сепаратором, сигнализирующем о начале или конце записи токена;

−Иначе символ пробела или табуляции пропускаются.

3.6 Перечень ключевых слов

Для удобства изучения языка программирования KDA-2024, в таблице 3.1 представлена исчерпывающая информация о всех ключевых словах, используемых в данном языке, а также символах операций, сепараторах и их соответствующих лексемах. Помимо этого, в таблице даны регулярные выражения, которые подробно описывают структуру и правила формирования указанных элементов.

Таблица 3.1 – Все ключевые слова, сепараторы и т.д. языка KDA-2024

Слово (токен)	Лексема	Описание
int	d	Целочисленный тип.
str	s	Строковый тип.
bool	b	Логический тип.
идентификатор	i	Идентификатор любого типа языка.
литерал	l	Литерал любого типа языка.
if	c	Условный оператор и истинная ветвь условного оператора.
else	e	Ложная ветвь условного оператора.
function	f	Начало объявления функции.
new	n	Объявление переменной.
return	r	Выходи из функции и возврат значения.
write	o	Вывод данных в консоль.
entry	m	Главная функция (точка входа в программу).
pow	p	Стандартная функция, возведения в степень целочисленного литерала, языка.

Окончание таблицы 3.1

abs	a	Стандартная функция, взятия абсолютного значения целочисленного литерала, языка.
, ;	, ;	Разделитель параметров функции. Признак конца инструкции.
{	{	Начало тела функции.
}	}	Конец тела функции.
(	(	Начало перечислений параметров функции, приоритет операций в выражениях.
)	)	Конец перечислений параметров приоритет функции, операций в выражениях.
+	+	Арифметический оператор (сложение).
-	-	Арифметический оператор (вычитание).
*	*	Арифметический оператор (умножение).
/	/	Арифметический оператор(деление).
%	%	Арифметический оператор (остаток от деления).
>	<	Оператор сравнения (больше).
<	>	Оператор сравнения (меньше).
<=	[	Оператор сравнения (меньше или равно).
>=	]	Оператор сравнения (больше или равно).
==	&	Оператор сравнения (равенство).
!=	!	Оператор сравнения (неравенство).
=	=	Арифметический оператор (присваивание).

В листинге 3.4 представлен фрагмент кода функции на языке C++, реализующей алгоритм разбора входной цепочки в соответствии с графами переходов языка KDA-2024.

```
     bool execute(FST& fst)
     {
  short* rstates = new short[fst.nstates];
memset(rstates, 0xff, sizeof(short) * fst.nstates);
short lstring = strlen(fst.string);   bool rc = true;
          for (short i = 0; i < lstring && rc; i++)
          {
               fst.position++;rc = step(fst, rstates);
          }
          delete[] rstates;
       return (rc?(fst.rstates[fst.nstates-1]==lstring):rc);
```

Листинг 3.4 – Функция разбора входной цепочки на языке KDA-2024

В данном листинге представлена функция разбора входной цепочки на языке KDA-2024.

3.7 Основные структуры данных

Основные структуры данных лексического анализатора: таблица токенов, таблица лексем и таблица идентификаторов.

В листинге 3.5 представлена структура токена и таблицы токенов.

```
     //
Структура токена
struct Token
     {
  char
token[258];
 int
length;
int line;
          int linePosition;
     };
     // Структура таблицы токенов
     struct TokenTable
     {
```

Листинг 3.5 – Структура токена и таблицы токенов языка KDA-2024

Структура TokenTable представляет таблицу токенов, где maxsize – число, равное максимальному размеру таблицы, size – текущий размер таблицы, а table – указатель на строку таблицы.

Структура Token представляет строку таблицы TokenTable, где в массив token записывается слово, length – длина слова, line – номер строки в исходном тексте программы, a linePosition – позицию в строке.

Структура LexTable представляет таблицу лексем, где maxsize – число, равное максимальному размеру таблицы, size – текущий размер таблицы, a table – указатель на строку таблицы.

Структура Entry представляет строку таблицы LexTable, где lexeme представляет лексему, sn – номер строки в исходном тексте программы, a idxTI – номер в таблице идентификаторов.

Реализация структуры таблицы лексем представлена в листинге 3.6.

```
        struct Entry
    {
     char
   lexema;
    int
   sn;
               int idxTI;
    };
    struct LexTable
    {
    Int
   maxsize;
```

Листинг 3.6 – Структура таблицы лексем языка KDA-2024

Реализация структуры таблицы идентификаторов представлена в листинге 3.7.

Структура IdTable представляет собой таблицу идентификаторов, где maxsize – число, равное максимальному размеру таблицы, size – текущий размер таблицы, a table – указатель на строку таблицы.

Структура Entry представляет строку таблицы IdTable, где переменная idxfirstLE – индекс первого вхождения в таблицу лексем, id – идентификатор, idDataType – тип данных, idType – тип идентификатора, vshort – целочисленное значение, len – длина строку, str – строка.

```
struct Entry
     {
   int
 idxfirstLE;
 std::string id;
 IDDATATYPE
 idDataType;
 IDTYPE
 idType;
          struct
          {
               int vshort;
               struct
               {
                    int len;
                    std::string str;
               } vstr;
          } value;
     };
     struct IdTable
     {
   int
 maxsize;
  int
 size;
          Entry* table;
      };
```

Листинг 3.7 – Структура таблицы идентификаторов языка KDA-2024

В листинге представлена структура таблицы идентификаторов языка KDA-2024.

3.8 Структура и перечень сообщений лексического анализатора

При возникновении ошибки в лексическом анализаторе формируется ошибка в следующем формате: Номер ошибки, пояснительный текст, строка в исходном тексте, позиция в строке.

Перечень сообщений лексического анализатора приведен в таблице 3.2.

Таблица 3.2 – Перечень сообщений лексического анализатора языка KDA-2024

Номер ошибки	Пояснительный текст
110	Недопустимый символ в исходном файле (-in:)
111	Превышена емкость таблицы лексем

Окончание таблицы 3.2

112	Превышено количество строк в таблице лексем
113	В таблице лексем отсутствует строка с заданным номером
114	Превышена емкость таблицы идентификаторов
115	Превышено количество строк в таблице идентификаторов
116	В таблице идентификаторов отсутствует строка с заданным номером
117	Не удалось определить тип лексемы
118	Превышена емкость таблицы токенов
119	Превышено количество токенов в таблице токенов
120	Ошибка с разбиением исходного текста на токены
121	Ошибка с разбором строкового литерала

В таблице представлена перечень сообщений лексического анализатора языка KDA-2024

3.9 Принцип обработки ошибок

При обнаружении ошибки в исходном коде программы лексический анализатор формирует сообщение об ошибке и выводит его в консоль и записывает в файл с протоколом работы, заданный параметром –log:.

3.10 Контрольный пример

Результат работы лексического анализатора, полученный при выполнении контрольного примера, а именно таблица лексем и таблица идентификаторов, представлен в приложении В.

4 Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

Синтаксический анализ языка KDA-2024 выполняется после завершения работы лексического анализатора. Синтаксический анализатор предназначен для сопоставления последовательности лексем языка KDA-2024 с его формальной грамматикой.

Входными данными являются: таблица лексем и таблица идентификаторов.

Результатом работы является дерево разбора.

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе языка KDA-2024 используется грамматика типа 2 иерархии Хомского (Контекстно-свободная грамматика) G = {N, T, P, S}, где N – конечный алфавит нетерминальных символов, приведенный в первом столбце таблицы 4.1; T – конечный алфавит терминальных символов; – P – конечное множество правил порождения; – S – начальный нетерминал грамматики G.

