

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 САУ-2024»

Выполнил студент Чередник Антон Юрьевич
(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 _____

Содержание

Введение.....	4
1. Спецификация языка программирования.....	5
1.1 Характеристика языка программирования.....	5
1.2 Определение алфавита языка программирования.....	5
1.3 Применяемые сепараторы.....	5
1.4 Применяемые кодировки	6
1.5 Типы данных	6
1.6 Преобразование типов данных	6
1.7 Идентификаторы	7
1.8 Литералы.....	7
1.9 Объявление данных	8
1.10 Инициализация данных.....	8
1.11 Инструкции языка.....	8
1.12 Операции языка.....	9
1.13 Выражение и их вычисление	10
1.14 Конструкции языка	11
1.15 Область видимости идентификаторов	11
1.16 Семантические проверки	11
1.17 Распределение оперативной памяти на этапе выполнения	12
1.18 Стандартная библиотека и её состав	12
1.19 Ввод и вывод данных.....	13
1.20 Точка входа	13
1.21 Препроцессор	13
1.22 Соглашение о вызовах.....	13
1.23 Объектный код	14
1.24 Классификация сообщений транслятора.....	14
1.25 Контрольный пример.....	14
2. Структура транслятора	15
2.1 Компоненты транслятора их назначение и принципы взаимодействия	15
2.2 Перечень входных параметров транслятора	16
2.3 Протоколы, формируемые транслятором.....	16
3. Разработка лексического анализатора.....	18
3.1 Структура лексического анализатора	18
3.2 Контроль входных символов	18
3.3 Удаление избыточных символов	19
3.4 Перечень ключевых слов	19
3.5 Основные структуры данных	21
3.6 Структура и перечень сообщений лексического анализатора.....	22
3.7 Принцип обработки ошибок.....	22
3.8 Параметры лексического анализатора.....	23
3.9 Алгоритм лексического анализа.....	23
3.10 Контрольный пример.....	24
4. Разработка синтаксического анализатора.....	25
4.1 Структура синтаксического анализатора.....	25

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка	25
4.3 Построение конечного магазинного автомата	27
4.4 Основные структуры данных	28
4.5 Описание алгоритма синтаксического разбора	28
4.6 Структура и перечень сообщений синтаксического анализатора	29
4.7 Параметры синтаксического анализатора и режимы его работы	30
4.8 Принцип обработки ошибок	30
4.9 Контрольный пример	31
5. Разработка семантического анализатора	32
5.1 Структура семантического анализатора	32
5.2 Функции семантического анализатора	32
5.3 Структура и перечень сообщений семантического анализатора	33
5.4 Принцип обработки ошибок	35
5.5 Контрольный пример	35
6. Вычисление выражений	36
6.1 Выражения, допускаемые языком	36
6.2 Польская запись и принцип ее построения	36
6.3 Программная реализация обработки выражений	37
6.4 Контрольный пример	38
7. Генерация кода	39
7.1 Структура генератора кода	39
7.2 Представление типов данных в оперативной памяти	39
7.3 Статическая библиотека	40
7.4 Особенности алгоритма генерации кода	40
7.5 Входные параметры, управляющие генерацией кода	41
7.6 Контрольный пример	41
8. Тестирование транслятора	42
8.1 Общие положения	42
8.2 Результаты тестирования	42
Заключение	46
Список использованных литературных источников	47
Приложение А	48
Приложение Б	50
Приложение В	66
Приложение Г	72
Приложение Д	89
Приложение Е	92
Приложение Ж	99
Приложение З	103

Введение

В данном курсовом проекте требуется разработать собственный язык программирования под названием САУ-2024, а также создать для него транслятор. Транслятор будет реализован на языке C++, а код, написанный на языке САУ-2024 будет преобразовываться в ассемблер.

Транслятор САУ-2024 включает следующие компоненты: – лексический и семантический анализаторы; – синтаксический анализатор; – генератор ассемблерного кода.

Основные задачи проекта можно распределить следующим образом:

- разработка спецификации языка САУ-2024;
- создание лексического анализатора;
- создание синтаксического анализатора;
- создание семантического анализатора;
- обработка арифметических выражений;
- разработка генератора кода;
- тестирование транслятора.

Описание решений каждой задачи будет представлено в отдельных главах проекта.

1. Спецификация языка программирования

1.1 Характеристика языка программирования

Язык программирования САУ-2024 основан на процедурной парадигме и не поддерживает объектно-ориентированные принципы. САУ -2024 является строго типизированным, что исключает автоматическое приведение типов, и предназначен для трансляции в машинный код.

1.2 Определение алфавита языка программирования

Алфавит языка САУ-2024 основан на кодировке Windows-1251, представленной на рисунке 1.1.

Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ
0	0	спец. NOP	32	20	спец. SP (Пробел)	64	40	@	96	60	·	128	80	Ъ	160	A0		192	C0	А
1	1	спец. SOH	33	21	!	65	41	А	97	61	а	129	81	Г	161	A1	Ҁ	193	C1	Б
2	2	спец. STX	34	22	"	66	42	В	98	62	в	130	82	Д	162	A2	҂	194	C2	В
3	3	спец. ETX	35	23	#	67	43	С	99	63	с	131	83	Е	163	A3	Ј	195	C3	Г
4	4	спец. EOT	36	24	\$	68	44	Д	100	64	д	132	84	Ж	164	A4	Ѓ	196	C4	Д
5	5	спец. ENQ	37	25	%	69	45	Е	101	65	е	133	85	З	165	A5	Ѕ	197	C5	Е
6	6	спец. ACK	38	26	&	70	46	Ж	102	66	ж	134	86	И	166	A6	Ї	198	C6	Ж
7	7	спец. BEL	39	27	'	71	47	Г	103	67	г	135	87	Й	167	A7	Ї	199	C7	З
8	8	спец. BS	40	28	(72	48	И	104	68	и	136	88	К	168	A8	Ї	200	C8	И
9	9	спец. Tab	41	29)	73	49	И	105	69	і	137	89	Л	169	A9	Ї	201	C9	Й
10	0A	спец. LF	42	2A	*	74	4A	Ј	106	6A	ј	138	8A	Ль	170	AA	Ї	202	CA	К
11	0B	спец. VT	43	2B	+	75	4B	К	107	6B	к	139	8B	Э	171	AB	Ї	203	CB	Л
12	0C	спец. FF	44	2C	,	76	4C	Л	108	6C	л	140	8C	Ь	172	AC	Ї	204	CC	М
13	0D	спец. CR	45	2D	-	77	4D	М	109	6D	м	141	8D	К	173	AD	Ї	205	CD	Н
14	0E	спец. SO	46	2E	.	78	4E	Н	110	6E	н	142	8E	Т	174	AE	Ї	206	CE	О
15	0F	спец. SI	47	2F	/	79	4F	О	111	6F	о	143	8F	Ц	175	AF	Ї	207	CF	П
16	10	спец. DLE	48	30	0	80	50	Р	112	70	р	144	90	Х	176	B0	Ї	208	D0	Р
17	11	спец. DC1	49	31	1	81	51	Q	113	71	q	145	91	·	177	B1	Ї	209	D1	С
18	12	спец. DC2	50	32	2	82	52	R	114	72	r	146	92	·	178	B2	Ї	210	D2	Т
19	13	спец. DC3	51	33	3	83	53	S	115	73	s	147	93	·	179	B3	Ї	211	D3	У
20	14	спец. DC4	52	34	4	84	54	T	116	74	t	148	94	·	180	B4	Ї	212	D4	Ф
21	15	спец. NAK	53	35	5	85	55	U	117	75	u	149	95	·	181	B5	Ї	213	D5	Х
22	16	спец. SYN	54	36	6	86	56	V	118	76	v	150	96	·	182	B6	Ї	214	D6	Ц
23	17	спец. ETB	55	37	7	87	57	W	119	77	w	151	97	·	183	B7	Ї	215	D7	Ч
24	18	спец. CAN	56	38	8	88	58	X	120	78	x	152	98	·	184	B8	Ї	216	D8	Ш
25	19	спец. EM	57	39	9	89	59	Y	121	79	y	153	99	·	185	B9	Ї	217	D9	Щ
26	1A	спец. SUB	58	3A	:	90	5A	Z	122	7A	z	154	9A	·	186	BA	Ї	218	DA	Ъ
27	1B	спец. ESC	59	3B	;	91	5B	[123	7B	{	155	9B	·	187	BB	Ї	219	DB	Ы
28	1C	спец. FS	60	3C	<	92	5C	\	124	7C		156	9C	·	188	BC	Ї	220	DC	Ь
29	1D	спец. GS	61	3D	=	93	5D]	125	7D	}	157	9D	·	189	BD	Ї	221	DD	Э
30	1E	спец. RS	62	3E	>	94	5E	^	126	7E	~	158	9E	·	190	BE	Ї	222	DE	Ю
31	1F	спец. US	63	3F	?	95	5F	_	127	7F	~	159	9F	·	191	BF	Ї	223	DF	Я

Рисунок 1.1 – Алфавит входных символов

В данной таблице определены латинские и русские символы, символы-разделители, символы математических операций и специальные символы.

1.3 Применяемые сепараторы

В языке САУ-2024 определён набор символов сепараторов для разделения лексем друг от друга, приведенные ниже в таблице 1.1.

Таблица 1.1 - Применяемые сепараторы

Сепаратор	Назначение сепаратором
{ }	Программный блок
Пробел Табуляция	Разделитель конструкций и лексем
;	Завершение операторов
()	Определение параметров функций и вызова функций, группировка выражений

Окончание таблицы 1.1

“”	Выделение строковых литералов
,	Разделитель параметров функции

Сепараторы играют ключевую роль в разделении структурных элементов программы, обеспечивая корректное восприятие и анализ исходного кода. Их использование строго регламентировано синтаксисом языка САУ-2024

1.4 Применяемые кодировки

Для написания исходного текста на разрабатываемом языке САУ-2024 используется кодировка windows-1251. Разрешены латинские и русские символы. Русские символы используются только для строковых литералов.

1.5 Типы данных

В языке САУ-2024 разрешены следующие типы данных, представленные в таблице 1.2.

Таблица 1.2 – Типы данных языка САУ-2024

Тип данных	Описание	Диапазон значений	Значение по умолчанию
int	Целочисленный тип данных (2 байта), используется для хранения целых чисел без дробной части.	от -32,768 до 32,767	0
str	Строковый тип данных, используется для хранения последовательности символов.	До 250 символов	Пустая строка ("")
bool	Логический тип данных.	true или false	false

Данные типы обеспечивают поддержку как базовых, так и сложных структур для работы с различными видами информации. Их выбор определяется требованиями к точности, объёму и характеру обрабатываемых данных.

1.6 Преобразование типов данных

Язык САУ-2024 не поддерживает преобразования типов, что обеспечивает строгий контроль над использованием данных и предотвращает ошибки, связанные с неявным изменением их представления. Такой подход повышает надёжность программного кода, требуя от разработчика явного указания всех операций над данными разных типов.

1.7 Идентификаторы

Идентификатор языка — это имя, которое используется для обозначения переменной, функции, класса, модуля или другого элемента в языке программирования. Идентификаторы должны быть уникальными в пределах своей области видимости, чтобы избежать конфликтов и путаницы.

Имя идентификатора не может совпадать с каким-либо ключевым словом языка. Может состоять только латинских букв, цифр и знака нижнего подчёркивания. Первой в имени идентификатора не может быть цифра.

Регулярное выражения для описания правил записи имени идентификатора: [a-zA-Z][a-zA-Z0-9_].

Примеры правильных идентификаторов: digit, digit17, digit_1.

Примеры неправильных идентификаторов: 17digit.

1.8 Литералы

Литерал — это константное (фиксированное) значение, которое используется в исходном коде программы. Литералы представляют собой конкретные данные, которые компилятор может непосредственно использовать в процессе выполнения программы.

В языке SAY-2024 разрешены литералы, представленные в таблице 1.3.

Таблица 1.3 – Литералы языка SAY-2024

Тип литерала	Описание	Правила записи
Целочисленный	Задаётся в десятичном, шестнадцатеричном или двоичном представлении.	Десятичный: 123, Шестнадцатеричный: 0x7B, Двоичный: [0-1]+b
Строковый	Представляет последовательности символов, заключённых в двойные ("") кавычки.	"Hello, World!"
Логический	Представляет логические значения true или false.	Записываются в нижнем регистре: true, false

Использование литералов позволяет задавать значения переменных непосредственно в коде программы, обеспечивая читаемость и упрощение процесса разработки. В языке SAY-2024 литералы строго типизированы, что исключает неоднозначность их интерпретации компилятором

1.9 Объявление данных

Переменная может быть объявлена в любом программном блоке, при условии, что она не была ранее объявлена в пределах данной области видимости. Она не может иметь глобальную область видимости; если переменная объявляется вне какого-либо программного блока, это будет расцениваться транслятором как синтаксическая ошибка.

Правила объявления переменной: <ключевое слово var><тип данных><идентификатор>

Пример: var int digit

Для объявления функций используется ключевое слово func, перед которым указывается тип возвращаемого значения.

Пример объявления функции: int func Sum(a, b).

1.10 Инициализация данных

В языке SAY-2024 инициализация данных может происходить как во время объявления переменной, так и после него. Если переменная не инициализирована явно, ей будет присвоено значение по умолчанию:

- Для целочисленных переменных – значение 0.
- Для строковых переменных – пустая строка.
- Для логических переменных – значение false.

Эти правила гарантируют, что переменные всегда имеют определенное начальное значение, что способствует предотвращению ошибок во время выполнения программы.

1.11 Инструкции языка

В языке SAY-2024 инструкции представляют собой команды, которые выполняют определённые действия. Ниже, в таблице 1.4, перечислены основные инструкции языка.

Таблица 1.4 — Инструкции языка

Инструкция	Синтаксис	
Объявление переменной	<ключевое слово var> <тип данных> <идентификатор>;	Используется для объявления новой переменной с указанным типом данных.
Присваивание	<идентификатор> = <присваиваемое значение>;	Присваивает указанное значение переменной.
Объявление функции	<тип данных> func <идентификатор функции> (<тип данных> <идентификатор>, ...) {<тело функции>}	Определяет новую функцию с указанным идентификатором, параметрами и телом функции.

Окончание таблицы 1.4

Вызов функции	<идентификатор функции> (<идентификатор >, ...));	Вызывает указанную функцию, передавая ей аргументы.
Вывод данных	write(<литерал> <идентификатор>); writeline(<литерал> <идентификатор>);	Вывод в стандартный поток вывода.
Цикл	while (<условие>) {<инструкции>;}	Позволяет выполнять блок инструкций до тех пор, пока заданное условие истинно.
Условие	if (<условие>) {<инструкции>;} else {<инструкции>;}	Позволяет выполнять блок инструкций, если заданное условие истинно.

Инструкции языка САУ-2024 обеспечивают базовую функциональность для работы с переменными, управления потоком выполнения и взаимодействия с пользователем. Эти команды составляют основу программирования на данном языке.

1.12 Операции языка

В языке САУ-2024 предусмотрены операции, представленные в таблице 1.5. Приоритетность операций определяется при помощи круглых скобок. При попытке выполнить операцию с разными типами возникнет ошибка.

Таблица 1.5 — Операции языка

Операция	Приоритетность	Арифметическое значение	Свойства
=	1	Присваивание	Не является коммутативной, так как присваивание происходит слева направо.
==	2	Равенство	Не является коммутативной, так как порядок операндов важен.
!=	2	Неравенство	Не является коммутативной, так как порядок операндов важен.

Окончание таблицы 1.5

<	3	Меньше	Антикоммутативность: $a < b$ не эквивалентно $b < a$.
>	3	Больше	Антикоммутативность
<=	3	Меньше или равно	Антикоммутативность
>=	3	Больше или равно	Антикоммутативность
+	4	Сложение	Коммутативность, ассоциативность
-	4	Вычитание	Не является коммутативной, не ассоциативна.
*	5	Умножение	Коммутативность, ассоциативность
/	5	Деление	Не является коммутативной, не ассоциативна.
%	5	Остаток от деления	Не является коммутативной, не ассоциативна.
()	0	Скобки	Определяют приоритет выполнения операций

Операции языка САУ-2024 позволяют выполнять вычисления, сравнения и присваивания, обеспечивая гибкость при работе с данными. Для упрощения анализа выражений важно учитывать их приоритетность и свойства.

1.13 Выражение и их вычисление

Выражение в языке САУ-2024 представляет собой комбинацию операндов (переменных, литералов или значений) и операторов (арифметических, логических и др.), результатом вычисления которого является некоторое значение. Выражения могут быть использованы в различных контекстах программы, таких как присваивание значений переменным, выполнение циклов, вызовы функций.

Вычисление выражений в языке САУ-2024 осуществляется по следующим правилам:

- В одном выражении могут участвовать только операнды одного и того же типа данных.
- Выражение не может содержать вызов функции.
- Возможность использования скобок для смены приоритета.

1.14 Конструкции языка

В языке САУ-2024 предусмотрены следующие конструкции: главная функция, функция, условие и цикл. Ключевые программные конструкции языка программирования САУ-2024 представлены в таблице 1.6.

Таблица 1.6 — Конструкции языка

Конструкция	Представление в языке
Главная функция	<code>main { ... }</code>
Функция	<code><тип данных> func <идентификатор> (<тип данных> <идентификатор>, ...) { ... ret <идентификатор> <литерал>; }</code>
Цикл	<code>while (<условие>) {<инструкция>; }</code>
Условный оператор	<code>if (<условие> {<инструкция>; } else {<инструкция>; }</code>

Эти конструкции составляют основу программной логики в языке САУ-2024, позволяя реализовывать как линейные, так и сложные ветвящиеся алгоритмы. Их использование обеспечивает гибкость при решении широкого спектра задач и упрощает структурирование кода.

1.15 Область видимости идентификаторов

Все идентификаторы, объявленные внутри блока, имеют локальную область видимости. Переменные, объявленные внутри функции, доступны только внутри этой функции, что также относится к её параметрам. Создание пользовательских областей видимости в языке не поддерживается.

1.16 Семантические проверки

Список семантических проверок представлен в таблице 1.7.

Таблица 1.7 — Семантические проверки языка

№	Проверка
1	Наличие одной главной функции main

Окончание таблицы 1.7

2	Переопределение идентификаторов
3	Совместимость типов при присваивании
4	Не допускаются идентификаторы с одинаковым названием
5	Инструкции if и while должны содержать условие
6	Возвращаемое значение функции должно совпадать с типом функции
7	Использование идентификатора до объявления или без него
8	Проверка на максимально допустимую длину литерала (250 символов)
9	Функция обязательно должна возвращать значение
10	Проверка на то, что все функции объявлены перед функцией главной функцией
11	Главная функция не должна возвращать значение
12	В логических операциях должно участвовать два операнда
13	Арифметическое или логическое выражение не должно содержать вызов функции
14	В условии цикла допустимы сравнения операндов только типа int
15	Функция должна возвращать значение в переменную
16	Проверка на вложенность конструкций языка
17	Проверка на рекурсию
18	Проверка на соответствие параметров при вызове функции
19	Проверка на операции со строками
20	Возвращаемым значением из функции не может быть функция или выражение
21	Writeline и write должны содержать значение
22	Деление на ноль
23	В условии if / while не может быть арифметических операций
24	Проверка на возможность бесконечного цикла
25	Наличие главной функции

Семантические проверки обеспечивают соответствие программы логическим правилам языка САУ-2024, предотвращая выполнение некорректных операций. Это повышает надёжность программного обеспечения и помогает выявлять ошибки на этапе компиляции.

1.17 Распределение оперативной памяти на этапе выполнения

В языке САУ-2024 для хранения промежуточных результатов в вычислении выражения используется стек. В сегмент констант записываются все литералы языка. В сегмент данных записываются все имена переменных.

1.18 Стандартная библиотека и её состав

В стандартной библиотеке языка САУ-2024 содержатся функции, представленные в таблице 1.8. Подключение вручную не требуется.

Таблица 1.8 — Функции стандартной библиотеки языка САУ-2024

Функция	Описание
<code>bool isEmpty(string)</code>	Проверяет, является ли строка пустой. Возвращает <code>true</code> , если строка пустая, иначе <code>false</code> .
<code>int rand()</code>	Возвращает случайное целое число.
<code>bool isEven(int)</code>	Проверяет, является ли число четным. Возвращает <code>true</code> , если число четное, иначе <code>false</code> .
<code>int len(string)</code>	Возвращает длину строки (количество символов).
<code>str date()</code>	Возвращает текущую дату в формате дд-мм-гггг

Стандартная библиотека языка САУ-2024 предоставляет базовые функции, упрощающие выполнение часто используемых операций. Все функции доступны без дополнительного подключения.

