

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора SES-2020»

Выполнил студент Шумский Евгений Сергеевич
(Ф.И.О.)

Руководитель проекта пр.ст. Наркевич Аделина Сергеевна
(учен. степень, звание, должность, подпись, Ф.И.О.)

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

Консультант пр.ст. Наркевич Аделина Сергеевна
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер пр.ст. Наркевич Аделина Сергеевна
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Оглавление

Введение	5
1. Спецификация языка программирования	6
1.1 Характеристика языка программирования	6
1.2 Алфавит языка	6
1.3 Символы-сепараторы	6
1.4 Применяемые кодировки	7
1.5 Типы данных	7
1.6 Преобразование типов данных	8
1.7 Идентификаторы	8
1.8 Литералы	8
1.9 Объявление данных	8
1.10 Инициализация данных	9
1.11 Инструкции языка	9
1.12 Операции языка	9
1.13 Выражения и их вычисление	10
1.14 Программные конструкции языка	10
1.15 Области видимости идентификаторов	10
1.16 Семантические проверки	11
1.17 Распределение оперативной памяти на этапе выполнения	11
1.18 Стандартная библиотека и её состав	11
1.19 Ввод и вывод данных	12
1.20 Точка входа	12
1.21 Препроцессор	12
1.22 Соглашения о вызовах	12
1.23 Объектный код	12
1.24 Классификация сообщений транслятора	13
1.25 Контрольный пример	13
2. Структура транслятора	14
2.1 Компоненты транслятора, их назначение и принципы взаимодействия	14

2.2	Перечень входных параметров транслятора	15
2.3	Перечень протоколов, формируемых транслятором и их содержимое	15
3.	Разработка лексического анализатора	17
3.1	Структура лексического анализатора	17
3.2.	Контроль входных символов	18
3.3	Удаление избыточных символов	18
3.4	Перечень ключевых слов	18
3.5	Основные структуры данных	20
3.6	Принцип обработки ошибок	22
3.7	Структура и перечень сообщений лексического анализатора	22
3.8	Параметры лексического анализатора	22
3.9	Алгоритм лексического анализа	22
3.10	Контрольный пример	23
4.	Разработка синтаксического анализатора	24
4.1	Структура синтаксического анализатора	24
4.2	Контекстно-свободная грамматика, описывающая синтаксис языка	24
4.3	Построение конечного магазинного автомата	26
4.4	Основные структуры данных	27
4.5	Описание алгоритма синтаксического разбора	27
4.6	Структура и перечень сообщений синтаксического анализатора	27
4.7.	Параметры синтаксического анализатора и режимы его работы	27
4.8.	Принцип обработки ошибок	28
4.9.	Контрольный пример	28
5	Разработка семантического анализатора	29
5.1	Структура семантического анализатора	29
5.2	Функции семантического анализатора	29
5.3	Структура и перечень сообщений семантического анализатора	29
5.4	Принцип обработки ошибок	30
5.5	Контрольный пример	30

6. Вычисление выражений	32
6.1 Выражения, допускаемые языком	32
6.2 Польская запись и принцип её построения	32
6.3 Программная реализация обработки выражений	32
6.4 Контрольный пример	33
7. Генерация кода	34
7.1 Структура генератора кода	34
7.2 Представление типов данных в оперативной памяти	34
7.3 Статическая библиотека	35
7.4 Особенности алгоритма генерации кода	35
7.5 Входные параметры генератора кода	35
7.6 Контрольный пример	35
8. Тестирование транслятора	37
8.1 Тестирование проверки на допустимость символов	37
8.2 Тестирование лексического анализатора	37
8.3 Тестирование синтаксического анализатора	37
8.4 Тестирование семантического анализатора	39
Заключение	40
Список использованных источников	41
Приложение А	42
Приложение В	48
Приложение Г	59
Приложение Д	67

Введение

Целью курсового проекта поставлена задача разработки компилятора для моего языка программирования – SES-2020. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над числами.

Компилятор SES-2020 – это программа, задачей которого является перевод программы, написанной на языке программирования SES-2020 в программу на язык ассемблера.

Транслятор SES-2020 состоит из следующих частей:

- лексический и семантический анализаторы;
- синтаксический анализатор;
- генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

- разработка спецификации языка программирования;
- разбратка структуры транслятора;
- разработка лексического и семантического анализаторов;
- разработка синтаксического анализатора;
- преобразование выражений;
- генерация кода в язык ассемблера;
- тестирование транслятора.