Контекстно-свободная грамматика G имеет нормальную форму Грейбах, если она не является леворекурсивной и правила P имеют вид:

−A → aα, где a ∈ T, α ∈ N*;

−S → λ, где S ∈ N – начальный символ, если есть такое правило, то S не должен встречаться в правой части правил.

Алгоритм преобразования грамматик в нормальную форму Грейбах:

−Исключить недостижимые символы из грамматики; − Исключить лямбда-правила из грамматики; − Исключить цепные правила.

Перечень и описание терминальных, нетерминальных символов и правил языка приведен в таблице 4.1.

Таблица 4.1 – Описание правил, составляющих грамматику языка KDA-2024

Нетерминальный символ	Цепочки правил	Описание
S	m{N}; m{N};S dfi(){N};S dfi(){N}; bfi(){N};S bfi(){N}; sfi(){N};S sfi(){N};	Правила, порождающие главную функцию main и глобальные функции.

Продолжение таблицы 4.1

Нетерминальный символ	Цепочки правил	Описание
	dfi(F){N}; bfi(F){N};S bfi(F){N}; sfi(F){N};S sfi(F){N};	

N	nTi;N
	nTi=E;N
	nTi;
	nTi=E;
	i=E;
	i=E;N
	oL;N
	oL;
	p(W);N
	p(W);
	a(W);N
	a(W);
	i(W);N
	i(W);
	i();N i();
	cQ{N}N cQ{N}
	cQ{N}e{N}
	cQ{N}e{N}N
	rL;N rL;
		Правила, порождающие конструкции в функциях.

F	di si bi di,F si,F	Правила, порождающие параметры объявления функции.
E	i l (E) i(W) i() p(W) a(W) iM lM	Правила, порождающие выражения.

Продолжение таблицы 4.1

Нетерминальный символ	Цепочки правил	Описание
	(E)M i(W)M i()M p(W)M a(W)M	
W	i l i,W l,W i() i(),W i(W) i(W),W p(W) p(W),W a(W) a(W),W	Правила, порождающие параметры вызываемой функции.
M	+E +EM -E -EM /E /EM *E *EM %E %EM !L &L >L <L	Правила, порождающие операторы.
Q	(L<L) (L>L) (L!L) (L&L)	Правила, порождающие условия в условных конструкциях.
L	L i	Правила, порождающие литерал и идентификатор

Окончание таблицы 4.1

Нетерминальный символ	Цепочки правил	Описание
T	d s b	Правила, порождающие типы данных.

В таблице представлено описание правил, составляющих грамматику языка KDA-2024 .

4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку, описание которой приведено ниже. M = {Q, V, Z, δ, q_0, z_0, F}, где

−Q – множество состояний;

− V – алфавит входных символов;

− Z – специальный алфавит магазинных символов;

− δ – функция переходов автомата;

− $q_0 \in Q$ – начальное состояние автомата;

−$z_0 \in Z$ – начальное состояние магазина (маркер дна); − $F \subseteq Q$ – Множество конечных состояний.

Схема конечного автомата с магазинной памятью изображена на рисунке 4.1.

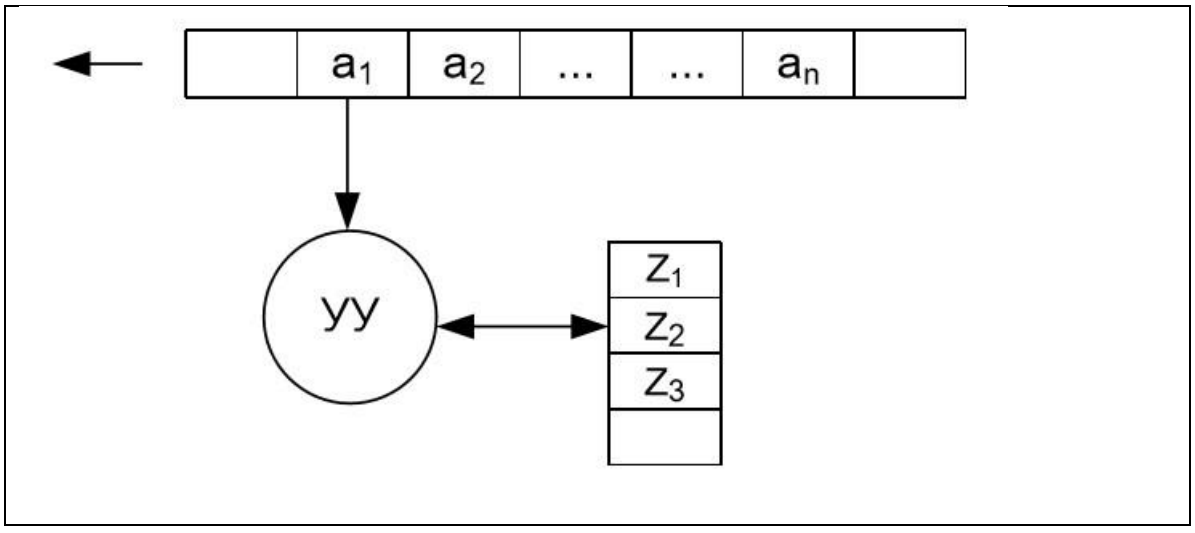

Рисунок 4.1 – Схема конечного автомата с магазинной памятью

Алгоритм работы конечного автомата с магазинной памятью:

−Состояние автомата (q, aα, zβ);

−Читает символ a находящийся под головкой (сдвигает ленту);

−Не читает ничего (читает λ, не сдвигает ленту);

−Из δ определяет новое состояние q', если (q', γ) ∈ δ(q, a, z) или (q', γ) ∈ δ(q, λ, z);

−Читает верхний (в стеке) символ z и записывает цепочку γ т.к. (q', γ) ∈ δ(q, a, z), при этом, если γ = λ, то верхний символ магазина просто удаляется;

−Работа автомата заканчивается (q, λ, λ).

Результат, демонстрирующий успешный разбор цепочки из контрольного примера, приведен в приложении Г.

4.4 Основные структуры данных

Программный код основных структур данных на языке C++, описывающих контекстно-свободную грамматику, представлен в приложении Д.

4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора языка KDA-2024:

1) В магазин заносится стартовый символ;
2) Формируется входная лента, полученная из таблицы лексем;
3) Нетерминальный символ раскрывается, согласно правилам, и записывается в магазин;
4) Если терминал на вершине стека и в начале ленты совпадают, то данный терминал удаляется из входной ленты. Иначе возвращается в предыдущее состояние и выбирает другую цепочку нетерминала;

5)Если в магазине встречается нетерминал, переход к пункту 3;

6) Если достигнуто дно стека и входная цепочка пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

Обобщенная блок-схема алгоритма синтаксического анализа изображена на рисунке 4.2.