1.19 Ввод и вывод данных

В языке САУ-2024 стандартный вывод данных осуществляется с использованием ключевых слов `write` и `writeline`. Ключевое слово `write` выводит строку без перевода каретки, а `writeline` — с переводом каретки на новую строку. Вывод может включать значения переменных или строковые литералы, которые заключаются в двойные кавычки

Функция ввода данных в языке САУ-2024 не предусмотрена, что делает его ориентированным исключительно на вывод информации.

1.20 Точка входа

В языке САУ-2024 программа может иметь только одну точку входа, которая определяется наличием функции `main`. Если в коде программы будет определено больше одной или не будет ни одной функции `main`, это приведёт к ошибке на этапе лексического или семантического анализа.

1.21 Препроцессор

Препроцессор — это инструмент для предварительной обработки исходного текста. Он может функционировать как самостоятельная программа или быть частью компилятора. В языке САУ-2024 использование препроцессора не предусмотрено.

1.22 Соглашение о вызовах

Соглашение о вызовах описывает технические детали взаимодействия с подпрограммами, включая: методы передачи параметров подпрограммам; способы их вызова; передачу результатов, полученных подпрограммами, в точку вызова; и методы возврата управления в вызывающий код.

В этом соглашении все параметры передаются в стек вручную, как для вызова функций, так и для выполнения инструкций. Параметры передаются в функцию, начиная с вершины стека: первый параметр будет находиться на вершине стека, а последующие — ниже.

1.23 Объектный код

Целевым языком трансляции выбран ассемблер. Компилятор будет преобразовывать исходный код программы в ассемблерный код, предназначенный для сборки и выполнения на целевой архитектуре процессора.

1.24 Классификация сообщений транслятора

В случае возникновения ошибки в коде программы на языке SAY-2024 и выявления её транслятором в текущий файл протокола выводится сообщение. Их классификация сообщений приведена в таблице 1.9.

Таблица 1.9 – Классификация сообщений транслятора

Диапазон ошибок	Описание
0-99	Системные ошибки.
110-119	Ошибки чтения и открытия файлов.
120-199	Ошибки лексического анализа.
200-299	Ошибки синтаксического анализа.
300-399	Ошибки семантического анализа.
400-999	Зарезервированные ошибки.

Сообщения транслятора языка SAY-2024 помогают выявить и классифицировать ошибки на разных этапах работы программы. Для каждой категории предусмотрен свой диапазон кодов.

1.25 Контрольный пример

Для того, чтобы полностью показать возможности языка SAY-2024 был представлен пример программы, который охватывает все основные конструкции языка. Пример представлен в приложении А.

2. Структура транслятора

2.1 Компоненты транслятора их назначение и принципы взаимодействия

Транслятор — это программа, которая преобразует исходный код, написанный на одном языке программирования, в эквивалентную программу на другом языке [2]. Основная цель транслятора — выполнение программы на компьютере, для которого был написан исходный код. Схема транслятора приведена на рисунке 2.1.

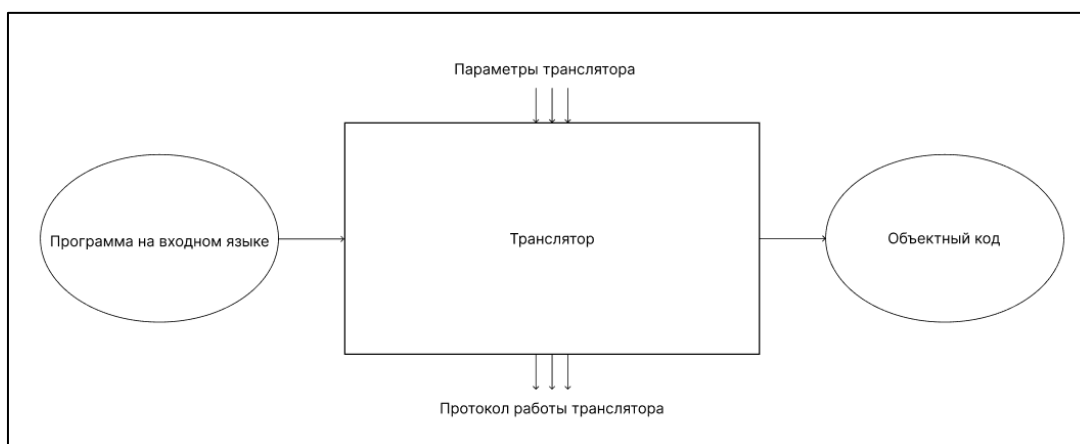


Рисунок 2.1 – Схема транслятора

Транслятор состоит из нескольких основных компонентов, каждый из которых выполняет свои задачи и взаимодействует с другими компонентами [3]. Ниже приведено описание принципа работы транслятора и его основные компоненты:

- Входные данные (код на входном языке) — исходный код программы, написанный на высокоуровневом языке программирования, передается на обработку транслятору.
- Лексический анализатор — на вход принимает исходный код. Разбивает исходный код на лексемы (токены), которые являются минимальными значимыми единицами языка программирования, такими как ключевые слова, идентификаторы, операторы и символы. На выходе мы получаем таблицу лексем и таблицу идентификаторов
- Синтаксический анализатор — принимает последовательность лексем от лексического анализатора и проверяет синтаксическую правильность кода. Если синтаксис корректен, создается синтаксическое дерево (дерево разбора), которое представляет структуру программы.
- Семантический анализатор — анализирует таблицу лексем для семантической корректности программы, таких как типизация переменных, корректность вызовов функций и другие логические проверки. Если обнаружены ошибки, они сообщаются пользователю.
- Генерация кода — принимает таблицу лексем и таблицу идентификаторов. Преобразует таблицу лексем в промежуточный код или непосредственный машинный код. Промежуточный код является более низкоуровневым представлением программы.

- Ассемблер — преобразует промежуточный код в объектный код. Компоновщик объединяет объектные файлы и библиотеки в единую исполняемую программу, готовую к запуску.

2.2 Перечень входных параметров транслятора

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

Таблица 2.1 - Входные параметры транслятора языка САУ-2024

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

Эти параметры позволяют настроить процесс трансляции, обеспечивая корректное взаимодействие между различными этапами анализа. Использование входных параметров делает работу транслятора гибкой и адаптируемой под различные задачи.

2.3 Протоколы, формируемые транслятором

В таблице 2.1 представлены основные параметры, которые задаются на этапе запуска транслятора языка САУ-2024 для управления процессами лексического, синтаксического и семантического анализа. Эти параметры позволяют точно настроить транслятор для обработки исходного кода.

Таблица 2.2 - Протоколы, формируемые транслятором языка САУ-2024

Формируемый протокол	Описание выходного протокола
Журнал работы, задаётся параметром -log:	Содержит отчёт о работе транслятора, включая обработку входных данных (количество символов и строк), протокол синтаксического анализа и ошибки с указанием текста, строки и позиции.

Окончание таблицы 2.2

Файл идентификаторов "IT.txt"	Включает таблицу идентификаторов, созданную на этапе лексического анализа.
Файл лексем "LT.txt"	Содержит таблицу лексем, сформированную во время лексического анализа.
Выходной файл, задаётся параметром -out: с расширением .asm	Результат трансляции, содержащий исходный код на языке ассемблера.

Выходные файлы, формируемые транслятором, обеспечивают полный контроль над всеми этапами обработки исходного кода. Они упрощают отладку, предоставляя разработчику доступ к ключевым данным, включая таблицы идентификаторов, лексем, протоколы анализа и финальный ассемблерный код. Это позволяет эффективно отслеживать процесс компиляции и устранять возникающие ошибки.

- T — разрешённый символ.
- F — запрещённый символ.
- S — сепаратор.

3.3 Удаление избыточных символов

Избыточный символ — это символ, отсутствие которого никоим образом не влияет на правильность и работоспособность исходного текста программы. В языке САУ-2024 такими символами являются пробелы и символы перевода каретки.

Алгоритм удаления избыточных символов:

- Считывание символов: лексический анализатор читает исходный код программы посимвольно. Если встречается символ табуляции, он заменяется пробелом.
- Проверка символов: каждый символ проверяется на допустимость. При встрече сепаратора (разделителя) запись в буфер прекращается и начинается проверка текущей лексемы.
- Удаление или обработка символов: пробелы и символы перевода каретки считаются избыточными и удаляются. При встрече символа перевода каретки увеличивается счетчик строк на единицу.

3.4 Перечень ключевых слов

Ключевые слова языка САУ-2024, сепараторы, символы операций, а также соответствующие им лексемы и регулярные выражения представлены в таблице 3.1.

Таблица 3.1 – Ключевые слова языка САУ-2024

Токен	Лексема	Описание
Идентификатор	i	Переменные и имена функций
Лексема	l	Литералы различных типов
int, str, bool	t	Типы данных: целое число, строка, булевый тип
func	f	Объявление функции
main	m	Главная функция
while	w	Цикл с предусловием
if	z	Условная конструкция
else	e	Альтернативная ветвь условной конструкции
var	v	Объявление переменной
ret	r	Возврат из функции
;	;	Разделитель инструкций
,	,	Разделитель параметров
((Открывающая круглая скобка

Окончание таблицы 3.1

))	Закрывающая круглая скобка
{	{	Открывающая фигурная скобка
}	}	Закрывающая фигурная скобка
+	+	Оператор сложения
-	-	Оператор вычитания
*	*	Оператор умножения
/	/	Оператор деления
%	%	Оператор взятия остатка от деления
=	=	Оператор присваивания
==	&	Оператор сравнения
!=	j	Оператор неравенства
>=	x	Оператор больше или равно
<=	k	Оператор меньше или равно
>	>	Оператор больше
<	<	Оператор меньше
date	date	Функция для работы с датами
rand	rand	Функция генерации случайного числа
len	len	Функция получения длины строки
isEmpty	isEmpty	Функция проверки пустоты строки
isEven	isEven	Функция проверки на четность
write	p	Оператор записи
writeline	s	Оператор записи с переводом строки

Для каждого выражения существует детерминированный конечный автомат, который используется для его анализа. В автомат подается токен, и разбор осуществляется с помощью регулярного выражения, соответствующего

определенному графу переходов. При успешном разборе выражение записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также добавляется в таблицу идентификаторов. На рисунке 3.3 представлена цепочка для объявления функции, ключевое слово `func`.

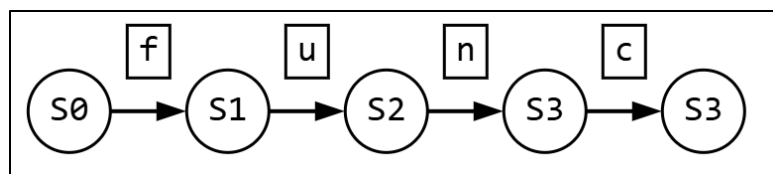


Рисунок 3.3 – Граф для распознавания объявления функции

На рисунке 3.3 представлена цепочка для объявления функции, а именно ключевое слово `var`.

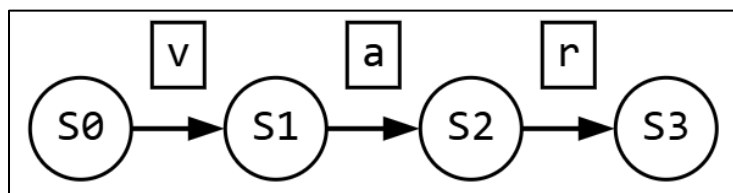


Рисунок 3.4 – Граф для распознавания объявления переменной

Код на C++, который реализует разбор всех цепочек, представлен в приложении Б.

3.5 Основные структуры данных

Основными структурами данных лексического анализатора языка САУ-2024 являются таблица лексем и таблица идентификаторов. Эти таблицы используются для хранения информации о лексемах и идентификаторах, обнаруженных в исходном коде программы.

– Таблица лексем:

1. Лексема: символ или последовательность символов, распознанных анализатором;
2. Номер лексемы: уникальный индекс, присвоенный лексеме во время разбора;
3. Номер строки: строка в исходном коде, где была обнаружена лексема;
4. Позиция в строке: положение символа в строке;
5. Индекс в таблице идентификаторов: ссылка на запись в таблице идентификаторов, если лексема является идентификатором или литералом.

– Таблица идентификаторов:

1. Имя идентификатора: название переменной или функции;
2. Номер в таблице лексем: ссылка на запись в таблице лексем, связанная с данным идентификатором;
3. Тип данных: класс данных, к которому принадлежит идентификатор;

4. Тип идентификатора: категория идентификатора;
5. Значение: текущее значение идентификатора;
6. Область видимости: область действия идентификатора;

Основные структуры таблиц лексем и идентификаторов данных языка программирования САУ-2024, используемых для хранения, представлены в приложении В.

3.6 Структура и перечень сообщений лексического анализатора

Для обработки ошибок лексический анализатор в языке программирования САУ-2024 использует таблицу с сообщениями. Сообщения содержат номер ошибки, вид ошибки, пояснительный текст сообщения, номер строки и позицию в исходном тексте программы, где возникла ошибка.

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 120–199. Текст ошибки содержит префикс «Лексический анализатор». Перечень сообщений лексического анализатора представлен в таблице 3.2.

Таблица 3.2 - Перечень ошибок при лексическом анализе языка САУ-2024

Код ошибки	Описание
120	Лексический анализатор: недопустимый размер таблицы при её создании
121	Лексический анализатор: превышен допустимый размер таблицы при добавлении элемента
122	Лексический анализатор: Недопустимый индекс при получении элемента таблицы
123	Лексический анализатор: недопустимый размер таблицы при её создании
124	Лексический анализатор: превышен допустимый размер таблицы при добавлении элемента
125	Лексический анализатор: Недопустимый индекс при получении элемента таблицы
126	Лексический анализатор: Превышен допустимый размер лексемы
127	Лексический анализатор: нераспознанная лексема

После обнаружения ошибки лексический анализатор немедленно прекращает работу, выводит сообщение об ошибке на консоль и записывает её в протокол.

3.7 Принцип обработки ошибок

В языке программирования САУ-2024, при обнаружении ошибки в исходном коде программы, лексический анализатор формирует сообщение об ошибке. Сообщение об ошибке содержит номер ошибки, описание ошибки, номер строки и позицию в строке, где была допущена ошибка. Затем это сообщение выводится в файл с протоколом работы, который задаётся параметром log, а также на консоль.

При обнаружении ошибки лексический анализатор немедленно прекращает свою работу.

3.8 Параметры лексического анализатора

Лексический анализатор в языке программирования САУ-2024 использует входные параметры, позволяющие ему правильно обрабатывать исходный код и записывать результаты анализа. Эти параметры имеют четкое назначение и принцип применения, обеспечивая эффективное функционирование лексического анализатора.

Параметр `-in` указывает файл, содержащий исходный код на языке программирования САУ-2024, который необходимо проанализировать. Лексический анализатор считывает содержимое указанного файла для дальнейшей обработки и разбора. Путь к файлу задается после параметра `-in`.

Параметр `-log` указывает файл, в который будет записан результат работы лексического анализатора, включая сообщения об ошибках. Лексический анализатор записывает результаты своего анализа и все обнаруженные ошибки в указанный файл. Путь к файлу задается после параметра `-log`.

3.9 Алгоритм лексического анализа

Алгоритм работы лексического анализатора можно описать следующим образом:

1. Считывание исходного кода: Исходный код обрабатывается посимвольно. Каждый символ помещается в строковый буфер до тех пор, пока не будет достигнут символ-разделитель.
2. Анализ с помощью конечных автоматов: Строка передаётся различным конечным автоматам. Если какой-либо автомат успешно разбирает строку, он возвращает одну символьную лексему, которая добавляется в таблицу лексем. Если лексема является идентификатором или литералом, переход к пункту 3; в противном случае возвращаемся к пункту 1.
3. Присвоение типа данных: В зависимости от предыдущих лексем, которые указывают на тип идентификатора, присваивается тип данных идентификатору.
4. Объявление идентификатора: если идентификатор только объявляется с явным указанием типа, происходит его поиск в частично заполненной таблице идентификаторов. Если идентификатор найден, выдается ошибка и анализ прекращается. Если нет, идентификатор добавляется в таблицу идентификаторов и таблицу лексем. Возвращаемся к пункту 1.
5. Использование идентификатора: если идентификатор уже объявлен и просто используется в программе, происходит его поиск в таблице идентификаторов. Если идентификатор не найден, выдается ошибка и анализ прекращается. В противном случае, лексеме присваивается соответствующий индекс таблицы, и она добавляется в таблицу лексем. Возвращаемся к пункту 1.
6. Обработка литералов: если лексема является литералом, определяется её тип и значение. Если такой литерал уже присутствует в таблице идентификаторов,

новая запись не создаётся. В противном случае, литерал добавляется в таблицу идентификаторов. Лексема записывается в таблицу лексем с указанием на таблицу идентификаторов. Возвращаемся к пункту 1.

7. Объявление функции: если идентификатор является функцией, он записывается в таблицу идентификаторов с указанием типа возвращаемого значения. Идентификаторы в круглых скобках записываются как параметры функции. Функция добавляется в стек для отметки области видимости последующих идентификаторов до завершения объявления функции. Возвращаемся к пункту 1.
8. Повторная обработка: если весь исходный код не пройден, алгоритм возвращается к пункту 1.

Этот алгоритм обеспечивает эффективное и структурированное преобразование исходного кода в лексемы, которые затем используются для синтаксического анализа и других этапов компиляции. Реализация данного алгоритма на языке C++ представлена в приложении Г.

3.10 Контрольный пример

Результат работы лексического анализатора, полученный при выполнении контрольного примера, представлен в приложении Д.

4. Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

Синтаксический анализатор является важной частью транслятора, выполняющей синтаксический анализ. Входными данными для него служат таблицы лексем и идентификаторов, а выходными — дерево разбора и информация об обнаруженных ошибках [2]. Эти ошибки выводятся на консоль и записываются в лог-файл, задаваемый параметром -log. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Граф для распознавания объявления переменной

Входными данными для синтаксического анализатора являются таблица лексем и идентификаторов. Выходными данными являются дерево разбора и ошибки, которые выводятся на консоль и в протокол работы транслятора.

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка САУ-2024 используется контекстно-свободная грамматика. Контекстно-свободная грамматика описывает синтаксис языка программирования и задается как четверка $G=\langle T,N,P,S\rangle$, где:

- T — множество терминальных символов (лексемы языка).
- N — множество нетерминальных символов.
- P — множество правил вывода.
- S — стартовый символ.

Алгоритм преобразования грамматик в нормальную форму Грейбах:

- исключить недостижимые символы из грамматики;
- исключить лямбда-правила из грамматики;
- исключить цепные правила.

Грамматика языка САУ-2024 имеет нормальную форму Грейбах, т. к. она не леворекурсивная (не содержит леворекурсивных правил). Полный перечень синтаксических правил в форме Грэйбах перечислены в таблице 4.1.

Таблица 4.1 - Перечень синтаксических правил в форме Грэйбах

Нетерминал	Цепочка	Описание
S	$m\{\} m\{N\} vti; vti;N; vti=E; vti=E;S z(E)\{N\} z(E)\{N\}e\{N\} z(E)\{N\}S z(E)\{N\}e\{N\} $	Стартовый символ, порождает всю структуру исходного кода.

Окончание таблицы 4.1

Нетерминал	Цепочка	Описание
S	$z(E)\{\}\mid z(E)\{N\}e\{\}\mid z(E)\{\}e\{N\}\mid z(E)\{\}S\mid z(E)\{\}e\{N\}S\mid z(E)\{\}e\{\}\mid z(O)\{\}\mid z(O)\{\}S\mid z(O)\{N\}\mid z(O)\{N\}e\{N\}S\mid w(E)\{N\}\mid w(E)\{N\}S\mid w(E)\{\}\mid w(E)\{\}S\mid w(O)\{N\}\mid w(O)\{N\}S\mid w(O)\{\}\mid w(O)\{\}S\mid tfi(O)\{\}\mid tfi(O)\{\}S\mid tfit(O)\{N\}\mid tfi(O)\{N\}S\mid tfi(F)\{N\}\mid tfi(F)\{N\}S$	Стартовый символ, порождает всю структуру исходного кода.
N	$i=E; i=EM; i=E; N i=EM; N vti; vti; N vti=E; vti=EM; vti=E; N vti=EM; N i=i(); i=i(W); i=i(); N i=i(W); N vti=i(); vti=i(W); N z(E)\{N\}\mid z(E)\{N\}e\{N\}\mid z(E)\{N\}N\mid z(E)\{N\}e\{N\}\mid z(E)\{\}\mid z(E)\{N\}e\{\}\mid z(E)\{\}e\{N\}\mid z(E)\{\}N\mid z(E)\{\}e\{N\}N\mid z(E)\{\}e\{\}\mid z(O)\{\}\mid z(O)\{N\}\mid z(O)\{N\}e\{N\}N\mid w(E)\{N\}\mid w(E)\{N\}N\mid w(E)\{\}\mid w(E)\{\}N\mid w(O)\{N\}\mid w(O)\{N\}S\mid w(O)\{\}\mid w(O)\{\}N rE; i(); i(); N i(W); i(W); N p(E); N p(E); s(E); N s(E); p(); N p(); s(); N s();$	Символ для правил, описывающие корректную запись операторов
E	$i l (E) iM lM (E)M i() i(W) i()M i(W)M$	Символ для правил, описывающие корректность записи выражений
M	$+E -E /E *E \%E >E <E &E jE xE kE +EM -EM *EM /EM \%EM$	Символ для правил, описывающие корректность записи подвыражений
F	$ti ti,F$	Символ для правил, описывающие корректность записи параметров функции во время её создания
W	$i l i,W l,W$	Символ для правил, описывающие корректность записи параметров функции при её вызове

Данная грамматика задает синтаксические правила, обеспечивающие корректность структуры исходного кода на языке SAY-2024. Использование нормальной формы Грейбах упрощает синтаксический анализ, позволяя эффективно

выявлять ошибки и трансформировать код в более формализованное представление для последующих этапов трансляции.