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

Глава 1. Спецификация языка программирования

1.1 Характеристика языка программирования

Язык программирования SES-2020 является процедурным, универсальным, строго типизированным, компилируемым и не объектно-ориентированным языком.

1.2 Алфавит языка

Алфавит языка SES-2020 основан на таблице символов ASCII, представленная на рис.1.1.

ASCII Code Chart																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	.	()	+	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	:	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	~	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Рисунок 1.1 – Алфавит входных символов

Символы, используемые на этапе выполнения: [a...z], [A...Z], [0...9], символы пробела, табуляции и перевода строки, спецсимволы: [] () , ; + - > < !.

1.3 Символы-сепараторы

Символы-сепараторы служат с целью разделения на токены цепочек языка. Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

Сепаратор(ы)	Назначение
' '	Разделитель цепочек. Допускается везде кроме идентификаторов и ключевых слов
[...]	Блок цикла
(...)	Блок параметров функции или блок условия цикла

Сепаратор(ы)	Назначение
,	Разделитель параметров функций
+ -	Арифметические операции
> < <= >= != ==	Логические операции (операции сравнения: больше, меньше, проверка на равенство, на неравенство), используемые в условии цикла/условной конструкции.
;	Разделитель программных конструкций
{ }	Блок функции
=	Оператор присваивания

1.4 Применяемые кодировки

Для написания исходного кода на языке SES-2020 использует кодировку ASCII, содержащую английский алфавит, а также некоторые специальные символы, такие как [] () , ; = + - > < ! { }.

1.5 Типы данных

В языке SES-2020 реализованы два типа данных: целочисленный и строковый. Описание типов данных, предусмотренных в данном языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка SES-2020

Тип данных	Описание типа данных
Целочисленный тип данных <i>int</i>	Фундаментальный тип данных. Используется для работы с числовыми значениями. В памяти занимает 4 байт. Максимальное значение: 4,294,967,295. Минимальное значение: 0. Инициализация по умолчанию: значение 0.
Строковый тип данных <i>string</i>	Фундаментальный тип данных. Используется для работы с символами, каждый символ в памяти занимает 4 байта. Максимальное количество символов: 255. Инициализация по умолчанию: строка нулевой длины "".

1.6 Преобразование типов данных

Преобразование типов данных в языке SES-2020 не поддерживается, т.е. язык является строго типизированным.

1.7 Идентификаторы

В имени идентификатора допускаются только символы латинского алфавита нижнего регистра. Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Максимальная длина имени идентификатора - 10 символов. Идентификаторы, объявленные внутри функционального блока, получают суффикс, идентичный имени функции, внутри которой они объявлены. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, имя параметра функции. Имена идентификаторов-функций могут совпадать с именами команд ассемблера.

1.8 Литералы

С помощью литералов осуществляется инициализация переменных. Все литералы являются *rvalue*. Типы литералов языка SES-2020 представлены в таблице 1.3.

Таблица 1.3 – Описание литералов

Тип литерала	Описание
Целочисленные литералы в десятичном представлении	Последовательность цифр [0...9]
Строковые литералы	Набор символов, заключённых в двойные кавычки

Ограничения на строковые литералы языка SES-2020: внутри литерала не допускается использование символов кириллицы, а также одинарных и двойных кавычек. Ограничения на целочисленные литералы: не могут начинаться с нуля, если их значение не ноль;

1.9 Объявление данных

Для объявления переменной используется ключевое слово *proo*, после которого указывается тип данных и имя идентификатора. Инициализация при объявлении не допускается.

Пример объявления числового типа с инициализацией:

```
proo int value1;
```

Пример объявления переменной символьного типа с инициализацией:

```
proo string str1;
```


Для объявления функций используется ключевое слово *func*, перед которым указывается тип функции (если функция возвращает значение), а после – имя функции либо процедуры. Далее обязателен список параметров и тело функции.

1.10 Инициализация данных

При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть только идентификаторы или литералы. При объявлении без инициализации предусмотрены значения по умолчанию: значение 0 для типа *int* и строка длины 0 ("") для типа *string*.

1.11 Инструкции языка

Инструкции языка SES-2020 представлены в таблице 1.4

Таблица 1.4 – Инструкции языка SES-2020

Инструкция	Запись на языке SES-2020
Объявление переменной	<i>proo</i> <тип данных> <идентификатор>;
Возврат из функции или процедуры	Для функций, возвращающих значение: <i>return</i> <идентификатор/литерал>;
Вывод данных	<i>out</i> <идентификатор/литерал>;
Вызов функции или процедуры	<идентификатор функции> (<список параметров>); Список параметров может быть пустым.
Присваивание	<идентификатор> = <выражение>;

1.12 Операции языка

В языке SES-2020 предусмотрены следующие операции с данными. Приоритетность операции сложения равна приоритетности операции вычитания. Операции языка представлены в таблице 1.5.