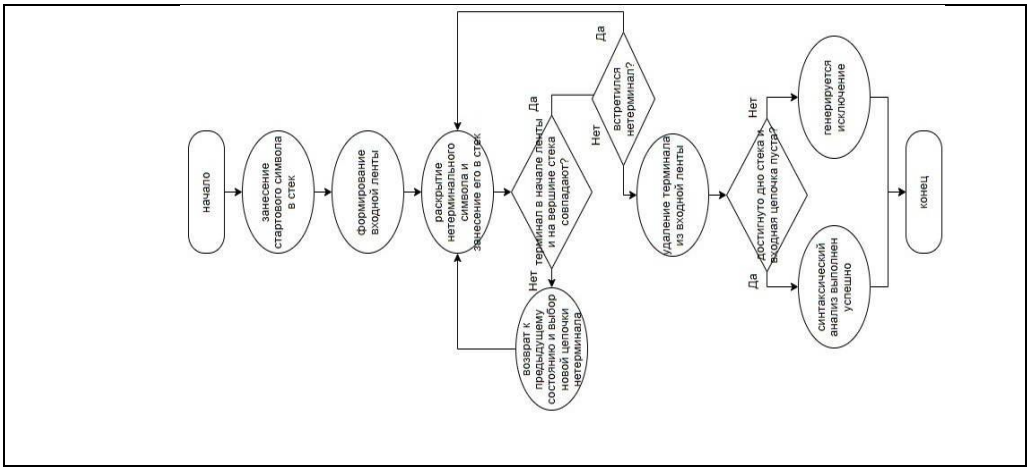

Рисунок 4.2 – Блок-схема алгоритма синтаксического анализа

В рисунке представлена блок-схема алгоритма синтаксического анализа.

4.6 Параметры синтаксического анализатора

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

Выходными параметрами являются правила разбора, которые выводятся на консоль.

4.7 Структура и перечень сообщений синтаксического анализатора

При возникновении ошибки в синтаксическом анализаторе формируется ошибка в следующем формате: Номер ошибки, пояснительный текст, строка в исходном тексте.

Перечень сообщений синтаксического анализатора представлен в таблице 4.2.

Таблица 4.2 – Перечень сообщений синтаксического анализатора языка KDA-2024

Номер ошибки	Пояснительный текст
600	Неверная структура программы
601	Ошибка в конструкции блока кода
602	Ошибка в выражении
603	Ошибка в параметрах функции
604	Ошибка в параметрах вызываемой функции
605	Ошибочный оператор
606	Ошибка в условной конструкции
607	Ошибка типа

В таблице представлена Перечень сообщений синтаксического анализатора языка KDA-2024

4.8 Принцип обработки ошибок

Обработка ошибок происходит по следующему алгоритму:

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

−При невозможности подбора подходящей цепочки, то генерируется соответствующая ошибка.

−Ошибки записываются в общую структуру ошибок.

−В случае нахождения ошибки, после всей процедуры трассировки в протокол и на консоль будет выведено диагностическое сообщение в файл протокола.log.

4.9 Контрольный пример

Результат работы синтаксического анализатора, полученный при выполнении контрольного примера, а именно дерево разбора, представлен в приложении Г.

5 Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализ языка KDA-2024 выполняется после выполнения лексического и синтаксического анализа. Несмотря на это, некоторые семантические проверки выполняются на этапе лексического анализа. На вход семантического анализатора подаются таблица лексем и таблица идентификаторов. Схема семантического анализатора представлена на рисунке 5.1.

Рисунок 5.1 – Схема семантического анализатора языка KDA-2024

На рисунке представлена Схема семантического анализатора языка KDA-2024

5.2 Функции семантического анализатора

Семантические проверки языка KDA-2024 с указанием фаз их выполнения приведены в таблице 5.1.

Таблица 5.1 – Семантические проверки языка KDA-2024

Семантическая проверка	Фаза выполнения
Превышение длины строки	Лексический анализ
Превышение целочисленного значения	Лексический анализ
Повторная реализация функции main	Лексический анализ
Превышение длины лексемы	Лексический анализ
Наличие точки входа в программу	Лексический анализ
Объявление идентификатора перед использованием	Семантический анализ
Повторное объявление идентификатора	Семантический анализ
Соответствие типов в выражении	Семантический анализ
Количество параметров функции	Семантический анализ

Окончание таблицы 5.1

Семантическая проверка	Фаза выполнения
Проверка левосторонних выражений	Семантический анализ
Повторная реализация функции	Семантический анализ
Соответствие типа возвращаемого значения типу функции	Семантический анализ
Соответствие параметров объявленной и вызываемой функции	Семантический анализ
Соответствие параметров встроенной функции	Семантический анализ

В таблице представлены Семантические проверки языка KDA-2024 с указанием фаз их выполнения

5.3 Структура и перечень сообщений семантического анализатора

Структура и текст сообщений семантического анализатора приведены в на рисунке 5.2. Таблица 5.2 – Перечень сообщений семантического анализатора языка KDA-2024

Номер ошибки	Пояснительный текст
130	Превышена длина строки в 255 символов
131	Функция main уже имеет реализацию
132	Превышена длина лексемы
133	Превышено значение целочисленного литерала (2 byte)
134	Не найдена точка входа в программу (main)
135	Идентификатор с таким именем не найден
136	Повторное объявление идентификатора
137	Несоответствие типов в выражении
138	Слишком много параметров в функции
139	Превышено количество функций
140	Несоответствие параметров объявленной и вызываемой функций

Окончание таблицы 5.2

141	Несоответствие параметров встроенной функции
142	Левостороннее выражение не является идентификатором и не должно являться функцией
143	Данная функция уже имеет реализацию
144	В вызове функции отсутствуют ()
145	Тип возвращаемого значения не соответствует типу функции
146	Оператор не предназначен для работы со строками
147	В функции отсутствует возвращаемое значение

В таблице представлена Перечень сообщений семантического анализатора языка KDA-2024

5.4 Принцип обработки ошибок

При обнаружении ошибки в исходном коде программы семантический анализатор формирует сообщение об ошибки и выводит его на консоль и в файл с протоколом работы, заданный параметром –log:.

5.5 Контрольный пример

Контрольный пример для демонстрации ошибок, диагностируемых семантическим анализатором вместе с отчетом выданных сообщений представлен в приложении А.

6 Вычисление выражений

6.1 Выражение, допускаемые языком

В языке KDA-2024 допускаются выражения, применимые к целочисленным типам данных. Допускается использование функций, возвращающих целочисленное значение, в выражениях. Операции и их приоритетность приведены в таблице 6.1.

Таблица 6.1 – Операции и их приоритетность языка KDA-2024

Приоритет	Операция
0	(
0	)
1	,
2	+
2	-
3	*
3	/
3	%

Примеры выражений из контрольного примера: 5*(pow(1011B,2)-B2H)+23O и т.п.

6.2 Польская запись и принцип её построения

Язык KDA-2024 транслируется в язык ассемблера, в котором все вычисления производятся через стек. Преобразование исходных выражений в обратную польскую запись, упрощает генерацию кода вычисления выражений в язык ассемблера. Алгоритм построения польской записи приведен ниже: Пока есть символы для чтения:

−Читаем очередной символ;

−Если символ является числом, добавляем его к выходной строке;

−Если символ является функцией, помещаем его в стек;

−Если символ является открывающей скобкой, помещаем его в стек; − Если символ является закрывающей скобкой:

До тех пор, пока верхним элементом стека не станет открывающая скобка, выталкиваем элементы из стека в выходную строку. При этом открывающая скобка удаляется из стека, но в выходную строку не добавляется.

–Если символ является операцией:

–Пока операция на вершине стека приоритетнее или пока на вершине стека функция;

–Помещаем операцию в стек.

Когда входная строка закончилась, выталкиваем все символы из стека в выходную строку.

Пример построения обратной польской записи: lllpl-*l+.