4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью для языка САУ-2024 можно представить в виде семерки $M = \{Q, V, Z, \delta, q_0, z_0, F\}$, 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 представлена на рисунке 4.2.

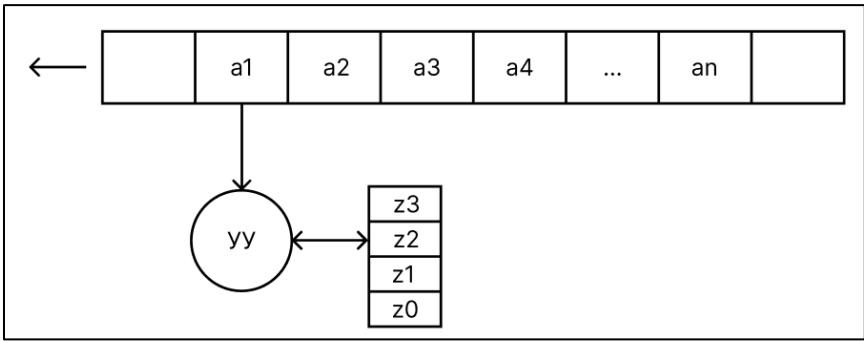


Рисунок 4.2 – Схема работы конечного автомата с магазинной памятью

Описание всех элементов конечного автомата приведено в таблице 4.2.

Таблица 4.2 – Компоненты магазинного автомата

Компонент	Определение	Описание
Q	множество состояний	Указывает текущее правило, обрабатываемую цепочку и содержимое стека
V	алфавит входных символов	Состоит из терминалов и нетерминалов языка
Z	специальный алфавит магазинных символов	Включает стартовый символ и маркер дна стека
δ	функция переходов автомата	Определяются правилами грамматики. Пример указан в таблице 4.1
q_0	начальное состояние автомата	Соответствует стартовому правилу грамматики

Окончание таблицы 4.2

z_0	начальное состояние магазина	Маркер \$, обозначающий дно стека
F	множество конечных состояний	Достигаются при пустом стеке и завершении чтения входной ленты

Алгоритм работы автомата:

1. Автомат начинает с начального состояния q_0 и стека, содержащего символ z_0 .
2. Для каждого входного символа a из алфавита V автомат сверяет текущий символ входной строки с верхним символом стека. На основании функции переходов δ , автомат обновляет свое состояние и модифицирует стек.
3. Автомат завершает работу, если достигается одно из состояний множества F и магазин становится пустым.

4.4 Основные структуры данных

Основными структурами данных, используемыми в синтаксическом анализаторе для языка САУ-2024, являются автомат с магазинной памятью, который обеспечивает обработку входных цепочек и структура грамматики Грейбах, описывающая контекстно-свободные правила языка. Данные структуры представлены в приложении Е.

4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора для языка программирования САУ-2024 описан ниже.

1. В стек автомата помещаются маркер дна (обычно символ \$) и стартовый символ грамматики (например, S).
2. На основании предварительно построенной таблицы лексем формируется входная лента, представляющая последовательность токенов, поступивших от лексического анализатора.
3. Автомат начинает работу, анализируя символы входной ленты и содержимое стека.
4. Текущий символ ленты сравнивается с символом на вершине стека. Если символ на вершине стека является нетерминалом, то из грамматики выбирается правило, соответствующее данному нетерминалу и текущему символу входной ленты. Правая часть выбранного правила записывается в стек в обратном порядке.
5. Если символ на вершине стека совпадает с текущим символом входной ленты, то терминал удаляется из стека, а указатель на входной ленте сдвигается на одну позицию вправо. Если терминалы не совпадают, производится возврат к последнему сохранённому состоянию, и для текущего нетерминала выбирается альтернативное правило (если такое существует).

6. Если в правиле встречается новый нетерминал, анализатор возвращается к шагу 4 для выбора соответствующего правила.

7. Если на вершине стека остаётся только маркер дна (\$), а входная лента полностью обработана, синтаксический анализ считается завершённым успешно. Если условия завершения не выполнены, фиксируется ошибка синтаксического анализа.

Блок-схема, описывающая работу синтаксического анализатора представлена на рисунке 4.3.

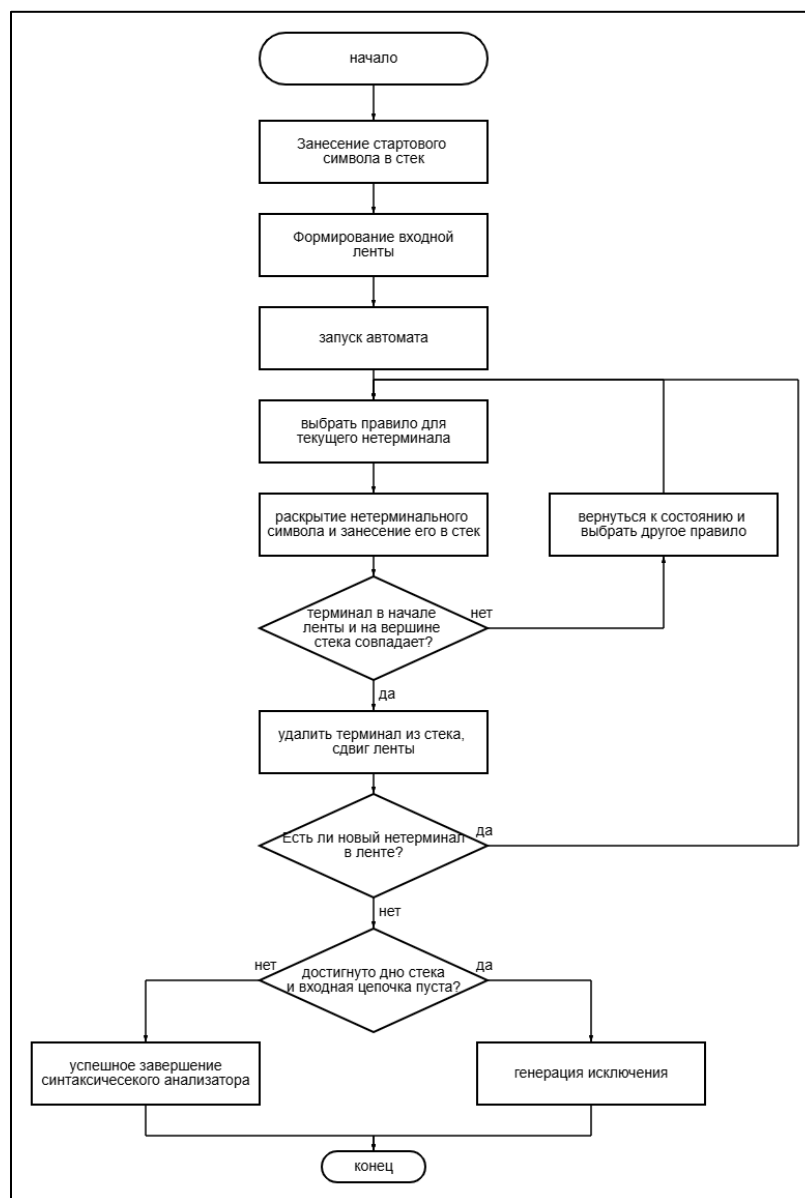


Рисунок 4.3 – Блок-схема работы синтаксического анализатора

Алгоритм обеспечивает эффективный разбор входных данных для языка САУ-2024, обрабатывая как стандартные случаи, так и возможные ошибки.

4.6 Структура и перечень сообщений синтаксического анализатора

Ошибки, выявляемые синтаксическим анализатором, имеют индексы в

диапазоне 200–299. Текст каждой ошибки начинается с префикса «Синтаксический анализатор». Список сообщений синтаксического анализатора для языка программирования САУ-2024 представлен в таблице 4.3.

Таблица 4.3 – Список сообщений синтаксического анализатора для языка программирования САУ-2024

Код ошибки	Описание
200	Синтаксический анализатор: неверная структура программы
201	Синтаксический анализатор: ошибочный оператор
202	Синтаксический анализатор: ошибка в выражении
203	Синтаксический анализатор: ошибка в параметрах функции
204	Синтаксический анализатор: ошибка в параметрах вызываемой функции
205	Синтаксический анализатор: ошибка в подвыражении
206	Синтаксический анализатор: недопустимый оператор
207	Синтаксический анализатор: синтаксический анализ завершён досрочно

Этот список ошибок позволяет разработчикам быстро находить и исправлять синтаксические ошибки в исходном коде, обеспечивая более эффективный процесс разработки программ на языке САУ-2024.

4.7 Параметры синтаксического анализатора и режимы его работы

В языке программирования САУ-2024 специальные параметры для управления режимом работы синтаксического анализатора не предусмотрены. Параметр `-log` указывает файл, в который будет записан результат работы синтаксического анализатора, включая все сообщения об ошибках.

4.8 Принцип обработки ошибок

Обработка ошибок синтаксическим анализатором осуществляется по следующему принципу:

1. При анализе входной последовательности синтаксический анализатор проверяет соответствие каждой конструкции исходного кода правилам грамматики.
2. Если подходящее правило или цепочка не найдены, фиксируется ошибка с указанием её типа и позиции в исходном коде.
3. Все обнаруженные ошибки регистрируются в общей структуре данных для дальнейшей обработки и вывода.
4. По завершении анализа, если были обнаружены ошибки, они выводятся в протокол в порядке их возникновения.

5. Количество выводимых сообщений об ошибках ограничено установленным порогом, чтобы избежать избыточного вывода и повысить удобство работы с диагностикой.

4.9 Контрольный пример

Синтаксический анализатор создает дерево разбора контрольного примера, которое можно найти в приложении А. Протокол анализа и само дерево разбора исходного кода приведены в приложении Ж, а графическое отображение представлено в Графической работе №1. Протокол анализа демонстрирует пошаговый процесс работы конечного автомата с использованием магазинной памяти. В файл, указанный через параметр -log, записываются следующие данные: номер текущего шага, анализируемое правило, состояние входной ленты и содержимое стека.

5. Разработка семантического анализатора
5.1 Структура семантического анализатора

Семантический анализатор отвечает за проверку логической и смысловой структуры программы, гарантируя её соответствие правилам семантики. Это включает: проверку типов данных: анализатор следит за тем, чтобы операции выполнялись над совместимыми типами, контроль областей видимости: проверяется корректность использования переменных и функций в рамках их объявлений, анализ использования операторов и функций, количество аргументов в функциях и так далее. Входными данными являются таблица лексем и идентификаторов. Структура семантического анализатора языка программирования САУ-2024 представлена на рисунке 5.1.



Рисунок 5.1 – Структура семантического анализатора языка САУ-2024

Семантический анализатор обеспечивает контроль корректности логики программы, проверяя соответствие её конструкции правилам языка. Основные задачи включают проверку типов данных, областей видимости и правильности использования функций и операторов.

5.2 Функции семантического анализатора

Семантический анализатор выполняет проверки, направленные на обеспечение корректности программы на уровне её логики и смысла. Эти проверки делятся на фазы выполнения и реализуются специальными функциями.

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

Таблица 5.1 – Семантические проверки языка САУ-2024

Семантическая проверка	Фаза выполнения
Превышение длины строки	Лексический анализ
Превышение целочисленного значения	Лексический анализ
Наличие точки входа в программу	Семантический анализ
Дублирование main	Лексический анализ
Дублирование идентификатора	Лексический анализ

Окончание таблицы 5.1

Дублирование параметра функции	Лексический анализ
Объявление идентификатора перед использованием	Лексический анализ
Количество параметров функции	Семантический анализ
Соответствие типов параметров при вызове функции	Семантический анализ
Соответствие параметров встроенных функций	Семантический анализ
Дублирование функции	Лексический анализ
Соответствие типа возвращаемого значения типу функции	Семантический анализ
Деление на ноль	Семантический анализ
Количество операндов в логическом выражении	Семантический анализ
Вызов функции без присваивания возвращаемого значения в переменную	Семантический анализ
Использование функций в выражениях	Семантический анализ
Количество операндов в условии if и while	Семантический анализ

Эти функции обеспечивают семантическую корректность программы на различных уровнях её логики и структуры.

5.3 Структура и перечень сообщений семантического анализатора

Ошибки, выявляемые семантическим анализатором, имеют индексы в диапазоне 300–399. Текст каждой ошибки начинается с префикса «Семантический анализатор». Список сообщений семантического анализатора для языка программирования САУ-2024 представлен в таблице 5.2.

Таблица 5.2 – Список сообщений семантического анализатора для языка программирования САУ-2024

Код ошибки	Описание
300	Семантический анализатор: отсутствует точка входа в программу
301	Семантический анализатор: Функция уже была определена
302	Семантический анализатор: Дублирование параметров функции
303	Семантический анализатор: Переменная уже была определена
304	Семантический анализатор: Ошибка данных во время присваивания
306	Семантический анализатор: Неверный тип операнда в условии while

Продолжение таблицы 5.2

307	Семантический анализатор: Неверная структура условия if/while, должно быть не более двух операндов
309	Семантический анализатор: Функция ничего не возвращает
310	Семантический анализатор: Тип возвращаемого значения отличается от типа функции
311	Семантический анализатор: Идентификатор не найден в доступной области видимости
313	Семантический анализатор: Условие if/while должно иметь два значения для сравнения
314	Семантический анализатор: Неверное указание кол-ва параметров при вызове функции
315	Семантический анализатор: Неверный тип данных параметров функции при её вызове
316	Семантический анализатор: В сравнении срок должно быть два операнда
317	Семантический анализатор: В условии if должно быть более одного операнда
318	Семантический анализатор: Ошибка, возможен бесконечный цикл
319	Семантический анализатор: В условии цикла while должно быть более одного операнда
320	Семантический анализатор: В условии if/while не должно содержаться знаков операций
321	Семантический анализатор: Функция main уже была определена
322	Семантический анализатор: Функция main не должна ничего возвращать
323	Семантический анализатор: Деление на ноль невозможно
327	Семантический анализатор: Неверный тип параметра функции isEmpty
328	Семантический анализатор: Неверный тип параметра функции isEven
329	Семантический анализатор: Неверный тип параметра функции len
330	Семантический анализатор: Конструкция writeline не должна быть пустой
331	Семантический анализатор: Конструкция write не должна быть пустой
332	Семантический анализатор: Конструкция ret не должна содержать арифметические операции
333	Семантический анализатор: Функцию нельзя использовать в арифметических операциях
334	Семантический анализатор: Неверное присваивание переменной типа bool
335	Семантический анализатор: Операции со строками невозможны

Окончание таблицы 5.2

336	Семантический анализатор: Неверное кол-во параметров при вызове функции isEmpty
337	Семантический анализатор: Неверное кол-во параметров при вызове функции isEven
338	Семантический анализатор: Неверное кол-во параметров при вызове функции len
339	Семантический анализатор: Функцию нельзя использовать в логических операциях
340	Семантический анализатор: Функцию нельзя использовать в условии if/while
341	Семантический анализатор: Конструкция ret не может содержать логические операции
342	Семантический анализатор: Функция не может возвращать функцию
343	Семантический анализатор: Функция не может вызывать саму себя
344	Семантический анализатор: Вложенность невозможна
345	Семантический анализатор: Функция должна возвращать значение в переменную

Эти сообщения об ошибках позволяют разработчикам быстро выявлять и исправлять проблемы в исходном коде, что помогает обеспечивать корректность и надежность программы.

5.4 Принцип обработки ошибок

При обнаружении ошибки в исходном коде программы семантический анализатор формирует сообщение об ошибке 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 Выражения, допускаемые языком

Язык программирования САУ-2024 поддерживает выражения, которые применимы к целочисленным типам данных. В выражениях не разрешается использование функций, возвращающих целочисленные значения. Операции, доступные в языке, а также их приоритетность, перечислены в таблице 6.1.

Таблица 6.1 – Операции и их приоритетность в языке САУ-2024

Операция	Приоритет
(0
)	0
+	4
-	4
*	5
/	5
%	5
>	3
<	3
>=	3
<=	3
==	2
!=	2
,	1

Примеры выражений из контрольного примера: $(\text{randomValue} - 100) * (2 - 1)$ или $\text{result} = \text{result} * \text{base}$.

6.2 Польская запись и принцип ее построения

Обратная польская запись используется для упрощения вычисления выражений при трансляции в язык ассемблера, где все вычисления производятся через стек. Преобразование выражений в ОНП позволяет легко и эффективно генерировать ассемблерный код для вычислений.

Алгоритм построения ОНП выглядит следующим образом:

1. Обрабатываем выражение слева направо;
2. Если символ является литералом или идентификатором, добавляем его к выходной строке;
3. Если символ является открывающей скобкой, помещаем его в стек;
4. Если символ является закрывающей скобкой, то выталкиваем элементы из стека в выходную строку до тех пор, пока верхним элементом стека не станет открывающая скобка. Удаляем открывающую скобку из стека;
5. Если символ является операцией, то пока операция на вершине стека имеет приоритет выше или равен текущей операции, или пока на

вершине стека функция, выталкиваем элементы из стека в выходную строку. Помещаем текущую операцию в стек;

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

Примеры преобразования выражений в обратную польскую нотацию: `il-ll-*|||` или `il-`.

6.3 Программная реализация обработки выражений

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

```

        else if (currentLexema.lexema == LEX_ID ||
currentLexema.lexema == LEX_LITERAL)
        {
            outputQueue.push(currentLexema);
        }
        else if (isOperationSign(currentLexema.lexema))
        {
            while (!operatorsStack.empty() &&
GetPriority(operatorsStack.top().lexema) >=
GetPriority(currentLexema.lexema))
            {
                outputQueue.push(operatorsStack.top());
                operatorsStack.pop();
            }
            operatorsStack.push(currentLexema);
        }
        else if (currentLexema.lexema == LEX_LEFTPAREN)
        {
            if (lextable.table[i - 1].idxTI != -1 &&
idtable.table[lextable.table[i - 1].idxTI].idtype == IT::F)
            {
                outputQueue.push(LT::Entry{ '~',
lextable.table[i].sn, -1 });
            }
            operatorsStack.push(currentLexema);
        }
    }

```

Листинг 6.1 – Фрагмент кода, реализующего преобразование выражений

Если текущая лексема является литералом или идентификатором, то она помещается в выходную строку.

Если текущая лексема – это знак операции, то в выходную строку помещаются элементы из стека операторов и вместе с этим выталкиваются элементы из стека операторов до тех пор, пока приоритет оператора в стеке операторов больше или равен текущему знаку операции.

Если лексема является открывающейся круглой скобкой, то идёт проверка на то, является ли предыдущая лексема идентификатором функции и если это так, то в

стек помещается знак '~' для того, чтобы отметить место начало записи параметров, после чего лексема помещается в стек операторов.

6.4 Контрольный пример

В таблице изображена часть протокола контрольного примера, показывающая результаты преобразования выражений в обратную польскую запись. В первой строке представлено исходное выражение, а во второй строке — преобразованное выражение в обратную польскую запись.

Таблица 6.2 – Преобразование выражений

Выражение	Польская запись
result = result * base	i=ii*
exponent = exponent - 1	i=il-
result = Power(base, exponent)	i=~iii
result > 100	il>
value = (randomValue - 100) * (2 - 1)	i=il-ll-*

Эти примеры демонстрируют процесс преобразования выражений из исходного кода в обратную польскую запись, что упрощает генерацию ассемблерного кода для вычислений.

7. Генерация кода

7.1 Структура генератора кода

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



Рисунок 7.1 – Структура генератора кода

Генератор кода обрабатывает лексемы поочередно, при необходимости запрашивая данные из таблицы идентификаторов. В зависимости от каждой обработанной лексемы создается соответствующий ассемблерный код.