Таблица 1.5 – Операции языка SES-2020

Тип оператора	Оператор
Арифметические	1. + – сложение 2. - – вычитание 3. = – присваивание
Строковые	1. = – присваивание
Логические	1. > – больше 2. < – меньше 3. <= - меньше или равно 4. >= - больше или равно 5. == - равно

1.13 Выражения и их вычисление

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

1. Выражение записывается в строку без переносов;
2. Использование двух подряд идущих операторов не допускается;
3. Допускается использовать в выражении вызов функции, вычисляющей и возвращающей целочисленное значение.

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

1.14 Программные конструкции языка

Программа на языке SES-2020 оформляется в виде функций пользователя и главной функции. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для лучшей читаемости кода.

Программные конструкции языка представлены в таблице 1.6.

Таблица 1.6 – Программные конструкции языка SES-2020

Конструкция	Запись на языке SES-2020
Главная функция(точка входа)	<i>main</i> { ... }
Цикл	Twirl(<идентификатор1> <оператор> <идентификатор2>) { ... }
Внешняя функция	<тип данных> <i>func</i> <идентификатор> (<тип> <идентификатор>, ...) {... <i>return</i> <идентификатор/литерал>; }

1.15 Области видимости идентификаторов

В языке SES-2020 все переменные являются локальными. Они обязаны находиться внутри программного блока функций (по принципу C++). Объявление глобальных переменных не предусмотрено. Каждая переменная

или параметр функции получают суффикс – название функции, внутри которой они находятся.

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

1.16 Семантические проверки

В языке программирования SES-2020 выполняются следующие семантические проверки приведённых в таблице 1.8.

Таблица 1.8 – Семантические проверки

Номер	Правило
1	Проверка соответствия типа функции и возвращаемого параметра
2	Правильность передаваемых в функцию параметров: количество, типы
3	Превышение размера строковых и числовых литералов
4	Правильность составленного условия цикла

1.17 Распределение оперативной памяти на этапе выполнения

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

1.18 Стандартная библиотека и её состав

В языке SES-2020 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержимое библиотеки и описание функций представлено в таблице 1.9.

Таблица 1.9 – Состав стандартной библиотеки

Функция	Описание
<i>int mod(int a, int b)</i>	Целочисленная функция. Возвращает остаток от деления числа a на число b.
<i>int square(int a, int b);</i>	Целочисленная функция. Возвращает число a возведенное в степень b.

Стандартная библиотека написана на языке C++, подключается к транслированному коду на этапе генерации кода.

Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Для вывода предусмотрен оператор *out*. Эти функции представлены в таблице 1.10.

Таблица 1.10 – Дополнительные функции стандартной библиотеки

Функция на языке C++	Описание
<code>void printi(int value)</code>	Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала.
<code>void prints(char* line)</code>	Функция для вывода в стандартный поток значения строкового идентификатора/литерала.

1.19 Ввод и вывод данных

В языке SES-2020 реализованы средства вывода данных с помощью оператора *out*. Допускается использование оператора *out* с литералами и идентификаторами.

Функции, управляющие выводом данных, реализованы на языке C++ и вызываются из транслированного кода, конечному пользователю недоступны. Пользовательская команда *out* в транслированном коде будут заменена вызовом нужных библиотечных функций. Библиотека, содержащая нужные процедуры, подключается на этапе генерации кода.

1.20 Точка входа

В языке SES-2020 каждая программа должна содержать главную функцию *main*, т.е. точку входа, с которой начнется последовательное выполнение программы.

1.21 Препроцессор

Препроцессор в языке программирования SES-2020 отсутствует.

1.22 Соглашения о вызовах

В языке вызов функций происходит по соглашению о вызовах *stdcall*. Особенности *stdcall*:

- все параметры функции передаются через стек;
- память высвобождает вызываемый код;
- занесение в стек параметров идёт справа налево.

1.23 Объектный код

Язык SES-2020 транслируется в язык ассемблера, а затем - в объектный код.

1.24 Классификация сообщений транслятора

В случае возникновения ошибки в коде программы на языке SES-2020 и выявления её транслятором в текущий файл протокола выводится сообщение. Их классификация сообщений приведена в таблице 1.10.