6.3 Программная реализация обработки выражений

Фрагмент кода, реализующего преобразование выражений в обратный польский формат представлен в листинге 6.1.

```
      case LEX_ID:
  {
       if(idtable.table[lextable.table[lextable_pos].idxTI].idType
== IT::IDTYPE::F)
       {
            stack.push(lextable.table[lextable_pos]);
            continue;
       }
       queue.push(lextable.table[lextable_pos]);
       continue;
  }  case
LEX_LITERAL:
  {
       queue.push(lextable.table[lextable_pos]);
       continue;
  }  case
LEX_LEFTHESIS:
  {
       stack.push(lextable.table[lextable_pos]);
       continue;
      } case
      LEX_RIGHTHESIS:
  {
```

Листинг 6.1 – Фрагмент кода, реализующего преобразование выражений

При встрече лексемы идентификатора, идет проверка на функцию. Если идентификатор является функцией, идентификатор помещается в стек. Иначе идентификатор помещается в выходную строку.

При встрече литерала, литерал помещается в выходную строку.

При встрече открывающей скобки, скобка кладется в стек.

При встрече закрывающей скобки, идет извлечение элементов из стека, пока не будет достигнута открывающая скобка.

6.4 Контрольный пример

В приложении В представлена обновленная таблица лексем, с выражениями, приведенными к обратной польской записи.

7 Генерация кода

7.1 Структура генератора кода

Трансляция с языка KDA-2024 производиться в язык ассемблера. Структура генератора кода изображена на рисунке 7.1.

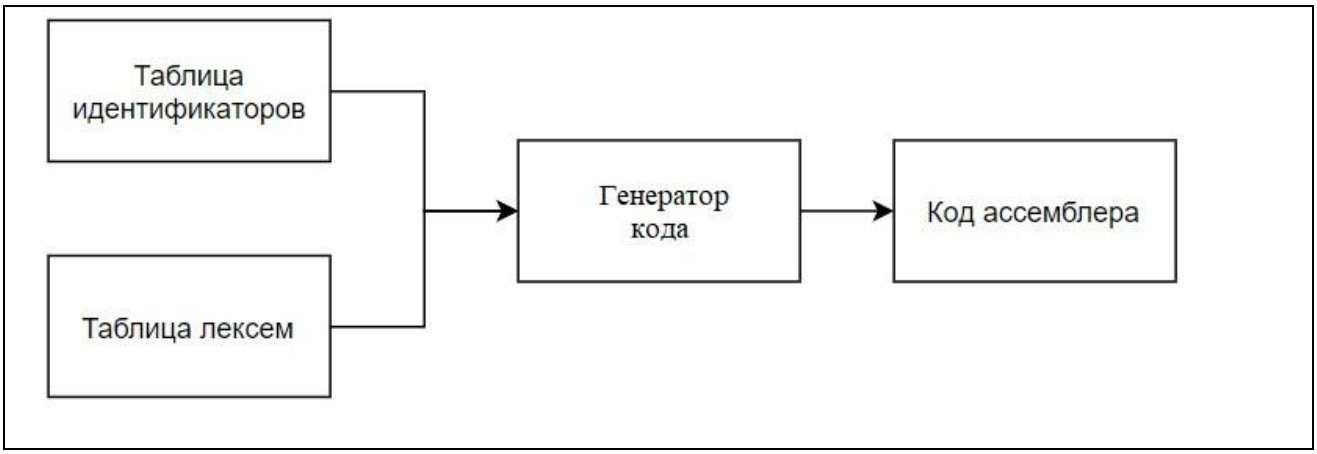

Рисунок 7.1 – Структура генератора кода

Генератор кода последовательно проходит таблицу лексем, при необходимости обращаясь к таблице идентификаторов.

7.2 Представление типов данных в оперативной памяти

Плоская модель памяти (flat): приложению для кода и данных предоставляется один непрерывный сегмент. Данный сегмент в свою очередь разбит на области:

–.STACK – стек;

–.CONST – константы;

– .DATA – переменные;

– .CODE – код.

Соответствие типов данных в исходном языке программирования KDA-2024 типам целевого языка приведены в таблице 7.1.

Таблица 7.1 – Соответствие типов идентификаторов языка KDA-2024 и языка ассемблера

Тип идентификатора языка KDA-2024	Тип идентификатора ассемблера	Пояснение
int	SDWORD	Хранит знаковый целочисленный тип.
str	DWORD	Хранит указатель на начало строки.
bool	DWORD	Хранит логическое значение.

В таблице представлено Соответствие типов идентификаторов языка KDA-2024 и языка ассемблера

7.3 Статическая библиотека

Функции, входящие в состав статической библиотеки языка KDA-2024, приведены в таблице 1.7.

Статическая библиотека написана на языке C++. Путь к статической библиотеке указан в Свойства проекта > Компоновщик > Командная строка. Библиотека подключается на этапе компоновки.

Таблица 1.7 – Функции стандартной библиотеки языка KDA-2024

Прототип функции	Описание
_pow(int a, int b);	Возводит число a в степень b и возвращает результат.
_abs(int a);	Берет абсолютное значение числа a и возвращает результат.
noutl(int value);	Выводит целочисленный идентификатор или литерал на консоль.
soutl(str value);	Выводит строковый идентификатор или литерал на консоль.
compare (str value1, str value1);	Выводит результат сравнения строк. Если первая строка больше возвращает 2, если строки равны, возвращает 1, если первая строка меньше, возвращает 0.

В таблице представлены функции стандартной библиотеки языка KDA-2024.

7.4 Особенности алгоритма генерации кода

Обобщенная блок-схема алгоритма генерации кода языка ассемблера изображена на рисунке 7.2.

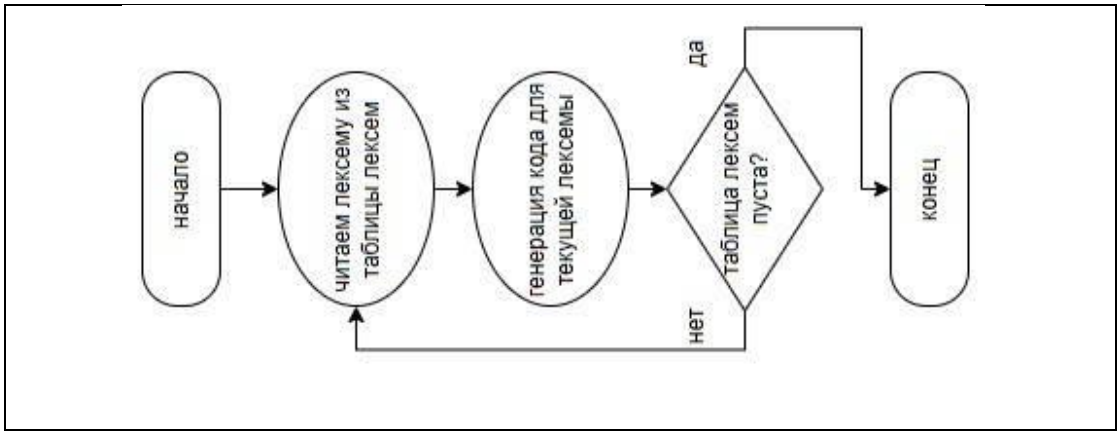

Рисунок 7.2 – Блок-схема алгоритма генерации кода в язык ассемблера

Пока таблица лексем не пуста, читаем лексему:

–Если для лексемы есть код генерации, генерируем код;

–Иначе читаем следующую лексему, пока таблица лексем не будет пуста.