7.2 Представление типов данных в оперативной памяти

Модель памяти в языке САУ-2024 основана на плоской организации, где приложение использует единый непрерывный сегмент для хранения кода и данных. Этот сегмент делится на следующие области: .STACK, .CONST, .DATA, .CODE [1].

- 1. .STACK – стек для данных и функций;
- 2. .CONST – область для хранения литералов (неизменяемых данных);
- 3. .DATA – область для хранения переменных;
- 4. .CODE – область кода программы.

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

Таблица 7.1 – Соответствие типов данных в языке САУ-2024

Тип переменной языка САУ-2024	Тип переменной ассемблера	Описание
int	dword	Знаковый целочисленный тип.
str	dword	Указатель на начало строки.
bool	dword	Логическое значение (0 или 1).

Модель памяти организована так, чтобы эффективно работать с типами данных, используемыми в языке САУ-2024, с учетом их представления в ассемблере.

7.3 Статическая библиотека

Статическая библиотека написана на языке C++. Путь к статической библиотеке указан в Свойства проекта > Компонент > Командная строка. Библиотека подключается на этапе компоновки. Также путь к библиотеке прописан в .asm файле [3].

Функции, входящие в состав статической библиотеки языка САУ-2024, приведены в таблице 7.2.

Таблица 7.2 — Функции стандартной библиотеки языка САУ-2024

Функция	Описание
bool isEmpty(string)	Проверяет, является ли строка пустой. Возвращает true, если строка пустая, иначе false.
int rand()	Возвращает случайное целое число.
bool isEven(int)	Проверяет, является ли число четным. Возвращает true, если число четное, иначе false.
int len(string)	Возвращает длину строки (количество символов).
str date()	Возвращает текущую дату в формате дд-мм-гггг

Эти функции предоставляют основные возможности для работы с данными и числами в языке САУ-2024.

7.4 Особенности алгоритма генерации кода

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

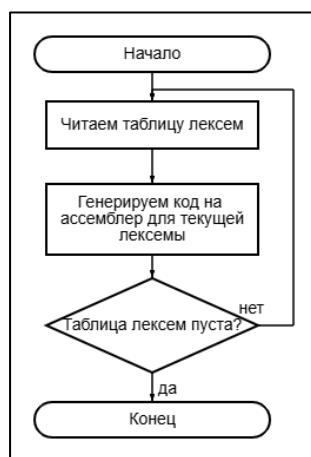


Рисунок 7.2 – Обобщенная блок-схема алгоритма генерации кода языка ассемблера

Пока таблица лексем не опустеет, анализируем каждую лексему. Если для текущей лексемы предусмотрена генерация кода, выполняем её. В противном случае переходим к следующей лексеме и продолжаем процесс до полного опустошения таблицы.

Генерация кода осуществляется с помощью функции `void AsmGeneration (LA::LEX lex, Out::OUT& fout)`. В листинге 7.1 представлен блок макросов, используемый для генерации кода.

```
#define HEAD \
".586P\n" \
".MODEL FLAT, STDCALL\n" \
"includelib libcrt.lib\n" \
"includelib kernel32.lib\n" \
"includelib ../Debug/CAY-2024ASMLIB.lib\n" \
"includelib ../Debug/CAY-2024LIB.lib\n"

#define PROTOTYPES \
"\nExitProcess PROTO : DWORD" \
"\nextrn WriteLineStr : proc" \
"\nextrn WriteBool : proc" \
"\nextrn WriteLineBool : proc" \
"\nextrn dateCAY : proc" \
"\nextrn randCAY : proc" \
"\nextrn isEvenCAY : proc" \
"\nextrn isEmptyCAY : proc" \
"\nextrn lenCAY : proc" \
"\nextrn compareCAY : proc" \
"\n\n.STACK 4096\n\n"
```

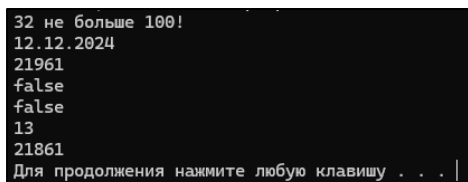
Листинг 7.1 – Применяемые макросы для генерации кода в ассемблер

7.5 Входные параметры, управляющие генерацией кода

На вход генератору кода поступают таблицы лексем и идентификаторов. Результат работы генератора кода выводится в файл, указанный параметром `-out`.

7.6 Контрольный пример

Результат генерации кода на основе контрольного примера из приложения А представлен в приложении 3. На рисунке 7.3 приведен результат работы контрольного примера.



```
32 не больше 100!
12.12.2024
21961
false
false
13
21861
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 7.3 – Результат работы контрольного примера.

Контрольный пример демонстрирует процесс генерации кода и его выполнение, обеспечивая наглядную иллюстрацию работы системы. Результаты, приведённые на рисунке 7.3, показывают корректность работы программной модели в реальных условиях.

8. Тестирование транслятора

8.1 Общие положения

Тестирование предназначено для выявления и устранения ошибок в работе компилятора. Тесты разрабатывались как на ранних, так и на поздних этапах создания компилятора.

При обнаружении ошибки работа транслятора немедленно прекращается, так как ошибка на одном этапе может привести к ошибкам на последующих этапах. Текст с номером и сообщением об ошибке выводится в файл протокола с параметром -log: и на консоль.

8.2 Результаты тестирования

Описание тестовых наборов, демонстрирующих проверки на разных этапах трансляции, представлено в таблице 8.1. Таблица содержит фрагмент исходного кода с ошибкой, соответствующее диагностическое сообщение (код ошибки, этап, текст сообщения), а также место ошибки с указанием строки и позиции в исходном коде.

Таблица 8.1 — Описание тестовых наборов языка САУ-2024

Фрагмент исходного кода	Сообщение
Синтаксический анализ	
var int a; main{ }	201: строка 3, Синтаксический анализатор: ошибочный оператор 200: строка 1, Синтаксический анализатор: неверная структура программы Ошибка 207: Синтаксический анализатор: синтаксический анализ завершён досрочно
main{ } int func Sum(){ }	201: строка 1, Синтаксический анализатор: ошибочный оператор 200: строка 1, Синтаксический анализатор: неверная структура программы Ошибка 207: Синтаксический анализатор: синтаксический анализ завершён досрочно
int func Sum(var int a){ }	204: строка 1, Синтаксический анализатор: ошибка в параметрах вызываемой функции 200: строка 1, Синтаксический анализатор: неверная структура программы Ошибка 207: Синтаксический анализатор: синтаксический анализ завершён досрочно
Семантический анализ	
int func Sum(){ } main{ }	Ошибка 309: Семантический анализатор: Функция ничего не возвращает. Строка:1 Позиция:7
int func Sum(){ ret ""; } main{ }	Ошибка 310: Семантический анализатор: Тип возвращаемого значения отличается от типа функции. Строка:1 Позиция:7

Продолжение таблицы 8.1

<pre>int func Sum(){ ret 1 + 1; } main{ }</pre>	Ошибка 332: Семантический анализатор: Конструкция ret не должна содержать арифметические операции. Строка:1 Позиция:9
<pre>main { if(3 > 2 > 2){ } }</pre>	Ошибка 307: Семантический анализатор: Неверная структура условия if/while, должно быть не более двух операндов. Строка:1 Позиция:10
<pre>main { if(3 == 3){ if(3 == 3){ } } }</pre>	Ошибка 344: Семантический анализатор: Вложенность невозможна. Строка:1 Позиция:8
<pre>int func Sum(int a, int b) { ret 0; } main { var int val = Sum("", 1); }</pre>	Ошибка 315: Семантический анализатор: Неверный тип данных параметров функции при её вызове. Строка:1 Позиция:9
<pre>int func Sum(int a, int b) { ret 0; } main { var int val = Sum(1); }</pre>	Ошибка 314: Семантический анализатор: Неверное указание кол-ва параметров при вызове функции. Строка:1 Позиция:9
<pre>main { var int val = rand(1); }</pre>	Ошибка 324: Семантический анализатор: Неверное кол-во параметров при вызове функции rand. Строка:2 Позиция:7
	Ошибка 300: Семантический анализатор: Отсутствует точка входа в программу.
<pre>main { var int val = ""; }</pre>	Ошибка 304: Семантический анализатор: Ошибка данных во время присваивания. Строка:2 Позиция:5
<pre>main { var str val = "" + ""; }</pre>	Ошибка 335: Семантический анализатор: Операции со строками невозможны. Строка:2 Позиция:7
<pre>main { var int val = 5 / 0; }</pre>	Ошибка 323: Семантический анализатор: Деление на ноль невозможно. Строка:2 Позиция:6
<pre>main { while(true){ } }</pre>	Ошибка 319: Семантический анализатор: В условии цикла while должно быть более одного операнда. Строка:2 Позиция:5
<pre>main { while(true == true){ } }</pre>	Ошибка 306: Семантический анализатор: Неверный тип операнда в условии while. Строка:2 Позиция:7

Окончание таблицы 8.1

main { while(3 == 3){} }	Ошибка 318: Семантический анализатор: Ошибка, возможен бесконечный цикл. Строка:2 Позиция:7
int func Sum() { var int val = Sum(); ret 0; } main {}	Ошибка 343: Семантический анализатор: Функция не может вызывать саму себя. Строка:3 Позиция:6
int func Add() { ret 0; } int func Sum() { ret Add(); } main {}	Ошибка 342: Семантический анализатор: Функция не может возвращать функцию. Строка:7 Позиция:3
int func Add() { ret 0 == 0; } main {}	Ошибка 341: Семантический анализатор: Конструкция ret не может содержать логические операции. Строка:3 Позиция:4
Лексический анализ	
main{ } main{ }	Ошибка 321: Семантический анализатор: Функция main уже была определена. Строка:2 Позиция:0
int func Sum(){ } int func Sum(){ } main { }	Ошибка 301: Семантический анализатор: Функция уже была определена. Строка:2 Позиция:2
main { var int val = rand(); var int val; }	Ошибка 303: Семантический анализатор: Переменная уже была определена. Строка:3 Позиция:2
main { var int i; i = 0; i = 0; i = 0; i = 0; ... i = 0; }	Ошибка 121: Таблица лексем: превышен допустимый размер таблицы при добавлении элемента

Эти результаты тестирования подтверждают корректность работы компилятора на разных этапах трансляции. Тесты были успешно выполнены и продемонстрировали способность компилятора обнаруживать и обрабатывать ошибки на различных уровнях анализа.

Заключение

В рамках курсовой работы был создан компилятор для языка САУ-2024, соответствующий минимальным требованиям проекта с рядом дополнений. Язык САУ-2024 включает следующие элементы:

1. 3 типа данных;
2. Условный оператор if/else;
3. Оператор цикла while;
4. 5 арифметических операторов;
5. 6 логических операторов;
6. 5 функций стандартной библиотеки;
7. Операторы для вывода данных в консоль.

Процесс разработки компилятора способствовал глубокому пониманию структур и процессов, применяемых при создании компиляторов. Также были изучены основные аспекты теории формальных грамматик и общей теории компиляторов.

Работа над проектом позволила не только достичь поставленных целей, но и значительно расширить знания в области разработки языков программирования и компиляторов, что является ценным опытом для дальнейшего профессионального роста.

Список использованных литературных источников

- 1) Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – М.: Вильямс, 2005. – 912с.
- 2) Курс лекций по КПО / Наркевич А.С.
- 3) Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с.

Приложение А

```

int func Power(int base, int exponent)
{
    var int result = 1;
    while(exponent > 0)
    {
        result = result * base;
        exponent = exponent - 1;
    }
    ret result;
}

main
{
    var int base = 2;
    var int exponent = 5;
    var int result = Power(base, exponent);

    if(result > 100)
    {
        write(result);
        writeline(" больше 100!");
    }
    else
    {
        write(result);
        writeline(" не больше 100!");
    }

    var str currentDate = date();
    var int randomValue = rand();
    var bool isEvenValue = isEven(11);
    var bool isEmptyValue = isEmpty("Hello!");
    var int textLength = len("Hello, World!");

    var int value = (randomValue - 100) * (2 - 1);
}

```

Листинг 1 – Контрольный пример на языке САУ-2024

```

int func Power(int base, int exponent)
{
    var int result = 1;
    while(exponent > 0)
    {
        result = result * base;
        exponent = exponent - 1;
    }
    ret result;
}

main
{

```



```

var int base = 2;
var int exponent = 5;
var int result = Power(base, exponent, base);

if(result > 100 > 10)
{
    write(result);
    writeline(" больше 100!");
}
else
{
    write(result);
    writeline(" не больше 100!");
}

var str currentDate = date() + "Hello";
var int randomValue = rand();
var bool isEvenValue = isEven(11);
var bool isEmptyValue = isEmpty("Hello!");
var int textLength = len("Hello, World!");

var int value = (randomValue - 100) * (2 - 1);
}

```

Листинг 2 – Контрольный пример с 3-мя семантическими ошибками
Допущенный ошибки:

- Ошибка 314: Семантический анализатор: Неверное указание кол-ва параметров при вызове функции. Строка:1 Позиция:9
- Ошибка 307: Семантический анализатор: Неверная структура условия if/while, должно быть не более двух операндов. Строка:18 Позиция:9
- Ошибка 335: Семантический анализатор: Операции со строками невозможны. Строка:29 Позиция:8

Приложение Б

```

#define FST_MAIN FST::FST _main(
    str,
    5,
    FST::NODE(1, FST::RELATION('m', 1)),
    FST::NODE(1, FST::RELATION('a', 2)),
    FST::NODE(1, FST::RELATION('i', 3)),
    FST::NODE(1, FST::RELATION('n', 4)),
    FST::NODE()
)

#define FST_INT FST::FST _int(
    str,
    4,
    FST::NODE(1, FST::RELATION('i', 1)),
    FST::NODE(1, FST::RELATION('n', 2)),
    FST::NODE(1, FST::RELATION('t', 3)),
    FST::NODE()
)

#define FST_STR FST::FST _string(
    str,
    4,
    FST::NODE(1, FST::RELATION('s', 1)),
    FST::NODE(1, FST::RELATION('t', 2)),
    FST::NODE(1, FST::RELATION('r', 3)),
    FST::NODE()
)

#define FST_BOOL FST::FST _bool(
    str,
    5,
    FST::NODE(1, FST::RELATION('b', 1)),
    FST::NODE(1, FST::RELATION('o', 2)),
    FST::NODE(1, FST::RELATION('o', 3)),
    FST::NODE(1, FST::RELATION('l', 4)),
    FST::NODE()
)

#define FST_TRUE FST::FST _true(
    str,
    5,
    FST::NODE(1, FST::RELATION('t', 1)),
    FST::NODE(1, FST::RELATION('r', 2)),
    FST::NODE(1, FST::RELATION('u', 3)),
    FST::NODE(1, FST::RELATION('e', 4)),
    FST::NODE()
)

#define FST_FALSE FST::FST _false(
    str,
    6,
    FST::NODE(1, FST::RELATION('f', 1)),

```

```

        FST::NODE(1, FST::RELATION('a', 2)), \
        FST::NODE(1, FST::RELATION('l', 3)), \
        FST::NODE(1, FST::RELATION('s', 4)), \
        FST::NODE(1, FST::RELATION('e', 5)), \
        FST::NODE() \
    )

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

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

#define FST_VARIABLE FST::FST _variable( \
    str, \
    4, \
    FST::NODE(1, FST::RELATION('v', 1)), \
    FST::NODE(1, FST::RELATION('a', 2)), \
    FST::NODE(1, FST::RELATION('r', 3)), \
    FST::NODE() \
)

#define FST_WHILE FST::FST _while( \
    str, \
    6, \
    FST::NODE(1, FST::RELATION('w', 1)), \
    FST::NODE(1, FST::RELATION('h', 2)), \
    FST::NODE(1, FST::RELATION('i', 3)), \
    FST::NODE(1, FST::RELATION('l', 4)), \
    FST::NODE(1, FST::RELATION('e', 5)), \
    FST::NODE() \
)

#define FST_IF FST::FST _if( \
    str, \
    3, \
    FST::NODE(1, FST::RELATION('i', 1)), \
    FST::NODE(1, FST::RELATION('f', 2)), \
    FST::NODE() \
)

```

```

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

#define FST_WRITE FST::FST _write(        \
    str,                                  \
    6,                                    \
    FST::NODE(1, FST::RELATION('w', 1)), \
    FST::NODE(1, FST::RELATION('r', 2)), \
    FST::NODE(1, FST::RELATION('i', 3)), \
    FST::NODE(1, FST::RELATION('t', 4)), \
    FST::NODE(1, FST::RELATION('e', 5)), \
    FST::NODE()                          \
)

#define FST_WRITELINE FST::FST _writeline( \
    str,                                  \
    10,                                   \
    FST::NODE(1, FST::RELATION('w', 1)), \
    FST::NODE(1, FST::RELATION('r', 2)), \
    FST::NODE(1, FST::RELATION('i', 3)), \
    FST::NODE(1, FST::RELATION('t', 4)), \
    FST::NODE(1, FST::RELATION('e', 5)), \
    FST::NODE(1, FST::RELATION('l', 6)), \
    FST::NODE(1, FST::RELATION('i', 7)), \
    FST::NODE(1, FST::RELATION('n', 8)), \
    FST::NODE(1, FST::RELATION('e', 9)), \
    FST::NODE()                          \
)

#define FST_ISEVEN FST::FST _iseven(      \
    str,                                  \
    7,                                    \
    FST::NODE(1, FST::RELATION('i', 1)), \
    FST::NODE(1, FST::RELATION('s', 2)), \
    FST::NODE(1, FST::RELATION('E', 3)), \
    FST::NODE(1, FST::RELATION('v', 4)), \
    FST::NODE(1, FST::RELATION('e', 5)), \
    FST::NODE(1, FST::RELATION('n', 6)), \
    FST::NODE()                          \
)

#define FST_DATE FST::FST _date(          \
    str,                                  \
    5,                                    \
    FST::NODE(1, FST::RELATION('d', 1)), \
    FST::NODE(1, FST::RELATION('a', 2)), \
    FST::NODE(1, FST::RELATION('t', 3)), \

```

```

        FST::NODE(1, FST::RELATION('e', 4)), \
        FST::NODE() \
    )

#define FST_RAND FST::FST _rand( \
    str, \
    5, \
    FST::NODE(1, FST::RELATION('r', 1)), \
    FST::NODE(1, FST::RELATION('a', 2)), \
    FST::NODE(1, FST::RELATION('n', 3)), \
    FST::NODE(1, FST::RELATION('d', 4)), \
    FST::NODE() \
)

#define FST_LEN FST::FST _len( \
    str, \
    4, \
    FST::NODE(1, FST::RELATION('l', 1)), \
    FST::NODE(1, FST::RELATION('e', 2)), \
    FST::NODE(1, FST::RELATION('n', 3)), \
    FST::NODE() \
)

#define FST_IEMPTY FST::FST _isempty( \
    str, \
    8, \
    FST::NODE(1, FST::RELATION('i', 1)), \
    FST::NODE(1, FST::RELATION('s', 2)), \
    FST::NODE(1, FST::RELATION('E', 3)), \
    FST::NODE(1, FST::RELATION('m', 4)), \
    FST::NODE(1, FST::RELATION('p', 5)), \
    FST::NODE(1, FST::RELATION('t', 6)), \
    FST::NODE(1, FST::RELATION('y', 7)), \
    FST::NODE() \
)

#define FST_INT_BIN_LITERAL FST::FST _int_bin_literal( \
    str, \
    6, \
    FST::NODE(3, FST::RELATION('-', 1), FST::RELATION('0', 2), \
FST::RELATION('1', 2)), \
    FST::NODE(2, FST::RELATION('0', 2), FST::RELATION('1', \
2)), \
    FST::NODE(1, FST::RELATION('b', 3)), \
    FST::NODE(2, FST::RELATION('0', 4), FST::RELATION('1', \
4)), \
    FST::NODE(4, FST::RELATION('0', 5), FST::RELATION('1', 5), \
    FST::RELATION('0', 4), FST::RELATION('1', 4)), \
    \

```

```

        FST::NODE()
\
)