Таблица 1.10 – Классификация сообщений транслятора

Сообщение	Описание
0-99	ошибка системы
100-199	Ошибка входного кода
200-299	Ошибка на этапе лексического анализа
300-399	Ошибка на этапе синтаксического анализа
400-499	Ошибка на этапе семантического анализа

1.25 Контрольный пример

Контрольный пример представлен в главе Приложения А.

2. Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор преобразует программу, написанную на языке SES-2020 в программу на языке ассемблера. Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера.

Принцип работы представлен на рисунке 2.1.

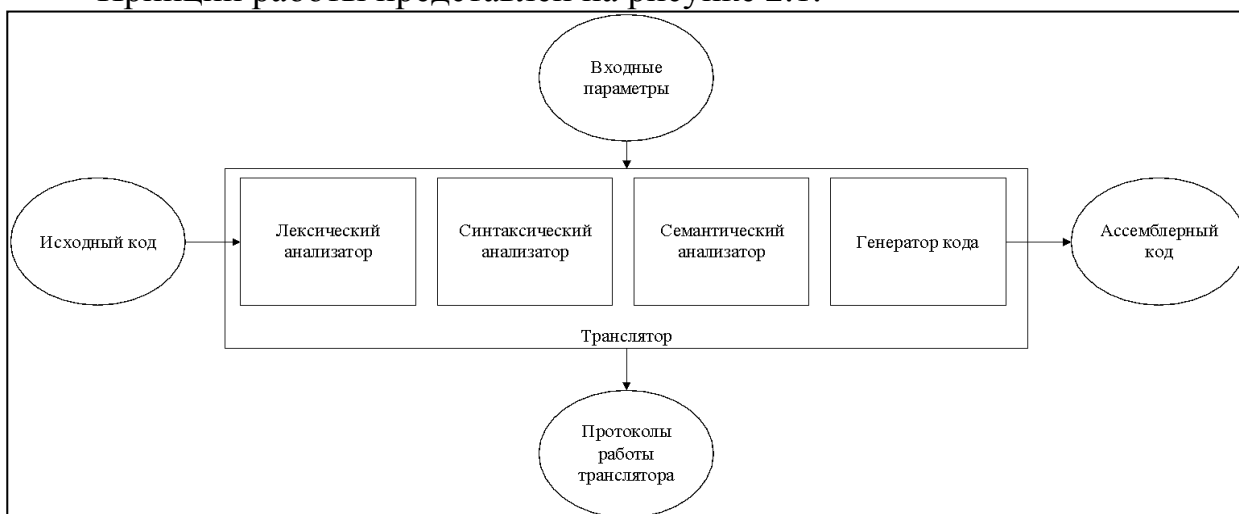


Рисунок 2.1 – Структура транслятора языка программирования SES-2020

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

Таблица лексем (ТЛ) и таблица идентификаторов (ТИ) являются входом для следующей фазы компилятора – синтаксического анализа (разбора, парсера).

Цели лексического анализатора:

- убрать все лишние пробелы;
- выполнить распознавание лексем;

- построить таблицу лексем и таблицу идентификаторов;
- при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

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

Семантический анализ в свою очередь является проверкой исходной программы SES-2020 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке SES-2020, прошедший все предыдущие этапы, в код на языке Ассемблера.

2.2 Перечень входных параметров транслятора

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка SES-2020

Входной параметр	Описание	Значение по умолчанию
-in:<имя_файла>	Входной файл с расширением .txt, в котором содержится исходный код на SES-2020. Данный параметр должен быть указан обязательно. В случае если он не будет задан, то выполнение этапа трансляции не начнётся.	Не предусмотрено
-log:<имя_файла>	Файл содержит в себе краткую информацию об исходном коде на языке SES-2020.	<имя_файла>.log
-out:<имя_файла>	Файл содержит в себе выходной код на языке ассемблера.	<имя_файла>.asm
-d	Ключ включает режим отладки. Выводит информацию о лексическом и синтаксическом анализе, а также результат польской записи	По умолчанию отключен
-lex	Ключ для вывода результата лексического анализа	По умолчанию отключен

2.3 Перечень протоколов, формируемых транслятором и их

содержимое

Таблица с перечнем протоколов, формируемых транслятором языка SES-2020 и их назначением представлена в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка SES-2020

Формируемый протокол	Описание протокола
Файл журнала, заданный параметром "-log:"	Файл содержит в себе краткую информацию об исходном коде на языке SES-2020. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора.
Файл таблицы лексем, с расширением ".lex"	Файл содержит в себе часть результата работы лексического анализатора: таблицу лексем.
Файл таблицы идентификаторов, с расширением ".id"	Файл содержит в себе часть результата работы лексического анализатора: таблицу идентификаторов.
Выходной файл, с расширением ".asm"	Результат работы программы – файл, содержащий исходный код на языке ассемблера.