В листинге 7.1 представлен блок макросов, используемый при генерации кода.

```
#define START \
".586\n"\
".model flat, stdcall\n"\
"includelib libucrt.lib\n"\
"includelib kernel32.lib\n"\

#define PROTOTYPES \
"\nExitProcess PROTO:DWORD "\
"\nSYSPAUSE PROTO "\
"\nsoutl PROTO : BYTE "\
"\nnoutl PROTO : SDWORD "\
"\n_pow PROTO  : SDWORD, :
SDWORD "\
"\n_abs PROTO  : SDWORD "\
"\n_compare PROTO  : SDWORD, :
SDWORD "\
"\n\n.STACK 4096\n\n"
```

```
#define FINISH \
"\tcall SYSPAUSE"\
"\n\tpush 0"\
"\n\tcall ExitProcess"\
"\nSOMETHINGWRONG::"\
"\n\tpush offset null_division"\
"\n\tcall soutl"\
"\njmp THEEND"\
"\noverflow::"\
"\n\tpush offset OVER_FLOW"\
"\n\tcall soutl"\
"\nTHEEND:"\
"\n\tcall SYSPAUSE"\
"\n\tpush -1"\
"\n\tcall ExitProcess"\
"\nmain ENDP\nend main"
```

Листинг 7.1 – Использование макросов при генерации

В листинге представлен блок макросов, используемый при генерации кода.

7.5 Параметры, управляющие генерацией кода

7.6 На вход генератору кода поступают таблицы лексем и идентификаторов.

Результат работы генератора кода выводится в файл с расширением .asm

7.7 Контрольный пример

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

На рисунке 7.3 приведен результат работы контрольного примера.

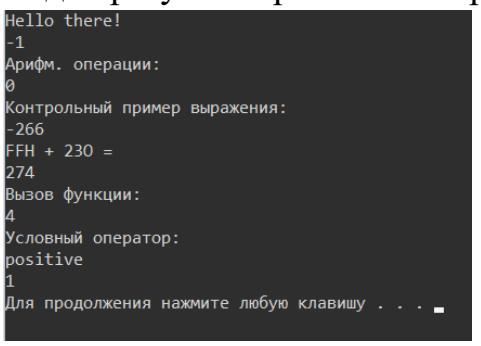

Рисунок 7.3 – Результат работы программы на языке KDA-2024

8 Тестирование транслятора

8.1 Общие положения

Тесты, как правило, предназначены для обнаружения различных ошибок, возникающих в процессе работы компилятора, а также для их последующего исправления. Эти ошибки могут быть выявлены как на ранних стадиях разработки компилятора, так и на более поздних этапах, что делает процесс тестирования неотъемлемой частью создания качественного программного обеспечения.

Важно отметить, что при возникновении ошибки процесс работы транслятора автоматически прекращается. Это связано с тем, что ошибка, возникшая на одном из этапов трансляции, может спровоцировать целую цепочку последующих ошибок, что, в свою очередь, усложняет процесс отладки (хотя для синтаксического анализатора могут быть исключения). Все найденные ошибки сопровождаются текстом, включающим номер ошибки и пояснительное сообщение.

Эта информация будет выведена как в файл протокола, так и на консоль, чтобы обеспечить максимальную прозрачность и удобство при анализе возникающих проблем.

8.2 Результаты тестирования

Тестовые наборы предназначены для проверки корректности работы компилятора на разных этапах трансляции. Их описание представлено в таблице 8.1, которая включает ключевые аспекты тестирования. Каждый тестовый набор ориентирован на выявление ошибок синтаксического, семантического и лексического анализа. Кроме того, тесты демонстрируют правильность генерации кода и оптимизации. Таким образом, таблица 8.1 охватывает все этапы трансляции, подтверждая функциональность разработанного компилятора.

Таблица 8.1 – Описание тестовых наборов языка KDA-2024

Фрагмент исходного кода	Диагностическое сообщение
Допустимость символов	
new int gf#;	Ошибка 110: Ошибка лексического анализатора: Недопустимый символ в исходном файле (-in:), строка 3, позиция 12

Продолжение таблицы 8.1

Лексический анализ	
24rgt str stroka;	Ошибка 117: Ошибка лексического анализатора: Не удалось определить тип лексемы, строка 3, позиция 1
Лексический анализ	
main{ new str str2 = “qwerty; };	Ошибка 121: Ошибка лексического анализатора: Ошибка с разбором строкового литерала, строка 2, позиция 1
Синтаксический анализ	
new main {}	Ошибка 600: Ошибка синтаксического анализа: Неверная структура программы, строка 1, позиция 1
if (5 > 6) {	Ошибка 601: Ошибка синтаксического анализа: Ошибка в конструкции блока кода, строка 3, позиция 1
new int x = 5 ++ 4;	Ошибка 602: Ошибка синтаксического анализа: Ошибка в выражении, строка 2, позиция 1
str function str1(str x,) {};	Ошибка 603: Ошибка синтаксического анализа: Ошибка в параметрах функции, строка 1, позиция 1
Синтаксический анализ	
new int x1 = pow(2,);	Ошибка 604: Ошибка синтаксического анализа: Ошибка в параметрах вызываемой функции, строка 2, позиция 1
new int x = (54 – 5;	Ошибка 605: Ошибка синтаксического анализа: Ошибочный оператор, строка 2, позиция 1
if (5 > d + 2)	Ошибка 606: Ошибка синтаксического анализа:
main{ new ind; };	Ошибка 607: Ошибка синтаксического анализа: Ошибка типа, строка 2, позиция 1
Семантический анализ	
main{new str s;}; main{new int i;};	Ошибка 131: Ошибка семантического анализа: Функция main уже имеет реализацию, строка 2, позиция 1
new int k = 32800;	Ошибка 133: Ошибка семантического анализа: Превышено значение целочисленного литерала (2 byte), строка 2, позиция 14

Продолжение таблицы 8.1

int function f1(){return 0;};	Ошибка 134: Ошибка семантического анализа: Не найдена точка входа в программу (main), строка -1, позиция -1
main{ x = 5; };	Ошибка 135: Ошибка семантического анализа: Идентификатор с таким именем не найден, строка 2, позиция 2
main{ new int r; new int r; };	Ошибка 136: Ошибка семантического анализа: Повторное объявление идентификатора, строка 3, позиция 13

main{
new int a = 5;
new str b = “stroka”;
a = a + b; };
	Ошибка 137: Ошибка семантического анализа: Несоответствие типов в выражении, строка 4, позиция 1

int function f1(int x){
return x;
};
entry{
f1("str");
};
	Ошибка 140: Ошибка семантического анализа: Несоответствие параметров объявленной и вызываемой функций, строка 6, позиция 1

main{
new int i = pow(2,3,4);
};
	Ошибка 141: Ошибка семантического анализа: Несоответствие параметров встроенной функции, строка 2, позиция 1

Семантический анализ	
int function f2(str x){…}; main{ f2 = 5; };	Ошибка 142: Ошибка семантического анализа: Левостороннее выражение не является идентификатором и не должно являться функцией, строка 7, позиция 1

int function f3(int a){
…
};
str function f3(str b){
…
};
	Ошибка 143: Ошибка семантического анализа: Данная функция уже имеет реализацию, строка 5, позиция 14

int function f4(int v){
…
};
main{
new int c = f4;
	Ошибка 144: Ошибка семантического анализа: В вызове функции отсутствуют (), строка 15, позиция 1

Окончание таблицы 8.1

int function f5(int v){
new str res = "qwerty";
return res;
};
	Ошибка 145: Ошибка семантического анализа: Тип возвращаемого значения не соответствует типу функции, строка 4, позиция 1

main{
new str s = "qwe" + "asd";
};
	Ошибка 146: Ошибка семантического анализа: Оператор не предназначен для работы со строками, строка 3, позиция 1

int function f6(){ new int s; };	Ошибка 147: Ошибка семантического анализа: В функции отсутствует возвращаемое значение, строка 1, позиция 1
main{ new int a = 3/0; };	Ошибка 148: Ошибка семантического анализа: Деление на 0 недопустимо, строка 3, позиция 1
main{ new str s = ""; };	Ошибка 149: Ошибка семантического анализа: Недопустимый строковый литерал, строка 3, позиция 13

Заключение

В ходе выполнения курсовой работы был разработан компилятор для языка KDA-2024. Выполнены минимальные требования к курсовому проекту, а также ряд дополнений.