#define FST_HEX_BIN_LITERAL FST::FST _int_hex_literal(
\
    str,
\
    5,
\
    FST::NODE(2, FST::RELATION('-', 1), FST::RELATION('0',
2)),
\
    FST::NODE(1, FST::RELATION('0', 2)),
\
    FST::NODE(1, FST::RELATION('x', 3)),
\
    FST::NODE(44,
\
        FST::RELATION('1', 4), FST::RELATION('1', 3),
FST::RELATION('A', 4), FST::RELATION('A', 3), FST::RELATION('e', 4),
FST::RELATION('e', 3), \
        FST::RELATION('2', 4), FST::RELATION('2', 3),
FST::RELATION('B', 4), FST::RELATION('B', 3), FST::RELATION('f', 4),
FST::RELATION('f', 3), \
        FST::RELATION('3', 4), FST::RELATION('3', 3),
FST::RELATION('C', 4), FST::RELATION('C', 3),
\
        FST::RELATION('4', 4), FST::RELATION('4', 3),
FST::RELATION('D', 4), FST::RELATION('D', 3),
\
        FST::RELATION('5', 4), FST::RELATION('5', 3),
FST::RELATION('E', 4), FST::RELATION('E', 3),
\
        FST::RELATION('6', 4), FST::RELATION('6', 3),
FST::RELATION('F', 4), FST::RELATION('F', 3),
\
        FST::RELATION('7', 4), FST::RELATION('7', 3),
FST::RELATION('a', 4), FST::RELATION('a', 3),
\
        FST::RELATION('8', 4), FST::RELATION('8', 3),
FST::RELATION('b', 4), FST::RELATION('b', 3),
\
        FST::RELATION('9', 4), FST::RELATION('9', 3),
FST::RELATION('c', 4), FST::RELATION('c', 3),
\
        FST::RELATION('0', 4), FST::RELATION('0', 3),
FST::RELATION('d', 4), FST::RELATION('d', 3)
\
    ),
\
    FST::NODE()
\

```

```

)
#define FST_INT_LITERAL FST::FST _int_literal(
\
    str,
\
    3,
\
    FST::NODE(21, FST::RELATION('-', 1),
\
        FST::RELATION('1', 1), FST::RELATION('1', 2),
\
        FST::RELATION('2', 1), FST::RELATION('2', 2),
\
        FST::RELATION('3', 1), FST::RELATION('3', 2),
\
        FST::RELATION('4', 1), FST::RELATION('4', 2),
\
        FST::RELATION('5', 1), FST::RELATION('5', 2),
\
        FST::RELATION('6', 1), FST::RELATION('6', 2),
\
        FST::RELATION('7', 1), FST::RELATION('7', 2),
\
        FST::RELATION('8', 1), FST::RELATION('8', 2),
\
        FST::RELATION('9', 1), FST::RELATION('9', 2),
\
        FST::RELATION('0', 1), FST::RELATION('0', 2)
\
    ),
\
    FST::NODE(20, FST::RELATION('0', 2), FST::RELATION('0', 1),
\
        FST::RELATION('1', 2), FST::RELATION('1', 1),
\
        FST::RELATION('2', 2), FST::RELATION('2', 1),
\
        FST::RELATION('3', 2), FST::RELATION('3', 1),
\
        FST::RELATION('4', 2), FST::RELATION('4', 1),
\
        FST::RELATION('5', 2), FST::RELATION('5', 1),
\
        FST::RELATION('6', 2), FST::RELATION('6', 1),
\
        FST::RELATION('7', 2), FST::RELATION('7', 1),
\
        FST::RELATION('8', 2), FST::RELATION('8', 1),
\
        FST::RELATION('9', 2), FST::RELATION('9', 1)),
\
    FST::NODE()
\

```

```

)
#define FST_IDENTIFIER FST::FST _identifier(
    \
    str,
    \
    3,
    \
    FST::NODE(104,
    \
        FST::RELATION('A', 1), FST::RELATION('A', 2),
    \
        FST::RELATION('B', 1), FST::RELATION('B', 2),
    \
        FST::RELATION('C', 1), FST::RELATION('C', 2),
    \
        FST::RELATION('D', 1), FST::RELATION('D', 2),
    \
        FST::RELATION('E', 1), FST::RELATION('E', 2),
    \
        FST::RELATION('F', 1), FST::RELATION('F', 2),
    \
        FST::RELATION('G', 1), FST::RELATION('G', 2),
    \
        FST::RELATION('H', 1), FST::RELATION('H', 2),
    \
        FST::RELATION('I', 1), FST::RELATION('I', 2),
    \
        FST::RELATION('J', 1), FST::RELATION('J', 2),
    \
        FST::RELATION('K', 1), FST::RELATION('K', 2),
    \
        FST::RELATION('L', 1), FST::RELATION('L', 2),
    \
        FST::RELATION('M', 1), FST::RELATION('M', 2),
    \
        FST::RELATION('N', 1), FST::RELATION('N', 2),
    \
        FST::RELATION('O', 1), FST::RELATION('O', 2),
    \
        FST::RELATION('P', 1), FST::RELATION('P', 2),
    \
        FST::RELATION('Q', 1), FST::RELATION('Q', 2),
    \
        FST::RELATION('R', 1), FST::RELATION('R', 2),
    \
        FST::RELATION('S', 1), FST::RELATION('S', 2),
    \
        FST::RELATION('T', 1), FST::RELATION('T', 2),
    \
        FST::RELATION('U', 1), FST::RELATION('U', 2),
    \
        FST::RELATION('V', 1), FST::RELATION('V', 2),
    \

```



```

FST::RELATION('w', 1), FST::RELATION('w', 2),
\
FST::RELATION('x', 1), FST::RELATION('x', 2),
\
FST::RELATION('y', 1), FST::RELATION('y', 2),
\
FST::RELATION('z', 1), FST::RELATION('z', 2),
\
FST::RELATION('a', 1), FST::RELATION('a', 2),
\
FST::RELATION('b', 1), FST::RELATION('b', 2),
\
FST::RELATION('c', 1), FST::RELATION('c', 2),
\
FST::RELATION('d', 1), FST::RELATION('d', 2),
\
FST::RELATION('e', 1), FST::RELATION('e', 2),
\
FST::RELATION('f', 1), FST::RELATION('f', 2),
\
FST::RELATION('g', 1), FST::RELATION('g', 2),
\
FST::RELATION('h', 1), FST::RELATION('h', 2),
\
FST::RELATION('i', 1), FST::RELATION('i', 2),
\
FST::RELATION('j', 1), FST::RELATION('j', 2),
\
FST::RELATION('k', 1), FST::RELATION('k', 2),
\
FST::RELATION('l', 1), FST::RELATION('l', 2),
\
FST::RELATION('m', 1), FST::RELATION('m', 2),
\
FST::RELATION('n', 1), FST::RELATION('n', 2),
\
FST::RELATION('o', 1), FST::RELATION('o', 2),
\
FST::RELATION('p', 1), FST::RELATION('p', 2),
\
FST::RELATION('q', 1), FST::RELATION('q', 2),
\
FST::RELATION('r', 1), FST::RELATION('r', 2),
\
FST::RELATION('s', 1), FST::RELATION('s', 2),
\
FST::RELATION('t', 1), FST::RELATION('t', 2),
\
FST::RELATION('u', 1), FST::RELATION('u', 2),
\
FST::RELATION('v', 1), FST::RELATION('v', 2),
\
FST::RELATION('w', 1), FST::RELATION('w', 2),
\

```

```

\      FST::RELATION('x', 1), FST::RELATION('x', 2),
\
\      FST::RELATION('y', 1), FST::RELATION('y', 2),
\
\      FST::RELATION('z', 1), FST::RELATION('z', 2)),
\
\      FST::NODE(126,
\
\      FST::RELATION('A', 1), FST::RELATION('A', 2),
\
\      FST::RELATION('B', 1), FST::RELATION('B', 2),
\
\      FST::RELATION('C', 1), FST::RELATION('C', 2),
\
\      FST::RELATION('D', 1), FST::RELATION('D', 2),
\
\      FST::RELATION('E', 1), FST::RELATION('E', 2),
\
\      FST::RELATION('F', 1), FST::RELATION('F', 2),
\
\      FST::RELATION('G', 1), FST::RELATION('G', 2),
\
\      FST::RELATION('H', 1), FST::RELATION('H', 2),
\
\      FST::RELATION('I', 1), FST::RELATION('I', 2),
\
\      FST::RELATION('J', 1), FST::RELATION('J', 2),
\
\      FST::RELATION('K', 1), FST::RELATION('K', 2),
\
\      FST::RELATION('L', 1), FST::RELATION('L', 2),
\
\      FST::RELATION('M', 1), FST::RELATION('M', 2),
\
\      FST::RELATION('N', 1), FST::RELATION('N', 2),
\
\      FST::RELATION('O', 1), FST::RELATION('O', 2),
\
\      FST::RELATION('P', 1), FST::RELATION('P', 2),
\
\      FST::RELATION('Q', 1), FST::RELATION('Q', 2),
\
\      FST::RELATION('R', 1), FST::RELATION('R', 2),
\
\      FST::RELATION('S', 1), FST::RELATION('S', 2),
\
\      FST::RELATION('T', 1), FST::RELATION('T', 2),
\
\      FST::RELATION('U', 1), FST::RELATION('U', 2),
\
\      FST::RELATION('V', 1), FST::RELATION('V', 2),
\
\      FST::RELATION('W', 1), FST::RELATION('W', 2),
\

```

```

\      FST::RELATION('x', 1), FST::RELATION('x', 2),
\
\      FST::RELATION('y', 1), FST::RELATION('y', 2),
\
\      FST::RELATION('z', 1), FST::RELATION('z', 2),
\
\      FST::RELATION('a', 1), FST::RELATION('a', 2),
\
\      FST::RELATION('b', 1), FST::RELATION('b', 2),
\
\      FST::RELATION('c', 1), FST::RELATION('c', 2),
\
\      FST::RELATION('d', 1), FST::RELATION('d', 2),
\
\      FST::RELATION('e', 1), FST::RELATION('e', 2),
\
\      FST::RELATION('f', 1), FST::RELATION('f', 2),
\
\      FST::RELATION('g', 1), FST::RELATION('g', 2),
\
\      FST::RELATION('h', 1), FST::RELATION('h', 2),
\
\      FST::RELATION('i', 1), FST::RELATION('i', 2),
\
\      FST::RELATION('j', 1), FST::RELATION('j', 2),
\
\      FST::RELATION('k', 1), FST::RELATION('k', 2),
\
\      FST::RELATION('l', 1), FST::RELATION('l', 2),
\
\      FST::RELATION('m', 1), FST::RELATION('m', 2),
\
\      FST::RELATION('n', 1), FST::RELATION('n', 2),
\
\      FST::RELATION('o', 1), FST::RELATION('o', 2),
\
\      FST::RELATION('p', 1), FST::RELATION('p', 2),
\
\      FST::RELATION('q', 1), FST::RELATION('q', 2),
\
\      FST::RELATION('r', 1), FST::RELATION('r', 2),
\
\      FST::RELATION('s', 1), FST::RELATION('s', 2),
\
\      FST::RELATION('t', 1), FST::RELATION('t', 2),
\
\      FST::RELATION('u', 1), FST::RELATION('u', 2),
\
\      FST::RELATION('v', 1), FST::RELATION('v', 2),
\
\      FST::RELATION('w', 1), FST::RELATION('w', 2),
\
\      FST::RELATION('x', 1), FST::RELATION('x', 2),
\

```

```

FST::RELATION('y', 1), FST::RELATION('y', 2),
\
FST::RELATION('z', 1), FST::RELATION('z', 2),
\
FST::RELATION('0', 1), FST::RELATION('0', 2),
\
FST::RELATION('1', 1), FST::RELATION('1', 2),
\
FST::RELATION('2', 1), FST::RELATION('2', 2),
\
FST::RELATION('3', 1), FST::RELATION('3', 2),
\
FST::RELATION('4', 1), FST::RELATION('4', 2),
\
FST::RELATION('5', 1), FST::RELATION('5', 2),
\
FST::RELATION('6', 1), FST::RELATION('6', 2),
\
FST::RELATION('7', 1), FST::RELATION('7', 2),
\
FST::RELATION('8', 1), FST::RELATION('8', 2),
\
FST::RELATION('9', 1), FST::RELATION('9', 2),
\
FST::RELATION('_', 1), FST::RELATION('_', 2)),
\
FST::NODE()
\
)

#define FST_STRING_LITERAL FST::FST _string_literal(
\
    str,
\
    4,
\
    FST::NODE(2, FST::RELATION('\\"', 1), FST::RELATION('\\"',
2)), \
    FST::NODE(280,
\
    FST::RELATION('A', 1), FST::RELATION('A', 2),
\
    FST::RELATION('B', 1), FST::RELATION('B', 2),
\
    FST::RELATION('C', 1), FST::RELATION('C', 2),
\
    FST::RELATION('D', 1), FST::RELATION('D', 2),
\
    FST::RELATION('E', 1), FST::RELATION('E', 2),
\
    FST::RELATION('F', 1), FST::RELATION('F', 2),
\
    FST::RELATION('G', 1), FST::RELATION('G', 2),
\

```

```

\      FST::RELATION('H', 1), FST::RELATION('H', 2),
\
\      FST::RELATION('I', 1), FST::RELATION('I', 2),
\
\      FST::RELATION('J', 1), FST::RELATION('J', 2),
\
\      FST::RELATION('K', 1), FST::RELATION('K', 2),
\
\      FST::RELATION('L', 1), FST::RELATION('L', 2),
\
\      FST::RELATION('M', 1), FST::RELATION('M', 2),
\
\      FST::RELATION('N', 1), FST::RELATION('N', 2),
\
\      FST::RELATION('O', 1), FST::RELATION('O', 2),
\
\      FST::RELATION('P', 1), FST::RELATION('P', 2),
\
\      FST::RELATION('Q', 1), FST::RELATION('Q', 2),
\
\      FST::RELATION('R', 1), FST::RELATION('R', 2),
\
\      FST::RELATION('S', 1), FST::RELATION('S', 2),
\
\      FST::RELATION('T', 1), FST::RELATION('T', 2),
\
\      FST::RELATION('U', 1), FST::RELATION('U', 2),
\
\      FST::RELATION('V', 1), FST::RELATION('V', 2),
\
\      FST::RELATION('W', 1), FST::RELATION('W', 2),
\
\      FST::RELATION('X', 1), FST::RELATION('X', 2),
\
\      FST::RELATION('Y', 1), FST::RELATION('Y', 2),
\
\      FST::RELATION('Z', 1), FST::RELATION('Z', 2),
\
\      FST::RELATION('a', 1), FST::RELATION('a', 2),
\
\      FST::RELATION('b', 1), FST::RELATION('b', 2),
\
\      FST::RELATION('c', 1), FST::RELATION('c', 2),
\
\      FST::RELATION('d', 1), FST::RELATION('d', 2),
\
\      FST::RELATION('e', 1), FST::RELATION('e', 2),
\
\      FST::RELATION('f', 1), FST::RELATION('f', 2),
\
\      FST::RELATION('g', 1), FST::RELATION('g', 2),
\
\      FST::RELATION('h', 1), FST::RELATION('h', 2),
\

```

```

\      FST::RELATION('i', 1), FST::RELATION('i', 2),
\
\      FST::RELATION('j', 1), FST::RELATION('j', 2),
\
\      FST::RELATION('k', 1), FST::RELATION('k', 2),
\
\      FST::RELATION('l', 1), FST::RELATION('l', 2),
\
\      FST::RELATION('m', 1), FST::RELATION('m', 2),
\
\      FST::RELATION('n', 1), FST::RELATION('n', 2),
\
\      FST::RELATION('o', 1), FST::RELATION('o', 2),
\
\      FST::RELATION('p', 1), FST::RELATION('p', 2),
\
\      FST::RELATION('q', 1), FST::RELATION('q', 2),
\
\      FST::RELATION('r', 1), FST::RELATION('r', 2),
\
\      FST::RELATION('s', 1), FST::RELATION('s', 2),
\
\      FST::RELATION('t', 1), FST::RELATION('t', 2),
\
\      FST::RELATION('u', 1), FST::RELATION('u', 2),
\
\      FST::RELATION('v', 1), FST::RELATION('v', 2),
\
\      FST::RELATION('w', 1), FST::RELATION('w', 2),
\
\      FST::RELATION('x', 1), FST::RELATION('x', 2),
\
\      FST::RELATION('y', 1), FST::RELATION('y', 2),
\
\      FST::RELATION('z', 1), FST::RELATION('z', 2),
\
\      FST::RELATION('A', 1), FST::RELATION('A', 2),
\
\      FST::RELATION('B', 1), FST::RELATION('B', 2),
\
\      FST::RELATION('B', 1), FST::RELATION('B', 2),
\
\      FST::RELATION('Г', 1), FST::RELATION('Г', 2),
\
\      FST::RELATION('Д', 1), FST::RELATION('Д', 2),
\
\      FST::RELATION('E', 1), FST::RELATION('E', 2),
\
\      FST::RELATION('Ё', 1), FST::RELATION('Ё', 2),
\
\      FST::RELATION('Ж', 1), FST::RELATION('Ж', 2),
\
\      FST::RELATION('З', 1), FST::RELATION('З', 2),
\

```

```

FST::RELATION('И', 1), FST::RELATION('И', 2),
\
FST::RELATION('Й', 1), FST::RELATION('Й', 2),
\
FST::RELATION('К', 1), FST::RELATION('К', 2),
\
FST::RELATION('Л', 1), FST::RELATION('Л', 2),
\
FST::RELATION('М', 1), FST::RELATION('М', 2),
\
FST::RELATION('Н', 1), FST::RELATION('Н', 2),
\
FST::RELATION('О', 1), FST::RELATION('О', 2),
\
FST::RELATION('П', 1), FST::RELATION('П', 2),
\
FST::RELATION('Р', 1), FST::RELATION('Р', 2),
\
FST::RELATION('С', 1), FST::RELATION('С', 2),
\
FST::RELATION('Т', 1), FST::RELATION('Т', 2),
\
FST::RELATION('У', 1), FST::RELATION('У', 2),
\
FST::RELATION('Ф', 1), FST::RELATION('Ф', 2),
\
FST::RELATION('Х', 1), FST::RELATION('Х', 2),
\
FST::RELATION('Ц', 1), FST::RELATION('Ц', 2),
\
FST::RELATION('Ч', 1), FST::RELATION('Ч', 2),
\
FST::RELATION('Ш', 1), FST::RELATION('Ш', 2),
\
FST::RELATION('Щ', 1), FST::RELATION('Щ', 2),
\
FST::RELATION('Ъ', 1), FST::RELATION('Ъ', 2),
\
FST::RELATION('Ы', 1), FST::RELATION('Ы', 2),
\
FST::RELATION('Ь', 1), FST::RELATION('Ь', 2),
\
FST::RELATION('Э', 1), FST::RELATION('Э', 2),
\
FST::RELATION('Ю', 1), FST::RELATION('Ю', 2),
\
FST::RELATION('Я', 1), FST::RELATION('Я', 2),
\
FST::RELATION('а', 1), FST::RELATION('а', 2),
\
FST::RELATION('б', 1), FST::RELATION('б', 2),
\
FST::RELATION('в', 1), FST::RELATION('в', 2),
\

```

```

\      FST::RELATION('Г', 1), FST::RELATION('Г', 2),
\
\      FST::RELATION('Д', 1), FST::RELATION('Д', 2),
\
\      FST::RELATION('Е', 1), FST::RELATION('Е', 2),
\
\      FST::RELATION('Ё', 1), FST::RELATION('Ё', 2),
\
\      FST::RELATION('Ж', 1), FST::RELATION('Ж', 2),
\
\      FST::RELATION('З', 1), FST::RELATION('З', 2),
\
\      FST::RELATION('И', 1), FST::RELATION('И', 2),
\
\      FST::RELATION('Й', 1), FST::RELATION('Й', 2),
\
\      FST::RELATION('К', 1), FST::RELATION('К', 2),
\
\      FST::RELATION('Л', 1), FST::RELATION('Л', 2),
\
\      FST::RELATION('М', 1), FST::RELATION('М', 2),
\
\      FST::RELATION('Н', 1), FST::RELATION('Н', 2),
\
\      FST::RELATION('О', 1), FST::RELATION('О', 2),
\
\      FST::RELATION('П', 1), FST::RELATION('П', 2),
\
\      FST::RELATION('Р', 1), FST::RELATION('Р', 2),
\
\      FST::RELATION('С', 1), FST::RELATION('С', 2),
\
\      FST::RELATION('Т', 1), FST::RELATION('Т', 2),
\
\      FST::RELATION('У', 1), FST::RELATION('У', 2),
\
\      FST::RELATION('Ф', 1), FST::RELATION('Ф', 2),
\
\      FST::RELATION('Х', 1), FST::RELATION('Х', 2),
\
\      FST::RELATION('Ц', 1), FST::RELATION('Ц', 2),
\
\      FST::RELATION('Ч', 1), FST::RELATION('Ч', 2),
\
\      FST::RELATION('Ш', 1), FST::RELATION('Ш', 2),
\
\      FST::RELATION('Щ', 1), FST::RELATION('Щ', 2),
\
\      FST::RELATION('Ъ', 1), FST::RELATION('Ъ', 2),
\
\      FST::RELATION('Ы', 1), FST::RELATION('Ы', 2),
\
\      FST::RELATION('Ь', 1), FST::RELATION('Ь', 2),
\

```