3. Разработка лексического анализатора

3.1 Структура лексического анализатора

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

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

Функции лексического анализатора:

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

- распознавание идентификаторов и ключевых слов;
- распознавание констант;
- распознавание разделителей и знаков операций.

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

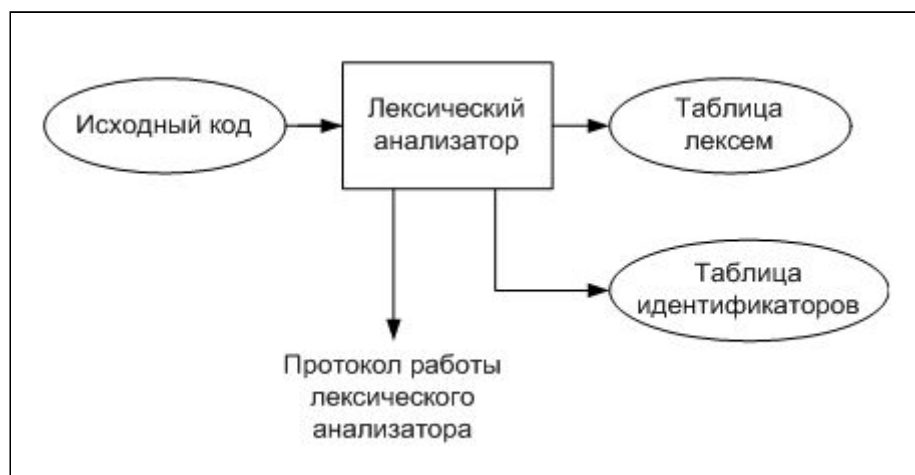


Рисунок 3.1 – Структура лексического анализатора

3.2. Контроль входных символов

Исходный код на языке программирования SES-2020 прежде чем транслироваться проверяется на допустимость символов. То есть изначально из входного файла считывается по одному символу и проверяется является ли он разрешённым.

Таблица для контроля входных символов представлена на рисунке 3.2., категории входных символов представлены в таблице 3.1.

```
#define IN_CODE_TABLE {\n    IN::I, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, \n    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \n    IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::T, IN::T, IN::T, IN::F, IN::T, IN::T, IN::T, IN::F, IN::T, \n    IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::F, IN::T, IN::T, IN::T, IN::T, IN::F, \n    IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \n    IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::F, IN::T, IN::F, IN::F, \n    IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \n    IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::F, IN::T, IN::F, IN::F, \n    \n    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \n    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \n    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \n    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \n    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \n    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \n    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \n}
```

Рисунок 3.2. – Таблица контроля входных символов

Таблица 3.1 – Соответствие символов и их значений в таблице

Значение в таблице входных символов	Символы
Разрешенный	T
Запрещенный	F

3.3 Удаление избыточных символов

Удаление избыточных символов не предусмотрено, так как после проверки на допустимость символов исходный код на языке программирования SES-2020 разбивается на токены, которые записываются в очередь.

Описание алгоритма удаления избыточных символов:

- 1) Посимвольно считываем файл с исходным кодом программы;
- 2) Встреча пробела или знака табуляции является своего рода встречей символа-сепаратора;
- 3) В отличие от других символов-сепараторов не записываем в очередь лексем эти символы, т.е. игнорируем.

3.4 Перечень ключевых слов

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного

представления исходной программы. Соответствие токенов и лексем приведено в таблице 3.2.

Таблица 3.2 – Соответствие токенов и сепараторов с лексемами

Токен	Лексема	Пояснение
main	m	Главная функция.
proo	d	Объявление переменной.
func	f	Объявление функции.
out	o	Ввод данных.
int, string	t	Названия типов данных языка.
Идентификатор	i	Идентификатор переменной, функции либо параметра функции.
Литерал	l	Литерал любого доступного типа.
return	r	Выход из функции/процедуры.
twirl	w	Указывает начало цикла
;	;	Разделение выражений.
,	,	Разделение параметров функций.
+	+	Знаки операций.
-	-	
> < <= >= !=	c	Знаки логических операторов
[[Открытие блока цикла.
]]	Заккрытие блока цикла.
}	}	Открытие блока функции.
}	}	Заккрытие блока функции.
((Передача параметров в функцию.
))	Заккрытие блока для передачи параметров, приоритет операций.
))	Заккрытие блока для передачи параметров.