Язык KDA-2024 включает в себя:

- 3 типа данных;
- оператор вывода данных в консоль;
- вызов функций из стандартной библиотеки;
- 5 арифметических операторов для вычисления выражений;
- 6 операций сравнения;
- условный оператор if-else;

Работа по разработке компилятора позволила получить необходимое представление о структурах и процессах, использующихся при построении компиляторов, также были изучены основы теории формальных грамматик и основы общей теории компиляторов.

Список использованных источников

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

2 Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – М.: Вильямс, 2005. – 912с.

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

4 Страуструп, Б. Принципы и практика использования С++ / Б. Страуструп – 2009 – 1238 с

5 Курс лекций по КПО / Наркевич А.С

Приложение А

Контрольный пример

```
int function func(int a, int b){
new int res = -1;
if(a<0)
{
res = abs(a);
}
else
{
res = pow(a, b);
}
return res;
};

str function setstring(str s){
return s;
};

main
{// There are comments here
new str str1 = setstring("Hello there!");
write str1;

new int res = compare("str2", "str1");
write res;
```

```
write "Арифм. операции:";
new int i = 5-2;
new int asd = 5%i;
write asd;
write "Контрольный пример выражения: ";
i = 5*(pow(1011B,2)-B2H)+23O;// 5*(pow(11,2)-178)+19 = -266
write i;
i = FFH + 23O;
write "FFH + 23O = ";
write i;

write "Вызов функции:";
new int num = func(-4, 3);
write num;

write "Условный оператор:";
new int c;
if(i >= 0)
{
write "positive";
c = 1;
}
else
{
write "negative";
c = -1;
```

```
}
write c;


};
```

Контрольный пример, содержащий 3 семантические ошибки

```
int function func(int a, int b){
  new int res = -1;
  if(a<0)
  {
       res = abs(a);
  }
  else
  {
       res = pow(a, b);
  }
  return res;
};

str function func(str s){
  return s;
};

main
{   // There are comments here  new str str1 =
setstring("Hello there!");       new str str1;  write
str1;
```

```
     new str str2 = “a”+5;

new int res = compare("str2", "str1");
write "Результат сравнения:";
     write res;




 write "Арифм.
операции:";  new int i =
26/13;
     write i;
     write "Контрольный пример выражения: ";
 i = 5*(pow(1011B,2)-B2H)+23O;  // 5*(pow(11,2)-178)+19 = -
266  write i;  i = FFH + 23O;  write "FFH + 23O = ";
     write i;

 write "Вызов
функции:";  new int num
= func(-4, 3);
     write num;

     write "Условный оператор:";
 new int c;
if(i >= 0)
     {
  write "positive";
c = 1;
     }
     else
     {
  write "negative";
 c = -1;
     }
     write c;

     new bool flag;
     if(flag == true)
     {
          write "Its true";
     }
     else
     {
          write "Its false";
```

Допущенные семантические ошибки:

−Ошибка 143: Ошибка семантического анализа: Данная функция уже имеет реализацию, строка 14, позиция 14

−Ошибка 136: Ошибка семантического анализа: Повторное объявление идентификатора, строка 21, позиция 10

−Ошибка 146: Ошибка семантического анализа: Оператор не предназначен для работы со строками, строка 21, позиция 1

Приложение Б

Таблица допустимых и запрещенных символов

```
#define IN_CODE_TABLE {\
    /*00*/ IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F,\
    /*16*/ 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,\
    /*32*/ IN::T, IN::T, IN::T, IN::F, IN::F, IN::T, IN::F,
IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\
    /*48*/ 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,\
    /*64*/ 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,\
    /*80*/ IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::I, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F,\
    /*96*/ 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,\
    /*112*/ 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::T, IN::F, IN::F,\
   \
    /*128*/ 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,\
    /*144*/ IN::F, IN::F, IN::F, IN::T, IN::T, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\
    /*160*/ 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,\
    /*176*/ 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,\
    /*192*/ 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,\
    /*208*/ 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,\
    /*224*/ 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,\
    /*240*/ 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,\ }
```

Приложение В

Таблица идентификаторов

№	Name	Type	Data type	First in LT	Value
0	func1	Function	Int	2	-
1	a1	Parameter	Int	5	-
2	b1	Parameter	Int	8	-
3	res10	Variable	Int	13	-
4	short0	Literal	Int	15	-1
5	short1	Literal	Int	21	0
6	setstring2	Function	Str	51	-
7	s2	Parameter	Str	54	-
8	stroka63	Variable	Str	66	-
9	str163	Variable	Str	70	-
10	str2	Literal	Str	74	"Hello there!"
11	str3	Literal	Str	81	"Арифм. операции:"
12	i63	Variable	Int	85	-
13	short4	Literal	Int	87	26
14	short5	Literal	Int	89	13
15	str6	Literal	Str	95	"Контрольный пример выражения: "
16	short7	Literal	Int	99	5
17	short8	Literal	Int	104	11
18	short9	Literal	Int	106	2
19	short10	Literal	Int	109	178
20	short11	Literal	Int	112	19
21	short12	Literal	Int	119	255
22	short13	Literal	Int	121	19
23	str14	Literal	Str	124	"FFH + 23O = "
24	str15	Literal	Str	130	"Вызов функции:"
25	num63	Variable	Int	134	-
26	short16	Literal	Int	138	-4
27	short17	Literal	Int	140	3
28	str18	Literal	Str	147	"Условный оператор:"
29	c63	Variable	Int	151	-
30	short19	Literal	Int	157	0
31	str20	Literal	Str	161	"positive"
32	short21	Literal	Int	165	1
33	str22	Literal	Str	171	"negative"

```
|34  |short23    |Literal      |Int          |175          |-1              |
|35  |flag63     |Variable     |Bool         |183          |-               |

|36  |bool24     |Literal      |Bool         |189          |true            |
|37  |str25      |Literal      |Str          |193          |"Its true"      |
|38  |str26      |Literal      |Str          |199          |"Its
false"       |
```