```

\      FST::RELATION('э', 1), FST::RELATION('э', 2),
\
\      FST::RELATION('ю', 1), FST::RELATION('ю', 2),
\
\      FST::RELATION('я', 1), FST::RELATION('я', 2),
\
\      FST::RELATION('0', 1), FST::RELATION('0', 2),
\
\      FST::RELATION('1', 1), FST::RELATION('1', 2),
\
\      FST::RELATION('2', 1), FST::RELATION('2', 2),
\
\      FST::RELATION('3', 1), FST::RELATION('3', 2),
\
\      FST::RELATION('4', 1), FST::RELATION('4', 2),
\
\      FST::RELATION('5', 1), FST::RELATION('5', 2),
\
\      FST::RELATION('6', 1), FST::RELATION('6', 2),
\
\      FST::RELATION('7', 1), FST::RELATION('7', 2),
\
\      FST::RELATION('8', 1), FST::RELATION('8', 2),
\
\      FST::RELATION('9', 1), FST::RELATION('9', 2),
\
\      FST::RELATION('.', 1), FST::RELATION('.', 2),
\
\      FST::RELATION(',', 1), FST::RELATION(',', 2),
\
\      FST::RELATION('!', 1), FST::RELATION('!', 2),
\
\      FST::RELATION('?', 1), FST::RELATION('?', 2),
\
\      FST::RELATION(':', 1), FST::RELATION(':', 2),
\
\      FST::RELATION('; ', 1), FST::RELATION('; ', 2),
\
\      FST::RELATION('-', 1), FST::RELATION('-', 2),
\
\      FST::RELATION('(', 1), FST::RELATION('(', 2),
\
\      FST::RELATION('}', 1), FST::RELATION('}', 2),
\
\      FST::NODE()
\
)

```

Листинг 1 – Цепочки разбора

Приложение В

```

#include "stdafx.h"

LT::Entry::Entry()
{
    (*this).lexema = NULL;
    (*this).sn = 0;
    (*this).idxTI = TI_NULLIDX;
}

LT::Entry::Entry(char lexema, int sn, int idxTI)
{
    (*this).lexema = lexema;
    (*this).sn = sn;
    (*this).idxTI = idxTI;
}

LT::LexTable LT::Create(int size)
{
    if (size > LT_MAXSIZE)
    {
        throw ERROR_THROW(120);
    }

    LexTable lextable;
    lextable.maxsize = size;
    lextable.size = 0;
    lextable.table = new Entry[size];

    return lextable;
}

void LT::Add(LexTable& lextable, Entry entry)
{
    if (lextable.size >= LT_MAXSIZE)
    {
        throw ERROR_THROW(121);
    }

    lextable.table[lextable.size++] = entry;
}

LT::Entry LT::GetEntry(LexTable& lextable, int n)
{
    if (n < 0 || n > LT_MAXSIZE)
    {
        throw ERROR_THROW(122);
    }

    return lextable.table[n];
}

void LT::WriteTable(LexTable& lextable)

```

```

{
    std::ofstream LTfile(LT_FILENAME);

    if (!LTfile.is_open())
    {
        throw ERROR_THROW(114);
    }

    int currentLine = 1;
    LTfile << currentLine << '\t';
    for (int i = 0; i < lextable.size; i++)
    {
        LT::Entry temp = LT::GetEntry(lextable, i);

        if (currentLine != temp.sn)
        {
            currentLine = temp.sn;
            LTfile << '\n' << currentLine << '\t';
        }

        LTfile << temp.lexema;
    }

    LTfile.close();
}

void LT::Delete(LexTable& lextable)
{
    delete[] lextable.table;
    lextable.table = nullptr;
}

```

Листинг 1 – Код, реализующий таблицу лексем

```

#include "stdafx.h"

IT::Entry::Entry()
{
    idxfirstLE = 0;
    id[0] = '\0';
    scope[0] = '\0';
    iddatatype = VOID;
}

IT::IdTable IT::Create(int size)
{
    if (size > TI_MAXSIZE)
    {
        throw ERROR_THROW(123);
    }

    IdTable idtable;

```

```

        idtable.maxsize = size;
        idtable.size = 0;
        idtable.table = new Entry[size];

        return idtable;
    }

void IT::Add(IdTable& idtable, Entry entry)
{
    if (idtable.size >= TI_MAXSIZE)
    {
        throw ERROR_THROW(124);
    }

    idtable.table[idtable.size++] = entry;
}

IT::Entry IT::GetEntry(IdTable& idtable, int n)
{
    if (n < 0 || n > TI_MAXSIZE)
    {
        throw ERROR_THROW(125);
    }

    return idtable.table[n];
}

int IT::IsId(IdTable& idtable, char id[ID_MAXSIZE])
{
    for (int i = 0; i < idtable.size; i++)
    {
        if (strcmp(idtable.table[i].id, id) == 0)
        {
            return i;
        }
    }

    return TI_NULLIDX;
}

int IT::Search(IdTable& idtable, Entry entry)
{
    for (int i = 0; i < idtable.size; i++)
    {
        if (strcmp(entry.id, idtable.table[i].id) == 0 &&
            strcmp(entry.scope, idtable.table[i].scope) == 0)
        {
            return i;
        }
    }

    return TI_NULLIDX;
}

```

```

void IT::WriteTable(IdTable& idtable)
{
    std::ofstream ITfile(TI_FILENAME);

    if (!ITfile.is_open())
    {
        throw ERROR_THROW(115);
    }

    ITfile << std::left << "Id      " << '|'
        << std::setw(16) << std::left << "Identifier" << '|'
        << std::setw(10) << std::left << "Data type" << '|'
        << std::setw(17) << std::left << "Identifier type" << '|'
        << std::setw(13) << std::left << "Index      " << '|'
        << std::setw(16) << std::left << "Scope" << '|'
        << std::setw(30) << std::left << "Value" << "\n";
    ITfile << std::setfill('-') << std::setw(114) << '-' <<
std::setfill(' ') << "\n";

    for (int i = 0; i < idtable.size; i++)
    {
        IT::Entry temp = IT::GetEntry(idtable, i);

        ITfile << std::setw(5) << std::left << i << " |";
        ITfile << std::setw(16) << std::left << temp.id << '|';
        switch (temp.iddatatype)
        {
            case IT::VOID:
                ITfile << std::setw(10) << std::left << "void" <<
'|';
                break;
            case IT::INT:
                ITfile << std::setw(10) << std::left << "int" << '|';
                break;
            case IT::FLOAT:
                ITfile << std::setw(10) << std::left << "float" <<
'|';
                break;
            case IT::STR:
                ITfile << std::setw(10) << std::left << "str" << '|';
                break;
            case IT::BOOL:
                ITfile << std::setw(10) << std::left << "bool" <<
'|';
                break;
        }
        switch (temp.idtype)
        {
            case IT::V:
                ITfile << std::setw(17) << std::left << "var" << '|';
                break;
            case IT::F:
                ITfile << std::setw(17) << std::left << "func" <<
'|';

```

```

        break;
    case IT::P:
        ITfile << std::setw(17) << std::left << "param" <<
'|';
        break;
    case IT::L:
        ITfile << std::setw(17) << std::left << "literal" <<
'|';
        break;
    }
    ITfile << std::setw(13) << std::left << temp.idxfirstLE <<
'|';
    ITfile << std::setw(16) << std::left << temp.scope << '|';
    switch (temp.idtype)
    {
    case IT::V:
        switch (temp.iddatatype)
        {
        case IT::INT:
            ITfile << std::setw(30) << std::left <<
temp.value.vint;
            break;
        case IT::FLOAT:
            ITfile << std::setw(30) << std::left <<
temp.value.vfloat;
            break;
        case IT::STR:
            ITfile << std::left << "\"" <<
temp.value.vstr.str << "\"(" << temp.value.vstr.len << ")";
            break;
        case IT::BOOL:
            ITfile << std::setw(30) << std::left <<
std::boolalpha << temp.value.vbool;
            break;
        }
        break;
    case IT::F:
        switch (temp.iddatatype)
        {
        case IT::INT:
            ITfile << std::setw(30) << std::left <<
temp.value.vint;
            break;
        case IT::FLOAT:
            ITfile << std::setw(30) << std::left <<
std::fixed << std::setprecision(6) << temp.value.vfloat;
            break;
        case IT::STR:
            ITfile << std::left << "\"" <<
temp.value.vstr.str << "\"(" << temp.value.vstr.len << ")";
            break;
        case IT::BOOL:
            ITfile << std::setw(30) << std::left <<
std::boolalpha << temp.value.vbool;

```

```

        break;
    }
    break;
case IT::P:
    ITfile << std::setw(30) << std::left << "null";
    break;
case IT::L:
    switch (temp.iddatatype)
    {
        case IT::INT:
            ITfile << std::setw(30) << std::left <<
temp.value.vint;
            break;
        case IT::FLOAT:
            ITfile << std::setw(30) << std::left <<
std::fixed << std::setprecision(6) << temp.value.vfloat;
            break;
        case IT::STR:
            ITfile << std::left << "\"" <<
temp.value.vstr.str << "\"(" << temp.value.vstr.len << ")";
            break;
        case IT::BOOL:
            ITfile << std::setw(30) << std::left <<
std::boolalpha << temp.value.vbool;
            break;
    }
    break;
    }
    ITfile << "\n";
}

void IT::Delete(IdTable& idtable)
{
    delete[] idtable.table;
    idtable.table = nullptr;
}

```

Листинг 2 – Код, реализующий таблицу идентификаторов

Приложение Г

```

#include "stdafx.h"

bool trueFlag = false;
bool falseFlag = false;
bool boolFlag = false;
bool intFlag = false;
bool floatFlag = false;
bool stringFlag = false;
bool declareFunctionFlag = false;
bool declareVariableFlag = false;
bool declareIfFlag = false;
bool declareWhileFlag = false;
bool declareElseFlag = false;

int scopeCounter = 0;

LA::LEX LA::LA(Parm::PARM, In::IN in)
{
    LEX lexresult;
    LT::LexTable lextable = LT::Create(LT_MAXSIZE);
    IT::IdTable idtable = IT::Create(TI_MAXSIZE);
    LT::Entry currentEntryLT = LT::Entry();
    IT::Entry currentEntryIT = IT::Entry();
    char* buffer = new char[LA_MAXSIZE];
    char lexema = NULL;
    int indexIT;
    int bufferIndex = 0;
    int numberOfLiterals = 0;
    int currentLine = 1;
    int column = 0;

    vector<string> scopeStack;
    string global = "global\0";
    scopeStack.push_back(global);

    std::vector<char> separators = { ';', ',', '!', '[', ']', '(',
    ')', '{', '}', '+', '-', '*', '/', '%', '>', '<', '=', '!', '|'};

    for (int i = 0; i < in.size; i++)
    {
        if (find(separators.begin(), separators.end(), in.text[i])
== separators.end())
        {
            if (in.text[i] == '\\' && lexema == NULL)
            {
                buffer[bufferIndex++ ] = in.text[i++];
                column++;
                while (in.text[i] != '\\' && i < in.size)
                {
                    buffer[bufferIndex++] = in.text[i++];
                    column++;
                }
            }
        }
    }
}

```



```

        buffer[bufferIndex++] = in.text[i];
        continue;
    }
    buffer[bufferIndex++] = in.text[i];

    if (bufferIndex > LA_MAXSIZE)
    {
        throw ERROR_THROW(126);
    }
}
else
{
    buffer[bufferIndex] = '\0';
    lexema = FST(buffer);

    if (lexema == LEX_MAIN)
    {
        strncpy(currentEntryIT.id, buffer, ID_MAXSIZE);
        currentEntryIT.id[strlen(buffer)] = '\0';
        currentEntryIT.idtype = IT::F;
        currentEntryIT.iddatatype = IT::VOID;
        currentEntryIT.idxfirstLE = currentLine;

        if (!scopeStack.empty())
        {
            strncpy(currentEntryIT.scope,
scopeStack.back().c_str(), strlen(scopeStack.back().c_str()));

            currentEntryIT.scope[strlen(scopeStack.back().c_str())] = '\0';
        }
        else
        {
            currentEntryIT.scope[0] = NULL;
        }

        indexIT = IT::Search(idtable, currentEntryIT);
        if (indexIT >= 0)
        {
            throw ERROR_THROW_IN(321, currentLine,
column);
        }
        else
        {
            currentEntryLT.idxTI = idtable.size;
            IT::Add(idtable, currentEntryIT);
            currentEntryIT = IT::Entry();
        }
    }

    if (lexema == LEX_LITERAL)
    {
        currentEntryIT.idtype = IT::L;
    }
}

```

```

        if (in.text[i - strlen(buffer)] == '\\' &&
in.text[i - 1] == '\\')
        {
            currentEntryIT.iddatatype = IT::STR;
            int bufferLength = strlen(buffer) - 2;
            int length = bufferLength > TI_STR_MAXSIZE
? TI_STR_MAXSIZE : bufferLength;
            strncpy(currentEntryIT.value.vstr.str,
buffer + 1, length);
            currentEntryIT.value.vstr.str[length] =
'\0';
            currentEntryIT.value.vstr.len = length;
        }

        if ((isdigit((in.text[i - strlen(buffer)])) ||
(in.text[i] >= 'A' && in.text[i] <= 'F'))
        {
            int intLiteral = 0;

            if (strcmp(buffer, "0") == 0)
            {
                intLiteral = 0;
            }
            else {
                bool isNegative = (in.text[i -
strlen(buffer) - 1] == LA_MINUS &&
                (in.text[i - strlen(buffer) - 2]
== '(' ||
                in.text[i - strlen(buffer) -
2] == '+' ||
                in.text[i - strlen(buffer) -
2] == '-' ||
                in.text[i - strlen(buffer) -
2] == '*' ||
                in.text[i - strlen(buffer) -
2] == '/' ||
                in.text[i - strlen(buffer) -
2] == '=' ||
                in.text[i - strlen(buffer) -
2] == '{') );

                if (isNegative)
                {
                    string bufferWithMinus = "-" +
string(buffer);

                    if (bufferWithMinus[2] == 'b')
                    {
                        string binaryString =
bufferWithMinus.substr(3);
                        intLiteral =
stoi(binaryString, nullptr, 2);
                    }

```

```

        else if (bufferWithMinus[2] ==
'x')
        {
            string binaryString =
bufferWithMinus.substr(3);
            intLiteral =
stoi(binaryString, nullptr, 16);
        }
        else
        {
            intLiteral = stoi(buffer);
        }
    }
    else
    {
        if (buffer[1] == 'b')
        {
            string binaryString =
string(buffer).substr(2);
            intLiteral =
stoi(binaryString, nullptr, 2);
        }
        else if (buffer[1] == 'x')
        {
            string binaryString =
string(buffer).substr(2);
            intLiteral =
stoi(binaryString, nullptr, 16);
        }
        else
        {
            intLiteral = stoi(buffer);
        }
    }

    if (isNegative)
    {
        intLiteral = -intLiteral;
        lextable.size--;
    }

    if (intLiteral >= TI_INT_MINSIZE &&
intLiteral <= TI_INT_MAXSIZE)
    {
        currentEntryIT.iddatatype = IT::INT;
        currentEntryIT.value.vint =
intLiteral;
    }
    else
    {
        currentEntryIT.iddatatype = IT::INT;
        currentEntryIT.value.vint = 0;
    }
}

```

```

        }
    }

    if (trueFlag)
    {
        currentEntryIT.iddatatype = IT::BOOL;
        currentEntryIT.value.vbool = true;
        trueFlag = false;
    }

    if (falseFlag)
    {
        currentEntryIT.iddatatype = IT::BOOL;
        currentEntryIT.value.vbool = false;
        falseFlag = false;
    }

    indexIT = IT::Search(idtable, currentEntryIT);

    if (indexIT >= 0)
    {
        currentEntryLT.idxTI = indexIT;
    }
    else
    {
        sprintf(currentEntryIT.id, "L%d",
numberOfLiterals);

        if (!scopeStack.empty())
        {
            strncpy(currentEntryIT.scope,
scopeStack.back().c_str(), ID_MAXSIZE);

            currentEntryIT.scope[strlen(scopeStack.back().c_str())] = '\\0';
        }
        else
        {
            currentEntryIT.scope[0] = NULL;
        }

        currentEntryLT.idxTI = idtable.size;
        currentEntryIT.idxfirstLE = currentLine;
        IT::Add(idtable, currentEntryIT);
        currentEntryIT = IT::Entry();
        numberOfLiterals++;
    }
}

if (lexema == LEX_ID)
{
    bool addedToITFlag = false;
    currentEntryLT.lexema = lexema;
    currentEntryIT.idtype = IT::V;
    strncpy(currentEntryIT.id, buffer, ID_MAXSIZE);

```

```

        currentEntryIT.id[strlen(buffer)] = '\\0';
        if (!scopeStack.empty())
        {
            strncpy(currentEntryIT.scope,
scopeStack.back().c_str(), strlen(scopeStack.back().c_str()));

            currentEntryIT.scope[strlen(scopeStack.back().c_str())] = '\\0';
        }
        else
        {
            currentEntryIT.scope[0] = NULL;
        }

        if (strcmp(buffer, DATE_FUNC) == 0 ||
strcmp(buffer, ISEVEN_FUNC) == 0 || strcmp(buffer, IEMPTY_FUNC) ==
0
            || strcmp(buffer, RAND_FUNC) == 0 ||
strcmp(buffer, LENGTH_FUNC) == 0)
        {
            currentEntryIT.idtype = IT::F;

            if (strcmp(buffer, DATE_FUNC) == 0)
            {
                currentEntryIT.iddatatype = IT::STR;
                currentEntryIT.value.vstr.str[0] =
TI_STR_DEFAULT;

                currentEntryIT.value.vstr.len =
strlen(currentEntryIT.value.vstr.str);
            }

            if (strcmp(buffer, ISEVEN_FUNC) == 0 ||
strcmp(buffer, IEMPTY_FUNC) == 0)
            {
                currentEntryIT.iddatatype = IT::BOOL;
                currentEntryIT.value.vbool =
TI_BOOL_DEFAULT;
            }

            if (strcmp(buffer, RAND_FUNC) == 0 ||
strcmp(buffer, LENGTH_FUNC) == 0)
            {
                currentEntryIT.iddatatype = IT::INT;
                currentEntryIT.value.vint =
TI_INT_DEFAULT;
            }

            currentEntryLT.idxTI = idtable.size;
            currentEntryIT.idxfirstLE = currentLine;
            IT::Add(idtable, currentEntryIT);
            currentEntryIT = IT::Entry();
            addedToITFlag = true;
        }
    }

```

```

        if (lextable.table[lextable.size - 2].lexema ==
LEX_VAR && declareVariableFlag)
        {
            if (lextable.table[lextable.size -
1].lexema == LEX_INTEGER && intFlag)
            {
                currentEntryIT.iddatatype = IT::INT;
                currentEntryIT.value.vint =
TI_INT_DEFAULT;

                intFlag = false;
            }
            if (lextable.table[lextable.size -
1].lexema == LEX_STRING && stringFlag)
            {
                currentEntryIT.iddatatype = IT::STR;
                currentEntryIT.value.vstr.str[0] =
TI_STR_DEFAULT;

                currentEntryIT.value.vstr.len =
strlen(currentEntryIT.value.vstr.str);
                stringFlag = false;
            }
            if (lextable.table[lextable.size -
1].lexema == LEX_BOOL && boolFlag)
            {
                currentEntryIT.iddatatype = IT::BOOL;
                currentEntryIT.value.vbool =
TI_BOOL_DEFAULT;

                boolFlag = false;
            }

            indexIT = Search(idtable, currentEntryIT);
            if (indexIT >= 0)
            {
                throw ERROR_THROW_IN(303, currentLine,
column);
            }
            else
            {
                currentEntryLT.idxTI = idtable.size;
                currentEntryIT.idxfirstLE =
currentLine;

                IT::Add(idtable, currentEntryIT);
                currentEntryIT = IT::Entry();
                addedToITFlag = true;
            }

            declareVariableFlag = false;
        }

        if (lextable.table[lextable.size - 1].lexema ==
LEX_FUNCTION && declareFunctionFlag)
        {
            currentEntryIT.idtype = IT::F;

```