Пример реализации таблицы лексем представлен в приложении Б.

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного автомата и пример графа перехода конечного автомата изображены на рисунках 3.3 и 3.4

СООТВЕТСТВЕННО.

```
namespace FST
{
    struct RELATION
    {
        char symbol;
        short nnode;
        RELATION(
            char c,
            short ns
        );
    };

    struct NODE
    {
        short n_relation;
        RELATION* relations;
        NODE();
        NODE(short n, RELATION rel, ...);
    };

    struct FST
    {
        char* string;
        short position;
        short nstates;
        NODE* node;
        short* rstates;
        FST(short ns, NODE n, ...);
        FST(char* s, FST& fst);
    };

    bool execute(FST& fst);
};
```

Рисунок 3.3 – Структура конечного автомата

```
#define A_MAIN(string) string, 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()
```

Рисунок 3.4 – Пример реализации графа конечного автомата для токена main (точки входа)

3.5 Основные структуры данных

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, лексему (lexema), полученную при разборе, номер строки в исходном коде (sn), и номер в таблице идентификаторов, если лексема является идентификатором (idxTI). Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (или параметры функций) (value). Код С++ со структурой таблицы лексем представлен на рисунке 3.3. Код С++ со структурой таблицы идентификаторов представлен

на рисунке 3.4.

```
namespace LT
{
    struct Entry
    {
        char lexema;
        int sn;
        int idxTI;
        char operation[3];

        Entry(char lexema, int sn, int idxTI, char* operation);
        Entry(char lexema, int sn, int idxTI);
        Entry();
    };

    struct LexTable
    {
        int maxsize;
        int size;
        Entry* table;
    };
}
```

Рисунок 3.3 – Структура таблицы лексем

```
struct Entry
{
    union
    {
        int vint;
        struct
        {
            int len;
            char str[STR_MAXSIZE - 1]; //сами символы
        } vstr; //строковое значение
        struct
        {
            int count; // кол-во пр. функции
            IDDATATYPE* types; //типы парм функции
        } params;
    } value; //значение идентификатора
    int idxfirstLE; //индекс в таблице лексем
    char id[SCOPED_ID_MAXSIZE]; //идентификатор
    IDDATATYPE iddatatype; //тип данных
    IDTYPE idtype; //тип идентификатора
    // дальше 2 конструктора
    Entry() = default;
    Entry(char* id, int idxLT, IDDATATYPE datatype, IDTYPE idtype)
    {
        strncpy_s(this->id, id, SCOPED_ID_MAXSIZE - 1);
        this->idxfirstLE = idxLT;
        this->iddatatype = datatype;
        this->idtype = idtype;
    };
};

struct IdTable //экземпляр таблицы идентификаторов
{
    int maxsize;
    int size;
    Entry* table;
};
```

Рисунок 3.4 – Структура таблицы идентификаторов

3.6 Принцип обработки ошибок

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. При возникновении сообщения, дальнейшая обработка прекращается. Перечень сообщений представлен на рисунке 3.5.

```
ERROR_ENTRY(200, "Таблица лексем переполнена"),
ERROR_ENTRY(201, "Нераспознанная лексема"),
ERROR_ENTRY(202, "Таблица идентификаторов переполнена"),
ERROR_ENTRY(203, "Перезапись идентификатора"),
ERROR_ENTRY(204, "Выход за границу таблицы лексем (или идентификаторов)"),
ERROR_ENTRY(205, "Не удалось создать файл с лексемами (или идентификаторами)"),
ERROR_ENTRY(206, "Слишком длинный литерал"),
ERROR_ENTRY(207, "Неверный формат строкового литерала"),
ERROR_ENTRY(208, "Слишком длинное имя переменной"),
ERROR_ENTRY(209, "Неизвестная переменная"),
ERROR_ENTRY(210, "Недопустимый идентификатор"),
ERROR_ENTRY_NODEF(211), ERROR_ENTRY_NODEF(212), ERROR_ENTRY_NODEF(213), ERROR_ENTRY_NODEF(214), ERROR_ENTRY_NODEF(215),
ERROR_ENTRY_NODEF(216), ERROR_ENTRY_NODEF(217), ERROR_ENTRY_NODEF(218), ERROR_ENTRY_NODEF(219),
ERROR_ENTRY_NODEF10(220), ERROR_ENTRY_NODEF10(230), ERROR_ENTRY_NODEF10(240), ERROR_ENTRY_NODEF10(250), ERROR_ENTRY_NODEF10(260),
ERROR_ENTRY_NODEF10(270), ERROR_ENTRY_NODEF10(280), ERROR_ENTRY_NODEF10(290),
```

Рисунок 3.5 – Сообщения лексического анализатора

3.7 Структура и перечень сообщений лексического анализатора

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

3.8 Параметры лексического анализатора

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов могут выводиться как в файл, так и в командную строку.