Приложение Г

==========LEX TABLE==========

1|№ |Line |Lexema |ID from IT|

2|0 |0 |d |- |

3|1 |0 |f |- |

4|2 |0 |i |0 |

5|3 |0 |(|- |

6|4 |0 |d |- |

7|5 |0 |i |1 |

8|6 |0 |, |- |

9|7 |0 |d |- |

10|8 |0 |i |2 |

11|9 |0 |) |- |

12|10 |0 |{ |- |

13|11 |1 |n |- |

14|12 |1 |d |- |

15|13 |1 |i |3 |

16|14 |1 |= |- |

17|15 |1 |l |4 |

18|16 |1 |; |- |

19|17 |2 |c |- |

20|18 |2 |(|- |

21|19 |2 |i |1 |

22|20 |2 |< |- |

23|21 |2 |l |5 |

24|22 |2 |) |- |

25|23 |3 |{ |- |

26|24 |4 |i |3 |

27|25 |4 |= |- |

28|26 |4 |a |- |

29|27 |4 |(|- |

30|29 |4 |) |- |

31|30 |4 |; |- |

32|31 |5 |} |- |

33|33 |7 |{ |- |

34|34 |8 |i |3 |

35	35	8	=	-
36	36	8	p	-
37	37	8	(	-
38	38	8	i	1
39	39	8	,	-
40	40	8	i	2
41	41	8	)	-
42	42	8	;	-
43	43	9	}	-
44	44	10	r	-
45	45	10	i	3
46	46	10	;	-
47	47	11	}	-
48	48	11	;	-
49	49	13	s	-
50	50	13	f	-
51	51	13	i	6
52	52	13	(	-
53	53	13	s	-
54	54	13	i	7
55	55	13	)	-
56	56	13	{	-
57	57	14	r	-
58	58	14	i	7
59	59	14	;	-
60	60	15	}	-
61	61	15	;	-
62	62	17	m	-
63	63	18	{	-
64	64	18	n	-
65	65	18	s	-
66	66	18	i	8
67	67	18	=	-
68	68	18	i	6
69	69	18	(	-
70	70	18	l	9
71	71	18	)	-

72|72 |18 |; |- |
73|73 |19 |o |- |
74|74 |19 |i |8 |
75|75 |19 |; |- |
76|76 |21 |o |- |
77|77 |21 |l |10 |
78|78 |21 |; |- |
79|79 |21 |n |- |
80|80 |21 |d |- |
81|81 |21 |i |11 |
82|82 |21 |= |- |
83|83 |21 |l |12 |
84|84 |21 |/ |- |
85|85 |21 |l |13 |
86|86 |21 |; |- |
87|87 |22 |o |- |
88|88 |22 |i |11 |
89|89 |22 |; |- |
90|90 |23 |o |- |
91|91 |23 |l |14 |
92|92 |23 |; |- |
93|93 |24 |i |11 |
94|94 |24 |= |- |
95|95 |24 |l |15 |
96|96 |24 |* |- |
97|97 |24 |(|- |
98|98 |24 |p |- |
99|99 |24 |(|- |
100|100 |24 |l |16 |
101|101 |24 |, |- |
102|102 |24 |l |17 |
103|103 |24 |) |- |
104|104 |24 |- |- |
105|105 |24 |l |18 |
106|106 |24 |) |- |
107|107 |24 |+ |- |
108|108 |24 |l |19 |

109	109	24	;	-
110	110	25	o	-
111	111	25	i	11
112	112	25	;	-
113	113	27	o	-
114	114	27	l	20
115	115	27	;	-
116	116	27	n	-
117	117	27	d	-
118	118	27	i	21
119	119	27	=	-
120	120	27	i	0
121	121	27	(	-
122	122	27	l	22
123	123	27	,	-
124	124	27	l	23
125	125	27	)	-
126	126	27	;	-
127	127	28	o	-
128	128	28	i	21
129	129	28	;	-
130	130	30	o	-
131	131	30	l	24
132	132	30	;	-
133	133	31	n	-
134	134	31	d	-
135	135	31	i	25
136	136	31	;	-
137	137	31	c	-
138	138	31	(	-
139	139	31	i	11
140	140	31	]	-
141	141	31	l	26
142	142	31	)	-
143	143	32	{	-
144	32	6	e	-
145	144	33	o	-

146|145 |33 |l |27 |
147|146 |33 |; |- |
148|147 |33 |i |25 |
149|148 |33 |= |- |
150|149 |33 |l |28 |
151|150 |33 |; |- |
152|151 |34 |} |- |
153|152 |35 |e |- |
154|153 |36 |{ |- |
155|154 |37 |o |- |
156|155 |37 |l |29 |
157|156 |37 |; |- |
158|157 |37 |i |25 |
159|158 |37 |= |- |
160|159 |37 |l |30 |
161|160 |37 |; |- |
162|161 |38 |} |- |
163|162 |39 |o |- |
164|163 |39 |i |25 |
165|164 |39 |; |- |
166|165 |41 |n |- |
167|166 |41 |b |- |
168|167 |41 |i |31 |
169|168 |41 |; |- |
170|169 |42 |c |- |
171|170 |42 |(|- |
172|171 |42 |i |31 |
173|172 |42 |& |- |
174|173 |42 |l |32 |
175|174 |42 |) |- |
176|175 |43 |{ |- |
177|176 |44 |o |- |
178|177 |44 |l |33 |
179|178 |44 |; |- |
180|179 |45 |} |- |
181|180 |46 |e |- |
182|181 |47 |{ |- |

183|182 |48 |o |- |
184|183 |48 |l |34 |
185|184 |48 |; |- |
186|185 |49 |} |- |
187|186 |50 |} |- |
188|187 |50 |; |- |

==========ID TABLE==========

Последовательность правил грамматики

0 : S->dfi(F){N};S	92 : L->i	157 : L->l
4 : F->di,F	94 : N->oL;N	160 : N->oL;N
7 : F->di	95 : L->l	161 : L->l
11 : N->nTi=E;N	97 : N->i=E;N	163 : N->i=E;
12 : T->d	99 : E->lM	165 : E->l
15 : E->l	100 : M->*E	170 : N->oL;N
17 : N->cQ{N}e{N}N	101 : E->(E)M	171 : L->l
18 : Q->(L<L)	102 : E->p(W)M	173 : N->i=E;
19 : L->i	104 : W->l,W	175 : E->l
21 : L->l	106 : W->l	178 : N->oL;N
24 : N->i=E;	108 : M->-E	179 : L->i
26 : E->a(W)	109 : E->l	181 : N->nTi;N

28 : W->i	111 : M->+E	182 : T->b
34 : N->i=E;	112 : E->l	185 : N->cQ{N}e{N}
36 : E->p(W)	114 : N->oL;N	186 : Q->(L&L)
38 : W->i,W	115 : L->i	187 : L->i
40 : W->i	117 : N->i=E;N	189 : L->l
44 : N->rL;	119 : E->lM	192 : N->oL;
45 : L->i	120 : M->+E	193 : L->l
49 : S->sfi(F){N};S	121 : E->l	198 : N->oL;
53 : F->si	123 : N->oL;N	199 : L->l
57 : N->rL;	124 : L->l	
58 : L->i	126 : N->oL;N	
62 : S->m{N};	127 : L->i	
64 : N->nTi;N	129 : N->oL;N	
65 : T->s	130 : L->l	
68 : N->nTi=E;N	132 : N->nTi=E;N	
69 : T->s	133 : T->d	
72 : E->i(W)	136 : E->i(W)	
74 : W->l	138 : W->l,W	
77 : N->oL;N	140 : W->l	
78 : L->i	143 : N->oL;N	
80 : N->oL;N	144 : L->i	
81 : L->l	146 : N->oL;N	
83 : N->nTi=E;N	147 : L->l	
84 : T->d	149 : N->nTi;N	
87 : E->lM	150 : T->d	
88 : M->/E	153 : N->cQ{N}e{N}N	
89 : E->l	154 : Q->(L]L)	