```

                if (lextable.table[lextable.size -
2].lexema == LEX_INTEGER && intFlag)
                {
                    currentEntryIT.iddatatype = IT::INT;
                    currentEntryIT.value.vint =
TI_INT_DEFAULT;

                    intFlag = false;
                }
                if (lextable.table[lextable.size -
2].lexema == LEX_STRING && stringFlag)
                {
                    currentEntryIT.iddatatype = IT::STR;
                    currentEntryIT.value.vstr.str[0] =
TI_STR_DEFAULT;

                    currentEntryIT.value.vstr.len =
strlen(currentEntryIT.value.vstr.str);
                    stringFlag = false;
                }
                if (lextable.table[lextable.size -
2].lexema == LEX_BOOL && boolFlag)
                {
                    currentEntryIT.iddatatype = IT::BOOL;
                    currentEntryIT.value.vbool =
TI_BOOL_DEFAULT;

                    boolFlag = false;
                }

                indexIT = Search(idtable, currentEntryIT);
                if (indexIT >= 0)
                {
                    throw ERROR_THROW_IN(301, currentLine,
column);
                }
                else
                {
                    currentEntryLT.idxTI = idtable.size;
                    currentEntryIT.idxfirstLE =
currentLine;

                    IT::Add(idtable, currentEntryIT);
                    currentEntryIT = IT::Entry();
                    addedToITFlag = true;
                }
            }

            if ((lextable.table[lextable.size - 2].lexema ==
LA_LEFTTHESIS &&
                lextable.table[lextable.size - 3].lexema ==
LEX_ID &&
                lextable.table[lextable.size - 3].idxTI ==
idtable.size - 1 &&
                idtable.table[idtable.size - 1].idtype ==
IT::F) ||
                lextable.table[lextable.size - 2].lexema ==
LA_COMMA && idtable.table[idtable.size - 1].idtype == IT::P)

```

```

        {
            currentEntryIT.idtype = IT::P;

            if (lextable.table[lextable.size -
1].lexema == LEX_INTEGER && intFlag)
            {
                currentEntryIT.iddatatype = IT::INT;
                currentEntryIT.value.vint =
TI_INT_DEFAULT;

                intFlag = false;
            }
            if (lextable.table[lextable.size -
1].lexema == LEX_BOOL && boolFlag)
            {
                currentEntryIT.iddatatype = IT::BOOL;
                currentEntryIT.value.vbool =
TI_BOOL_DEFAULT;

                boolFlag = false;
            }
            if (lextable.table[lextable.size -
1].lexema == LEX_STRING && stringFlag)
            {
                currentEntryIT.iddatatype = IT::STR;
                currentEntryIT.value.vstr.str[0] =
TI_STR_DEFAULT;

                currentEntryIT.value.vstr.len =
strlen(currentEntryIT.value.vstr.str);
                stringFlag = false;
            }

            indexIT = Search(idtable, currentEntryIT);
            if (indexIT >= 0)
            {
                throw ERROR_THROW_IN(302, currentLine,
column);
            }
            else
            {
                currentEntryLT.idxTI = idtable.size;
                currentEntryIT.idxfirstLE =
currentLine;

                IT::Add(idtable, currentEntryIT);
                currentEntryIT = IT::Entry();
                addedToITFlag = true;
            }
        }

        if (!addedToITFlag)
        {
            int indexIT = Search(idtable,
currentEntryIT);

            if (indexIT >= 0)
            {

```



```

        currentEntryLT.idxTI = indexIT;
        currentEntryIT = IT::Entry();
    }
    else
    {
        for (int i = scopeStack.size() - 2; i
>= 0; --i)
        {
            strncpy(currentEntryIT.scope,
scopeStack[i].c_str(), strlen(scopeStack[i].c_str()));

            currentEntryIT.scope[strlen(scopeStack[i].c_str())] = '\0';
            indexIT = Search(idtable,
currentEntryIT);

            if (indexIT >= 0)
            {
                break;
            }

            currentEntryLT.idxTI = indexIT;
            currentEntryIT = IT::Entry();
        }

        if (indexIT < 0)
        {
            throw ERROR_THROW_IN(311, currentLine,
i);
        }
    }

    if (lexema == NULL && buffer[0] != '\0' &&
std::find(separators.begin(), separators.end(),
in.text[i]) == separators.end())
    {
        throw ERROR_THROW_IN(127, currentLine, column);
    }

    bufferIndex = 0;
}

if (lexema != NULL)
{
    currentEntryLT.lexema = lexema;
    currentEntryLT.sn = currentLine;
    LT::Add(lextable, currentEntryLT);
    lexema = NULL;
    currentEntryLT = LT::Entry();
}

switch (in.text[i])
{

```

```

        case LA_NEW_LINE:
            currentLine++;
            column = 0;
            currentEntryLT = LT::Entry();
            break;
        case LA_SEMICOLON:
            currentEntryLT.lexema = LEX_SEMICOLON;
            currentEntryLT.sn = currentLine;
            LT::Add(lextable, currentEntryLT);
            currentEntryLT = LT::Entry();
            break;
        case IN_CODE_SPACE:
            column++;
            break;
        case LEX_LEFTPAREN:
            currentEntryLT.lexema = LEX_LEFTPAREN;
            currentEntryLT.sn = currentLine;
            LT::Add(lextable, currentEntryLT);

            if (declareFunctionFlag)
            {
                for (int i = idtable.size - 1; i >= 0; i--)
                {
                    if (idtable.table[i].idtype == IT::F)
                    {
                        scopeStack.push_back(idtable.table[i].id);
                        break;
                    }
                }
            }
            if (declareIfFlag || declareWhileFlag)
            {
                char scope[ID_MAXSIZE];
                if (declareIfFlag)
                {
                    sprintf(scope, "if_scope%d", scopeCounter);
                }
                if (declareWhileFlag)
                {
                    sprintf(scope, "while_scope%d",
scopeCounter);
                }
                scopeStack.push_back(scope);
            }

            currentEntryLT = LT::Entry();
            break;
        case LEX_RIGHTPAREN:
            currentEntryLT.lexema = LEX_RIGHTPAREN;
            currentEntryLT.sn = currentLine;
            LT::Add(lextable, currentEntryLT);

            if (!scopeStack.empty() && declareFunctionFlag)

```

```

        {
            scopeStack.pop_back();
            declareFunctionFlag = false;
        }
        if (!scopeStack.empty() && (declareIfFlag ||
declareElseFlag || declareWhileFlag))
        {
            scopeStack.pop_back();
        }

        currentEntryLT = LT::Entry();
        break;
    case LEX_LEFTBRACE: {
        currentEntryLT.lexema = LEX_LEFTBRACE;
        currentEntryLT.sn = currentLine;
        LT::Add(lextable, currentEntryLT);

        char scope[ID_MAXSIZE] = "";

        if (!declareWhileFlag && !declareElseFlag &&
!declareIfFlag) {
            for (int i = idtable.size - 1; i >= 0; i--) {
                if (idtable.table[i].idtype == IT::F) {
                    strncpy(scope, idtable.table[i].id,
ID_MAXSIZE - 1);

                    scope[ID_MAXSIZE - 1] = '\\0';
                    scopeStack.push_back(scope);

                    break;
                }
            }
        }
        else
        {
            if (declareIfFlag) {
                sprintf(scope, "if_scope%d", scopeCounter);
                declareIfFlag = false;
            }
            else if (declareElseFlag) {
                sprintf(scope, "else_scope%d",
scopeCounter);

                declareElseFlag = false;
            }
            else if (declareWhileFlag) {
                sprintf(scope, "while_scope%d",
scopeCounter);

                declareWhileFlag = false;
            }

            if (scope[0] != '\\0') {
                scopeStack.push_back(scope);
            }
        }
        scopeCounter++;
    }

```

```

        currentEntryLT = LT::Entry();
        break;
    }
    case LEX_RIGHTBRACE: {
        currentEntryLT.lexema = LEX_RIGHTBRACE;
        currentEntryLT.sn = currentLine;
        LT::Add(lextable, currentEntryLT);

        if (!scopeStack.empty())
        {
            scopeStack.pop_back();
        }

        currentEntryLT = LT::Entry();
        break;
    }
    case LA_COMMA:
        currentEntryLT.lexema = LEX_COMMA;
        currentEntryLT.sn = currentLine;
        LT::Add(lextable, currentEntryLT);
        currentEntryLT = LT::Entry();
        break;
    case LA_PLUS:
        currentEntryLT.lexema = LEX_PLUS;
        currentEntryLT.sn = currentLine;
        LT::Add(lextable, currentEntryLT);
        currentEntryLT = LT::Entry();
        break;
    case LA_MINUS:
        currentEntryLT.lexema = LEX_MINUS;
        currentEntryLT.sn = currentLine;
        LT::Add(lextable, currentEntryLT);
        currentEntryLT = LT::Entry();
        break;
    case LA_MULTIPLY:
        currentEntryLT.lexema = LEX_MULTIPLY;
        currentEntryLT.sn = currentLine;
        LT::Add(lextable, currentEntryLT);
        currentEntryLT = LT::Entry();
        break;
    case LA_DIVISION:
        currentEntryLT.lexema = LEX_DIVISION;
        currentEntryLT.sn = currentLine;
        LT::Add(lextable, currentEntryLT);
        currentEntryLT = LT::Entry();
        break;
    case LA_PERCENT:
        currentEntryLT.lexema = LEX_PERCENT;
        currentEntryLT.sn = currentLine;
        LT::Add(lextable, currentEntryLT);
        currentEntryLT = LT::Entry();
        break;
    case LA_EQUAL:

```

```

        currentEntryLT.lexema = LEX_EQUAL_SIGN;
        if (in.text[i + 1] == LA_EQUAL) {
            currentEntryLT.lexema = LEX_EQUAL;
            i++;
        }
        currentEntryLT.sn = currentLine;
        LT::Add(lextable, currentEntryLT);
        currentEntryLT = LT::Entry();
        break;
    case LA_MORE:
        currentEntryLT.lexema = LEX_MORE;
        if (in.text[i + 1] == LA_EQUAL) {
            currentEntryLT.lexema = LEX_MORE_EQUAL;
            i++;
        }
        currentEntryLT.sn = currentLine;
        LT::Add(lextable, currentEntryLT);
        currentEntryLT = LT::Entry();
        break;
    case LA_LESS:
        currentEntryLT.lexema = LEX_LESS;
        if (in.text[i + 1] == LA_EQUAL) {
            currentEntryLT.lexema = LEX_LESS_EQUAL;
            i++;
        }
        currentEntryLT.sn = currentLine;
        LT::Add(lextable, currentEntryLT);
        currentEntryLT = LT::Entry();
        break;
    case LA_NOT:
        if (in.text[i + 1] == LA_EQUAL)
        {
            currentEntryLT.lexema = LEX_NOT_EQUAL;
            i++;
            currentEntryLT.sn = currentLine;
            LT::Add(lextable, currentEntryLT);
        }
        else
        {
            throw ERROR_THROW_IN(206, currentLine, column);
        }
        currentEntryLT = LT::Entry();
        break;
    }
}

lexresult.lextable = lextable;
lexresult.idtable = idtable;

LT::WriteTable(lextable);
IT::WriteTable(idtable);

delete[] buffer;
return lexresult;

```

```

}

char LA::FST(char* str)
{
    FST_MAIN;
    FST_INT;
    FST_STR;
    FST_BOOL;
    FST_TRUE;
    FST_FALSE;
    FST_IDENTIFIER;
    FST_FUNCTION;
    FST_VARIABLE;
    FST_RETURN;

    FST_WHILE;
    FST_IF;
    FST_ELSE;

    FST_WRITE;
    FST_WRITELINE;
    FST_DATE;
    FST_LEN;
    FST_IEMPTY;
    FST_ISEVEN;
    FST_RAND;

    FST_INT_BIN_LITERAL;
    FST_HEX_BIN_LITERAL;
    FST_INT_LITERAL;
    FST_STRING_LITERAL;

    if (FST::execute(_main)) {
        return LEX_MAIN;
    }
    if (FST::execute(_int_bin_literal)) {
        return LEX_LITERAL;
    }
    if (FST::execute(_int_hex_literal)) {
        return LEX_LITERAL;
    }
    if (FST::execute(_int_literal)) {
        return LEX_LITERAL;
    }
    if (FST::execute(_string_literal)) {
        return LEX_LITERAL;
    }
    if (FST::execute(_int)) {
        intFlag = true;
        return LEX_INTEGER;
    }
    if (FST::execute(_string)) {
        stringFlag = true;
        return LEX_STRING;
    }

```

```

}
if (FST::execute(_true)) {
    trueFlag = true;
    return LEX_LITERAL;
}
if (FST::execute(_false)) {
    falseFlag = true;
    return LEX_LITERAL;
}
if (FST::execute(_bool)) {
    boolFlag = true;
    return LEX_BOOL;
}
if (FST::execute(_variable))
{
    declareVariableFlag = true;
    return LEX_VAR;
}
if (FST::execute(_function))
{
    declareFunctionFlag = true;
    return LEX_FUNCTION;
}
if (FST::execute(_return))
{
    return LEX_RETURN;
}
if (FST::execute(_while))
{
    declareWhileFlag = true;
    return LEX_WHILE;
}
if (FST::execute(_if))
{
    declareIfFlag = true;
    return LEX_IF;
}
if (FST::execute(_else))
{
    declareElseFlag = true;
    return LEX_ELSE;
}
if (FST::execute(_write))
{
    return LEX_WRITE;
}
if (FST::execute(_writeline))
{
    return LEX_WRITELINE;
}
if (FST::execute(_identifier))
{
    return LEX_ID;
}

```

```
}
```

Листинг 1 – Код, реализующий работу лексического анализатора

Приложение Д

```
1  tfi(ti,ti)
2  {
3  vti=1;
4  w(il>)
5  {
6  i=ii*;
7  i=il-;
8  }
9  ri;
10 }
12 m
13 {
14 vti=1;
15 vti=1;
16 vti=~iii||;
18 z(il>)
19 {
20 p(i|;
21 s(l|;
22 }
23 e
24 {
25 p(i|;
26 s(l|;
27 }
29 vti=~i|;
30 vti=~i|;
31 vti=~li|;
32 vti=~li|;
33 vti=~li|;
35 vti=il-ll-*||||;
36 }
```

Листинг 1 – Таблица лексем

Id	Identifier	Data type	Identifier type	Index
Scope	Value			

0	Power	int	func	1
global	0			
1	base	int	param	1
Power	null			
2	exponent	int	param	1
Power	null			
3	result	int	var	3
Power	0			
4	L0	int	literal	3
Power	1			
5	L1	int	literal	4
while_scope1	0			

6	L2	int	literal	7
while_scope1	1			
7	main	void	func	12
global				
8	base	int	var	14
main	0			
9	L3	int	literal	14
main	2			
10	exponent	int	var	15
main	0			
11	L4	int	literal	15
main	5			
12	result	int	var	16
main	0			
13	L5	int	literal	18
if_scope3	100			
14	L6	str	literal	21
if_scope3	"100!"(12)			
15	L7	str	literal	26
else_scope4	"100!"(15)			
16	currentDate	str	var	29
main	""(0)			
17	date	str	func	29
main	""(0)			
18	randomValue	int	var	30
main	0			
19	rand	int	func	30
main	0			
20	isEvenValue	bool	var	31
main	false			
21	isEven	bool	func	31
main	false			
22	L8	int	literal	31
main	11			
23	isEmptyValue	bool	var	32
main	false			
24	isEmpty	bool	func	32
main	false			
25	L9	str	literal	32
main	"Hello!"(6)			
26	textLength	int	var	33
main	0			
27	len	int	func	33
main	0			
28	L10	str	literal	33
main	"Hello, World!"(13)			
29	value	int	var	35
main	0			
30	L11	int	literal	35
main	100			
31	L12	int	literal	35
main	2			
32	L13	int	literal	35
main	1			

Листинг 2 – Таблица идентификаторов

Приложение Е

```

#include "stdafx.h"

namespace MFST
{
    MfstState::MfstState()
    {
        lenta_position = 0;
        nrule = -1;
        nrulechain = -1;
    };

    MfstState::MfstState(short pposition, MFSTSTSTACK pst, short
pnrulechain)
    {
        lenta_position = pposition;
        st = pst;
        nrulechain = pnrulechain;
    };

    MfstState::MfstState(short pposition, MFSTSTSTACK pst, short
pnrule, short pnrulechain)
    {
        lenta_position = pposition;
        st = pst;
        nrule = pnrule;
        nrulechain = pnrulechain;
    };

    Mfst::MfstDiagnosis::MfstDiagnosis()
    {
        lenta_position = -1;
        rc_step = SURPRISE;
        nrule = -1;
        nrule_chain = -1;
    };

    Mfst::MfstDiagnosis::MfstDiagnosis(short plenta_position,
RC_STEP prc_step, short pnrule, short pnrule_chain)
    {
        lenta_position = plenta_position;
        rc_step = prc_step;
        nrule = pnrule;
        nrule_chain = pnrule_chain;
    };

    Mfst::Mfst()
    {
        lenta = 0;
        lenta_size = 0;
        lenta_position = 0;
    };
}

```

```

Mfst::Mfst(LT::LexTable& lextable, GRB::Greibach pgrebach)
{
    grebach = pgrebach;
    lex = lextable;
    lenta = new short[lenta_size = lex.size];
    for (int k = 0; k < lenta_size; k++)
    {
        lenta[k] = GRB::Rule::Chain::T(lex.table[k].lexema);
    }
    lenta_position = 0;
    st.push(grebach.stbottomT);
    st.push(grebach.startN);
    nrulechain = -1;
}

Mfst::RC_STEP Mfst::step(Log::LOG& log)
{
    RC_STEP rc = SURPRISE;
    if (lenta_position < lenta_size)
    {
        if (GRB::Rule::Chain::isN(st.top()))
        {
            GRB::Rule rule;
            if ((nrule = grebach.getRule(st.top(), rule)) >=
0)
            {
                GRB::Rule::Chain chain;
                if ((nrulechain =
rule.getNextChain(lenta[lenta_position], chain, nrulechain + 1)) >=
0)
                {
                    MFST_TRACE1(*log.stream)
                        savestate(log);
                    st.pop();
                    push_chain(chain);
                    rc = NS_OK;
                    MFST_TRACE2(*log.stream)
                }
                else
                {
                    MFST_TRACE4(*log.stream,
"TNS_NORULECHAIN/NS_NORULE")
                        savediagnosis(NS_NORULECHAIN);
                    rc = resetstate(log) ? NS_NORULECHAIN
: NS_NORULE;
                }
            }
            else
            {
                rc = NS_ERROR;
            }
        }
        else if ((st.top() == lenta[lenta_position]))
        {

```

```

        lenta_position++;
        st.pop();
        nrulechain = -1;
        rc = TS_OK;
        MFST_TRACE3(*log.stream)
    }
    else
    {
        MFST_TRACE4(*log.stream, TS_NOK /
NS_NORULECHAIN)
        rc = resetstate(log) ? TS_NOK :
NS_NORULECHAIN;
    };
}
else
{
    rc = LENTA_END;
    MFST_TRACE4(*log.stream, LENTA_END);
};

return rc;
};

bool Mfst::push_chain(GRB::Rule::Chain chain)
{
    for (int k = chain.size - 1; k >= 0; k--)
    {
        st.push(chain.nt[k]);
    }

    return true;
};

bool Mfst::savestate(Log::LOG& log)
{
    storestate.push(MfstState(lenta_position, st, nrule,
nrulechain));
    MFST_TRACE6(*log.stream, "SAVESTATE:", storestate.size());

    return true;
};

bool Mfst::resetstate(Log::LOG& log)
{
    bool rc = false;
    MfstState state;
    if (rc = (storestate.size() > 0))
    {
        state = storestate.top();
        lenta_position = state.lenta_position;
        st = state.st;
        nrule = state.nrule;
        nrulechain = state.nrulechain;
        storestate.pop();
    }
}

```

```

        MFST_TRACE5(*log.stream, "RESSTATE")
        {
            MFST_TRACE2(*log.stream)
        }
    };

    return rc;
};

bool Mfst::savediagnosis(RC_STEP prc_step)
{
    bool rc = false;
    short k = 0;

    while (k < MFST_DIAGN_NUMBER && lenta_position <=
diagnosis[k].lenta_position)
    {
        k++;
    }

    if (rc = (k < MFST_DIAGN_NUMBER))
    {
        diagnosis[k] = MfstDiagnosis(lenta_position,
prc_step, nrule, nrulechain);

        for (int i = k + 1; i < MFST_DIAGN_NUMBER; i++)
        {
            diagnosis[i].lenta_position = -1;
        }
    }

    return rc;
};

bool Mfst::start(Log::LOG& log)
{
    bool rc = false;
    RC_STEP rc_step = SURPRISE;
    char buf[MFST_DIAGN_MAXSIZE]{};
    rc_step = step(log);
    while (rc_step == NS_OK || rc_step == NS_NORULECHAIN ||
rc_step == TS_OK || rc_step == TS_NOK)
    {
        rc_step = step(log);
    }

    switch (rc_step)
    {
    case LENTA_END:
    {
        MFST_TRACE4(*log.stream, "----->LENTA_END")
            * log.stream << "-----
-----
" << std::endl;

```