3.9 Алгоритм лексического анализа

- 1) Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы, удаляет лишние пробелы и добавляет сепаратор для вычисления номера строки для каждой лексемы;
- 2) для выделенной части входного потока выполняется функция распознавания лексемы;
- 3) при успешном распознавании информация о выделенной лексеме

заносятся в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;

4) при неуспешном распознавании выдается сообщение об ошибке.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «*string*» представлен на рисунке 3.2, где S0 – начальное, а S6 – конечное состояние автомата.

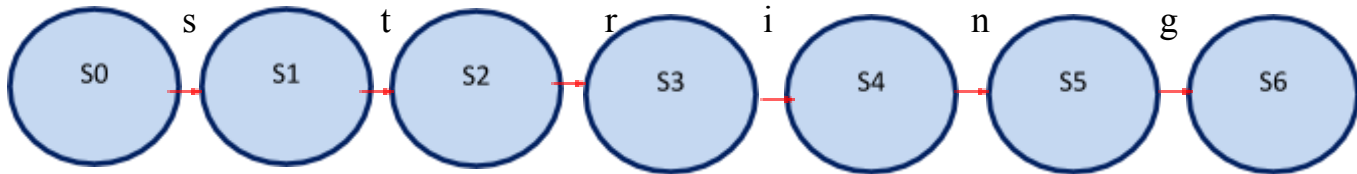


Рисунок 3.2 – Граф переходов для цепочки “*string*”

3.10 Контрольный пример

Результат работы лексического анализатора в виде таблиц лексем и идентификаторов, соответствующих контрольному примеру, представлен в приложении Б.

4. Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

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

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

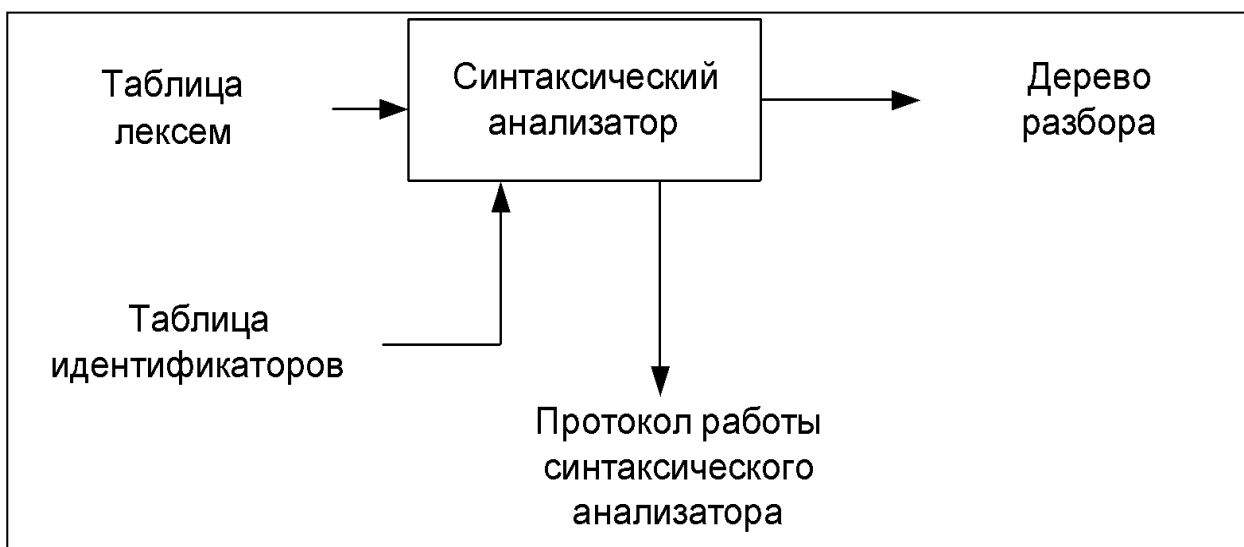


Рисунок 4.1 – Структура синтаксического анализатора.