Приложение Д

Структур данных, описывающие контекстно-свободную грамматику

```
struct Rule            // правило в грамматике Грейбах
   {
  GRBALPHABET nn; // нетерминал (левый символ правила) < 0   int
iderror; // идентификатор диагностического сообщения   short size; //
количество цепочек - правых частей правила   struct Chain   //
цепочка (правая часть правила)
        {
             short size;                    // длина цепочки
   GRBALPHABET* nt;  // цепочка терминалов (> 0) и нетерминалов (< 0)
             Chain() { size = 0; nt = 0; };
             Chain(
    short psize, // количество символов в цепочке      GRBALPHABET s,
...   // символы (терминал или нетерминал)
             );
   char* getCChain(char* b);        // получить правую сторону
правила
              static GRBALPHABET T(char t) { return GRBALPHABET(t);
};    // терминал
              static GRBALPHABET N(char n) { return -GRBALPHABET(n);
};   // нетерминал    static bool isT(GRBALPHABET s) { return s >
0; };   // терминал?    static bool isN(GRBALPHABET s) { return
!isT(s); };   // нетерминал?
   static char alphabet_to_char(GRBALPHABET s) { return isT(s) ?
char(s) : char(-s); }; // GRBALPHABET->char
        }*chains;      // массив цепочек - правых частей правила
        Rule() { nn = 0x00; size = 0; };
        Rule(
             GRBALPHABET pnn,         // нетерминал (< 0)
   int iderror,  // идентификатор диагностического сообщения (Error)
   short psize,  // количество цепочек - правых частей правила
   Chain c, ...   // множество цепочек - правых частей правила
        );
  char* getCRule(   // получить правило в виде Nцепочки (для
распечатки)
             char* b,                   // буфер
              short nchain // номер цепочки (правой части) в правиле
        );
        short getNextChain(           // получить следующую за j
подходящую цепочку, вернуть её номер или -1
```

```
   GRBALPHABET t,    // первый символ цепочки    Rule::Chain& pchain,
// возвращаемая цепочка
              short j                          // номер цепочки
         );
    };

    struct Greibach                           // грамматика Грейбах
    {
         short size;                          // количество правил
         GRBALPHABET startN;                  // стартовый символ
         GRBALPHABET stbottomT;               // дно стека
         Rule* rules;                         // множество правил
  Greibach() { short size = 0; startN = 0; stbottomT = 0;  rules =
0; };
         Greibach(
   GRBALPHABET pstartN, // стартовый символ    GRBALPHABET
pstbottomT, // дно стека    short psize,    // количество
правил
              Rule r, ...                     // правила
         );
         short getRule(                       // получить правило,
возвращающая номер правила или -1
              GRBALPHABET pnn,                // левый символ
              Rule& prule                 // возвращаемое правило грамматики
         );
          Rule getRule(short n);          // получить правило по номеру
     };
```

Приложение Е

Результат генерации кода

```
.586 .model flat,
stdcall includelib
libucrt.lib
includelib
kernel32.lib

ExitProcess PROTO:DWORD
SYSPAUSE PROTO  soutl PROTO
: BYTE  noutl PROTO : SDWORD
_pow PROTO  : SDWORD, :
SDWORD
_abs PROTO  : SDWORD

.STACK 4096

.CONST  null_division BYTE
'ERROR:
DIVISION BY ZERO', 0
 OVER_FLOW BYTE 'ERROR:
OVERFLOW', 0  true BYTE 'true',
0  false BYTE 'false', 0  short0
SDWORD -1  short1 SDWORD 0  str2
BYTE "Hello there!", 0  str3
BYTE "Арифм. операции:",
0  short4 SDWORD 26  short5
SDWORD 13  str6 BYTE
"Контрольный пример
выражения: ", 0  short7 SDWORD
5  short8 SDWORD 11  short9
SDWORD 2  short10 SDWORD 178
short11 SDWORD 19  short12
SDWORD 255  short13 SDWORD 19
str14 BYTE "FFH + 23O = ", 0
str15 BYTE "Вызов функции:",
0  short16 SDWORD -4
short17 SDWORD 3  str18
BYTE "Условный оператор:",
0  short19 SDWORD 0  str20
BYTE "positive", 0
short21 SDWORD 1  str22
BYTE "negative", 0  mov
res10, eax
```

```
short23 SDWORD -1  bool24 DWORD
1  str25 BYTE "Its true", 0
str26 BYTE "Its false", 0
.DATA  res10 SDWORD 0
str163 DWORD ?  i63
SDWORD 0  num63
SDWORD 0  c63 SDWORD
0
    flag63 DWORD 0

.CODE
 func1 PROC b1 : SDWORD, a1 :
SDWORD
 push short0  pop eax
cmp eax, 32767  jg
overflow  cmp eax, -
32768  jl overflow
mov res10, eax  mov
eax, a1
 cmp eax, short1  jl
ifi1  jge else1 ifi1:
 push a1  call _abs
push eax  pop eax  cmp
eax, 32767  jg
overflow  cmp eax, -
32768  jl overflow
mov res10, eax  jmp
ifEnd1 else1:
 push a1  push b1
call _pow  push eax
pop eax  cmp eax,
32767  jg overflow
cmp eax, -32768  jl
overflow  push eax
```

```
ifEnd1:
 push res10  jmp
local0 local0:
     pop eax
     ret
func1 ENDP

setstring2 PROC s2 : DWORD
 push s2  jmp
local1 local1:
 pop eax  ret
setstring2 ENDP
 main PROC  push
offset str2  call
setstring2  push
eax
     pop str163

push str163
call soutl

push offset str3
call soutl
 push short4
push short5
pop ebx  pop
eax  cmp ebx,0
     je SOMETHINGWRONG
     cdq
 idiv ebx  push
eax  pop eax  cmp
eax, 32767  jg
overflow  cmp eax,
-32768  jl
overflow
     mov i63, eax
 push i63
call
noutl

push offset str6
call soutl
 push short7
push short8  push
short9  call _pow
mov num63, eax
push short10
pop ebx pop
eax sub eax,
ebx push eax
pop eax pop
ebx mul ebx
push eax push
short11 pop
eax pop ebx
add eax, ebx
push eax pop
eax cmp eax,
32767 jg
overflow cmp
eax, -32768 jl
overflow mov
i63, eax
```

```
push i63
call noutl
 push short12
push short13  pop
eax  pop ebx  add
eax, ebx  push eax
pop eax  cmp eax,
32767  jg overflow
cmp eax, -32768
jl overflow
     mov i63, eax

push offset str14
call soutl
 push
i63 call
noutl

push offset str15
call soutl
 push short16
push short17  call
func1  push eax
pop eax  cmp eax,
32767  jg overflow
cmp eax, -32768
jl overflow
ifEnd3:
```

```
 push num63
call noutl

push offset str18 call
soutl
 mov eax, i63  cmp
eax, short19  jge ifi2
jl else2 ifi2:
push offset str20 call
soutl
 push short21  pop
eax  cmp eax, 32767
jg overflow  cmp eax,
-32768  jl overflow
mov c63, eax  jmp
ifEnd2 else2:
```

```
     call SYSPAUSE
     push 0
call ExitProcess
SOMETHINGWRONG::
          push offset
null_division
     call soutl
jmp THEEND
overflow::
          push offset
OVER_FLOW
     call soutl
THEEND:
 call
SYSPAUSE
push -1
call
ExitProcess
main ENDP
end main
```

```
push offset str22
call soutl
 push short23
pop eax  cmp
eax, 32767  jg
overflow  cmp
eax, -32768  jl
overflow  mov
c63, eax ifEnd2:
push c63
call noutl  mov
eax, flag63  cmp
eax, bool24  jz
ifi3  jnz else3
ifi3:
push offset str25
call soutl
 jmp ifEnd3
else3:
push offset str26
call soutl
```