```

        sprintf_s(buf, MFST_DIAGN_MAXSIZE, "Всего строк %d,
синтаксический анализ выполнен без ошибок", 0, lenta_size);
        std::cout << std::setw(4) << std::left << "Всего
строк " << lenta_size << ", синтаксический анализ выполнен успешно!"
<< std::endl;

        rc = true;
        break;
    }

    case NS_NORULE:
    {
        MFST_TRACE4(*log.stream, "----->NS_NORULE")
        std::cout << "-----"
-----"
<< std::endl;

        std::cout << getDiagnosis(0, buf) << std::endl;
        *log.stream << getDiagnosis(0, buf) << std::endl;
        std::cout << getDiagnosis(1, buf) << std::endl;
        *log.stream << getDiagnosis(1, buf) << std::endl;
        std::cout << getDiagnosis(2, buf) << std::endl;
        *log.stream << getDiagnosis(2, buf) << std::endl;
        throw ERROR_THROW(207);
        break;
    }

    case NS_NORULECHAIN:
        MFST_TRACE4(*log.stream, "----->NS_NORULECHAIN")
        break;
    case NS_ERROR:
        MFST_TRACE4(*log.stream, "----->NS_ERROR")
        break;
    case SURPRISE:
        MFST_TRACE4(*log.stream, "----->NS_SURPRISE")
        break;
    }
    return rc;
};

char* Mfst::getCSt(char* buf)
{
    short p;
    for (int k = (signed)st.size() - 1; k >= 0; --k)
    {
        p = st.c[k];
        buf[st.size() - 1 - k] =
GRB::Rule::Chain::alphabet_to_char(p);
    }
    buf[st.size()] = '\0';

    return buf;
}

char* Mfst::getCLenta(char* buf, short pos, short n)
{

```



```

        short i, k = (pos + n < lenta_size) ? pos + n :
lenta_size;
        for (i = pos; i < k; i++)
        {
            buf[i - pos] =
GRB::Rule::Chain::alphabet_to_char(lenta[i]);
        }
        buf[i - pos] = 0x00;

        return buf;
    }

    char* Mfst::getDiagnosis(short n, char* buf)
    {
        char* rc = new char[200] {};
        int errid = 0;
        int lpos = -1;
        if (n < MFST_DIAGN_NUMBER && (lpos =
diagnosis[n].lenta_position) >= 0)
        {
            errid = grebach.getRule(diagnosis[n].nrule).iderror;
            Error::ERROR err = Error::geterror(errid);
            sprintf_s(buf, MFST_DIAGN_MAXSIZE, "%d: строка
%d,%s", err.id, lex.table[lpos].sn, err.message);
            rc = buf;
        }

        return rc;
    }

    void Mfst::printrules(Log::LOG& log)
    {
        MfstState state;
        GRB::Rule rule;
        for (unsigned short i = 0; i < storestate.size(); i++)
        {
            state = storestate.c[i];
            rule = grebach.getRule(state.nrule);
            MFST_TRACE7(*log.stream)
        };
    };

    bool Mfst::savededucation()
    {
        MfstState state;
        GRB::Rule rule;
        deducation.nrules = new short[deducation.size =
storestate.size()];
        deducation.nrulechains = new short[deducation.size];

        for (unsigned short i = 0; i < storestate.size(); i++)
        {
            state = storestate.c[i];
            deducation.nrules[i] = state.nrule;

```

```
        deducation.nrulechains[i] = state.nrulechain;
    }
    return true;
}
}
```

Листинг 1 – Структура автомата с магазинной памятью

Приложение Ж

0	:	S->tfi() {}	tfi(ti, ti) {vti=1; w(i>1) {i	S\$
0	:	SAVESTATE:	1	
0	:		tfi(ti, ti) {vti=1; w(i>1) {i	tfi() {}\$
1	:		fi(ti, ti) {vti=1; w(i>1) {i=	fi() {}\$
2	:		i(ti, ti) {vti=1; w(i>1) {i=i	i() {}\$
3	:		(ti, ti) {vti=1; w(i>1) {i=i*	() {}\$
4	:		ti, ti) {vti=1; w(i>1) {i=i*i) {}\$
5	:	2		
5	:	RESSTATE		
5	:		tfi(ti, ti) {vti=1; w(i>1) {i	S\$
6	:	S->tfi() {}S	tfi(ti, ti) {vti=1; w(i>1) {i	S\$
6	:	SAVESTATE:	1	
6	:		tfi(ti, ti) {vti=1; w(i>1) {i	tfi() {}S\$
7	:		fi(ti, ti) {vti=1; w(i>1) {i=	fi() {}S\$
8	:		i(ti, ti) {vti=1; w(i>1) {i=i	i() {}S\$
9	:		(ti, ti) {vti=1; w(i>1) {i=i*	() {}S\$
10	:		ti, ti) {vti=1; w(i>1) {i=i*i) {}S\$
11	:	2		
11	:	RESSTATE		
11	:		tfi(ti, ti) {vti=1; w(i>1) {i	S\$
12	:	S->tfi() {N}	tfi(ti, ti) {vti=1; w(i>1) {i	S\$
12	:	SAVESTATE:	1	
12	:		tfi(ti, ti) {vti=1; w(i>1) {i	tfi() {N}\$
13	:		fi(ti, ti) {vti=1; w(i>1) {i=	fi() {N}\$
14	:		i(ti, ti) {vti=1; w(i>1) {i=i	i() {N}\$
15	:		(ti, ti) {vti=1; w(i>1) {i=i*	() {N}\$
16	:		ti, ti) {vti=1; w(i>1) {i=i*i) {N}\$
17	:	2		
17	:	RESSTATE		
17	:		tfi(ti, ti) {vti=1; w(i>1) {i	S\$
18	:	S->tfi() {N}S	tfi(ti, ti) {vti=1; w(i>1) {i	S\$
18	:	SAVESTATE:	1	
18	:		tfi(ti, ti) {vti=1; w(i>1) {i	tfi() {N}S\$
19	:		fi(ti, ti) {vti=1; w(i>1) {i=	fi() {N}S\$
20	:		i(ti, ti) {vti=1; w(i>1) {i=i	i() {N}S\$
21	:		(ti, ti) {vti=1; w(i>1) {i=i*	() {N}S\$
22	:		ti, ti) {vti=1; w(i>1) {i=i*i) {N}S\$
23	:	2		
23	:	RESSTATE		
23	:		tfi(ti, ti) {vti=1; w(i>1) {i	S\$
24	:	S->tfi(F) {N}	tfi(ti, ti) {vti=1; w(i>1) {i	S\$
24	:	SAVESTATE:	1	
24	:		tfi(ti, ti) {vti=1; w(i>1) {i	tfi(F) {N}\$
25	:		fi(ti, ti) {vti=1; w(i>1) {i=	fi(F) {N}\$
26	:		i(ti, ti) {vti=1; w(i>1) {i=i	i(F) {N}\$
27	:		(ti, ti) {vti=1; w(i>1) {i=i*	(F) {N}\$
28	:		ti, ti) {vti=1; w(i>1) {i=i*i	F) {N}\$
29	:	F->ti	ti, ti) {vti=1; w(i>1) {i=i*i	F) {N}\$
29	:	SAVESTATE:	2	
29	:		ti, ti) {vti=1; w(i>1) {i=i*i	ti) {N}\$
30	:		i, ti) {vti=1; w(i>1) {i=i*i;	i) {N}\$
31	:		, ti) {vti=1; w(i>1) {i=i*i; i) {N}\$

```

32 : 2
32 : RESSTATE
32 : ti,ti){vti=1;w(i>1){i=i*i F){N}$
33 : F->ti,F ti,ti){vti=1;w(i>1){i=i*i F){N}$
33 : SAVESTATE: 2
33 : ti,ti){vti=1;w(i>1){i=i*i ti,F){N}$
34 : i,ti){vti=1;w(i>1){i=i*i; i,F){N}$
35 : ,ti){vti=1;w(i>1){i=i*i;i ,F){N}$
36 : ti){vti=1;w(i>1){i=i*i;i= F){N}$
37 : F->ti ti){vti=1;w(i>1){i=i*i;i= F){N}$
37 : SAVESTATE: 3
37 : ti){vti=1;w(i>1){i=i*i;i= ti){N}$
38 : i){vti=1;w(i>1){i=i*i;i=i i){N}$
39 : )){vti=1;w(i>1){i=i*i;i=i- )){N}$
40 : {vti=1;w(i>1){i=i*i;i=i-l {N}$
41 : vti=1;w(i>1){i=i*i;i=i-l; N}$
42 : N->vti; vti=1;w(i>1){i=i*i;i=i-l; N}$
42 : SAVESTATE: 4
42 : vti=1;w(i>1){i=i*i;i=i-l; vti;}$
43 : ti=1;w(i>1){i=i*i;i=i-l; } ti;}$
44 : i=1;w(i>1){i=i*i;i=i-l; }r i;}$
45 : =1;w(i>1){i=i*i;i=i-l; }ri ;}$
46 : 2
46 : RESSTATE
46 : vti=1;w(i>1){i=i*i;i=i-l; N}$
47 : N->vti;N vti=1;w(i>1){i=i*i;i=i-l; N}$
47 : SAVESTATE: 4
47 : vti=1;w(i>1){i=i*i;i=i-l; vti;N}$
48 : ti=1;w(i>1){i=i*i;i=i-l; } ti;N}$
49 : i=1;w(i>1){i=i*i;i=i-l; }r i;N}$
50 : =1;w(i>1){i=i*i;i=i-l; }ri ;N}$
51 : 2
51 : RESSTATE
51 : vti=1;w(i>1){i=i*i;i=i-l; N}$
52 : N->vti=E; vti=1;w(i>1){i=i*i;i=i-l; N}$
52 : SAVESTATE: 4
52 : vti=1;w(i>1){i=i*i;i=i-l; vti=E;}$
53 : ti=1;w(i>1){i=i*i;i=i-l; } ti=E;}$
54 : i=1;w(i>1){i=i*i;i=i-l; }r i=E;}$
55 : =1;w(i>1){i=i*i;i=i-l; }ri =E;}$
56 : l;w(i>1){i=i*i;i=i-l; }ri; E;}$
57 : E->l l;w(i>1){i=i*i;i=i-l; }ri; E;}$
57 : SAVESTATE: 5
57 : l;w(i>1){i=i*i;i=i-l; }ri; l;}$
58 : ;w(i>1){i=i*i;i=i-l; }ri; } ;}$
59 : w(i>1){i=i*i;i=i-l; }ri; }m }$
60 : 2
60 : RESSTATE
60 : l;w(i>1){i=i*i;i=i-l; }ri; E;}$
61 : E->lM l;w(i>1){i=i*i;i=i-l; }ri; E;}$
61 : SAVESTATE: 5
61 : l;w(i>1){i=i*i;i=i-l; }ri; lM;}$
62 : ;w(i>1){i=i*i;i=i-l; }ri; } M;}$
63 : TNS_NORULECHAIN/NS NORULE

```

```

63 : RESSTATE
63 :          1;w(i>1){i=i*i;i=i-1;}ri;      E;}$
64 : TNS_NORULECHAIN/NS_NORULE
64 : RESSTATE
64 :          vti=1;w(i>1){i=i*i;i=i-1;      N}$
65 : N->vti=EM;          vti=1;w(i>1){i=i*i;i=i-1;      N}$
65 : SAVESTATE:          4
65 :          vti=1;w(i>1){i=i*i;i=i-1;      vti=EM;}$
66 :          ti=1;w(i>1){i=i*i;i=i-1;}      ti=EM;}$
67 :          i=1;w(i>1){i=i*i;i=i-1;}r      i=EM;}$
68 :          =1;w(i>1){i=i*i;i=i-1;}ri      =EM;}$
69 :          1;w(i>1){i=i*i;i=i-1;}ri;      EM;}$
70 : E->l          1;w(i>1){i=i*i;i=i-1;}ri;      EM;}$
70 : SAVESTATE:          5
70 :          1;w(i>1){i=i*i;i=i-1;}ri;      lM;}$
71 :          ;w(i>1){i=i*i;i=i-1;}ri;}      M;}$
72 : TNS_NORULECHAIN/NS_NORULE
72 : RESSTATE
72 :          1;w(i>1){i=i*i;i=i-1;}ri;      EM;}$
73 : E->lM          1;w(i>1){i=i*i;i=i-1;}ri;      EM;}$
73 : SAVESTATE:          5
73 :          1;w(i>1){i=i*i;i=i-1;}ri;      lMM;}$
74 :          ;w(i>1){i=i*i;i=i-1;}ri;}      MM;}$
75 : TNS_NORULECHAIN/NS_NORULE
75 : RESSTATE
75 :          1;w(i>1){i=i*i;i=i-1;}ri;      EM;}$
76 : TNS_NORULECHAIN/NS_NORULE
76 : RESSTATE
76 :          vti=1;w(i>1){i=i*i;i=i-1;      N}$
77 : N->vti=E;N          vti=1;w(i>1){i=i*i;i=i-1;      N}$
77 : SAVESTATE:          4
77 :          vti=1;w(i>1){i=i*i;i=i-1;      vti=E;N}$
78 :          ti=1;w(i>1){i=i*i;i=i-1;}      ti=E;N}$
79 :          i=1;w(i>1){i=i*i;i=i-1;}r      i=E;N}$
80 :          =1;w(i>1){i=i*i;i=i-1;}ri      =E;N}$
81 :          1;w(i>1){i=i*i;i=i-1;}ri;      E;N}$
82 : E->l          1;w(i>1){i=i*i;i=i-1;}ri;      E;N}$
82 : SAVESTATE:          5
82 :          1;w(i>1){i=i*i;i=i-1;}ri;      l;N}$
83 :          ;w(i>1){i=i*i;i=i-1;}ri;}      ;N}$
84 :          w(i>1){i=i*i;i=i-1;}ri;}m      N}$
85 : N->w(E){N}          w(i>1){i=i*i;i=i-1;}ri;}m      N}$
85 : SAVESTATE:          6
85 :          w(i>1){i=i*i;i=i-1;}ri;}m      w(E){N}}$
86 :          (i>1){i=i*i;i=i-1;}ri;}m{      (E){N}}$
87 :          i>1){i=i*i;i=i-1;}ri;}m{v      E){N}}$
88 : E->i          i>1){i=i*i;i=i-1;}ri;}m{v      E){N}}$
88 : SAVESTATE:          7
88 :          i>1){i=i*i;i=i-1;}ri;}m{v      i){N}}$
89 :          >1){i=i*i;i=i-1;}ri;}m{vt      )}{N}}$
90 : 2
90 : RESSTATE
90 :          i>1){i=i*i;i=i-1;}ri;}m{v      E){N}}$
91 : E->iM          i>1){i=i*i;i=i-1;}ri;}m{v      E){N}}$

```

```

91 : SAVESTATE: 7
91 : i>l) {i=i*i;i=i-l;}ri;}m{v iM) {N}}$
92 : >l) {i=i*i;i=i-l;}ri;}m{vt M) {N}}$
93 : M->>E >l) {i=i*i;i=i-l;}ri;}m{vt M) {N}}$
93 : SAVESTATE: 8
93 : >l) {i=i*i;i=i-l;}ri;}m{vt >E) {N}}$
94 : l) {i=i*i;i=i-l;}ri;}m{vti E) {N}}$
95 : E->l l) {i=i*i;i=i-l;}ri;}m{vti E) {N}}$
95 : SAVESTATE: 9
95 : l) {i=i*i;i=i-l;}ri;}m{vti l) {N}}$
96 : ) {i=i*i;i=i-l;}ri;}m{vti= ) {N}}$
97 : {i=i*i;i=i-l;}ri;}m{vti=l {N}}$
98 : i=i*i;i=i-l;}ri;}m{vti=l; N}}$
99 : N->i=E; i=i*i;i=i-l;}ri;}m{vti=l; N}}$
99 : SAVESTATE: 10
99 : i=i*i;i=i-l;}ri;}m{vti=l; i=E;}}$
100 : =i*i;i=i-l;}ri;}m{vti=l;v =E;}}$
...

```

Листинг 1 – Дерево разбора контрольного примера на языке SAY-2024

Приложение 3

```
.586P
.MODEL FLAT, STDCALL
includelib libucrt.lib
includelib kernel32.lib
includelib ../Debug/CAY-2024ASMLIB.lib
includelib ../Debug/CAY-2024LIB.lib

ExitProcess PROTO : DWORD
copy_string PROTO
extrn WriteInt : proc
extrn WriteLineInt : proc
extrn WriteStr : proc
extrn WriteLineStr : proc
extrn WriteBool : proc
extrn WriteLineBool : proc
extrn dateCAY : proc
extrn randCAY : proc
extrn isEvenCAY : proc
extrn isEmptyCAY : proc
extrn lenCAY : proc
extrn compareCAY : proc

.STACK 4096

.CONST
L0 DWORD 1
L1 DWORD 0
L2 DWORD 1
L3 DWORD 2
L4 DWORD 5
L5 DWORD 100
L6 BYTE " больше 100!", 0
L7 BYTE " не больше 100!", 0
L8 DWORD 11
L9 BYTE "Hello!", 0
L10 BYTE "Hello, World!", 0
L11 DWORD 100
L12 DWORD 2
L13 DWORD 1

.DATA
result_Power DWORD 0
base_main DWORD 0
exponent_main DWORD 0
result_main DWORD 0
currentDate_main BYTE 14 DUP(0)
randomValue_main DWORD 0
isEvenValue_main DWORD 0
isEmptyValue_main DWORD 0
textLength_main DWORD 0
value_main DWORD 0
```

```

.CODE

Power_global PROC, exponent_Power : DWORD, base_Power : DWORD
    ; Копируем значение
    mov eax, L0
    mov result_Power, eax

    ; Начало цикла while
    mov ecx, exponent_Power
start_while0:
    cmp ecx, L1
    jle end_while0
    push result_Power
    push base_Power
    ; Получаем из стека два значения
    pop ebx
    pop eax

    ; Умножение
    imul eax, ebx
    push eax

    mov result_Power, eax
    push exponent_Power
    push L2
    ; Получаем из стека два значения
    pop ebx
    pop eax

    ; Вычитание
    sub eax, ebx
    push eax

    mov exponent_Power, eax
    mov ecx, exponent_Power
    jmp start_while0

; Конец цикла while
end_while0:
    mov eax, result_Power
    ret
Power_global ENDP

main PROC
START :
    ; Копируем значение
    mov eax, L3
    mov base_main, eax

    ; Копируем значение
    mov eax, L4
    mov exponent_main, eax

    push base_main

```



```

    push exponent_main
    ; Вызов функции
    call Power_global
    mov result_main, eax
    xor eax, eax

    ; Начало блока условия
    mov eax, result_main
    cmp eax, L5
    jg main0
    jmp else_main0

; Блок if
main0:
    ; Вывод значения
    push result_main
    call WriteInt
    ; Вывод значения
    push offset L6
    call WriteLineStr
    jmp end_main0

; Блок else
else_main0:
    ; Вывод значения
    push result_main
    call WriteInt
    ; Вывод значения
    push offset L7
    call WriteLineStr

; Конец блока if / else
end_main0:
    ; Вызов функции
    call dateCAY
    push eax
    push offset currentDate_main
    call copy_string

    ; Вызов функции
    call randCAY
    mov randomValue_main, eax

    push L8
    ; Вызов функции
    call isEvenCAY
    mov isEvenValue_main, eax

    push offset L9
    ; Вызов функции
    call isEmptyCAY
    mov isEmptyValue_main, eax

    push offset L10

```

```

; Вызов функции
call lenCAY
mov textLength_main, eax

; Вывод значения
push offset currentDate_main
call WriteLineStr
; Вывод значения
push randomValue_main
call WriteLineInt
; Вывод значения
mov eax, isEvenValue_main
push eax
call WriteLineBool
; Вывод значения
mov eax, isEmptyValue_main
push eax
call WriteLineBool
; Вывод значения
push textLength_main
call WriteLineInt
push randomValue_main
push L11
; Получаем из стека два значения
pop ebx
pop eax

; Вычитание
sub eax, ebx
push eax

push L12
push L13
; Получаем из стека два значения
pop ebx
pop eax

; Вычитание
sub eax, ebx
push eax

; Получаем из стека два значения
pop ebx
pop eax

; Умножение
imul eax, ebx
push eax

mov value_main, eax
; Вывод значения
push value_main
call WriteLineInt
push 0

```

```
        call ExitProcess  
main ENDP  
  
end main
```

Листинг 1 – Результат генерации кода на основе контрольного примера на языке программирования СAY-2024