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка SES-2020 используется контекстно-свободная грамматика $G = \langle T, N, P, S \rangle$, где

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

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

2) $S \rightarrow \lambda$, где $S \in N$ — начальный символ, при этом если такое правило существует, то нетерминал S не встречается в правой части правил. Описание нетерминальных символов содержится в таблице 4.1.

Таблица 4.1 – Таблица правил переходов нетерминальных символов

Символ	Правила	Какие правила порождает
S	$S \rightarrow \text{tfi}(F) \{NrE;\}$ S $S \rightarrow m \{NrE;\}$	Стартовые правила, описывающее общую структуру программы
F	$F \rightarrow ti$ $F \rightarrow ti, F$	Правила для параметров объявляемых функций
N	$N \rightarrow dti;$ $N \rightarrow rE;$ $N \rightarrow i=E;$ $N \rightarrow o(E);$ $N \rightarrow dti; N$ $N \rightarrow i=E; N$ $N \rightarrow o(E); N$ $N \rightarrow w(E)[N]$ $N \rightarrow w(E)[N]N$	Правила для тела функций
E	$E \rightarrow i$ $E \rightarrow l$ $E \rightarrow (E)$ $E \rightarrow i(W)$ $E \rightarrow iM$ $E \rightarrow lM$ $E \rightarrow (E)M$ $E \rightarrow i(W)M$ $E \rightarrow icE$ $E \rightarrow s(i,i)$ $E \rightarrow s(i,l)$ $E \rightarrow s(l,l)$ $E \rightarrow s(l,i)$ $E \rightarrow s(i,i)M$ $E \rightarrow s(i,l)M$ $E \rightarrow s(l,l)M$ $E \rightarrow s(l,i)M$ $E \rightarrow k(i,i)$ $E \rightarrow k(i,l)$ $E \rightarrow k(l,l)$	Правила для выражения

	$E \rightarrow k(l,i)$ $E \rightarrow k(i,i)M$ $E \rightarrow k(i,l)M$ $E \rightarrow k(l,l)M$ $E \rightarrow k(l,i)M$	
M	$M \rightarrow +E$ $M \rightarrow -E$ $M \rightarrow +E$ $M \rightarrow +EM$	Правила для выражений
W	$W \rightarrow i$ $W \rightarrow l$ $W \rightarrow i, W$ $W \rightarrow l, W$	Правила для вывозов функций

4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку $M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$. Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2 – Описание компонентов магазинного автомата

Компонент a	Определение	Описание
Q	Множество состояний автомата	Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата
V	Алфавит входных символов	Алфавит представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблица 3.1 и 4.1.
Z	Алфавит специальных магазинных символов	Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ \$)
δ	Функция переходов автомата	Функция представляет из себя множество правил грамматики, описанных в таблице 4.1.
q_0	Начальное состояние автомата	Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики
z_0	Начальное состояние магазина	Символ маркера дна стека \$

	автомата	
F	Множество конечных состояний	Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты

4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка SES-2020. Данные структуры в приложении В.

4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата следующий:

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

4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен на рисунке 4.3.

```

ERROR_ENTRY(300, "Неверная структура программы"),
ERROR_ENTRY(301, "Ошибочный сепаратор"),
ERROR_ENTRY(302, "Ошибка в выражении"),
ERROR_ENTRY(303, "Ошибка в параметрах функции"),
ERROR_ENTRY(304, "Ошибка в параметрах вызываемой функции"),
ERROR_ENTRY_NODEF(305), ERROR_ENTRY_NODEF(306), ERROR_ENTRY_NODEF(307),
ERROR_ENTRY_NODEF(308), ERROR_ENTRY_NODEF(309),
ERROR_ENTRY_NODEF10(310), ERROR_ENTRY_NODEF10(320), ERROR_ENTRY_NODEF10(330), ERROR_ENTRY_NODEF10(340), ERROR_ENTRY_NODEF10(350),
ERROR_ENTRY_NODEF10(360), ERROR_ENTRY_NODEF10(370), ERROR_ENTRY_NODEF10(380), ERROR_ENTRY_NODEF10(390),

```

Рисунок 4.3 – Сообщения синтаксического анализатора

4.7. Параметры синтаксического анализатора и режимы его работы

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

4.8. Принцип обработки ошибок

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке (если она найдена). Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

4.9. Контрольный пример

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

5 Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.

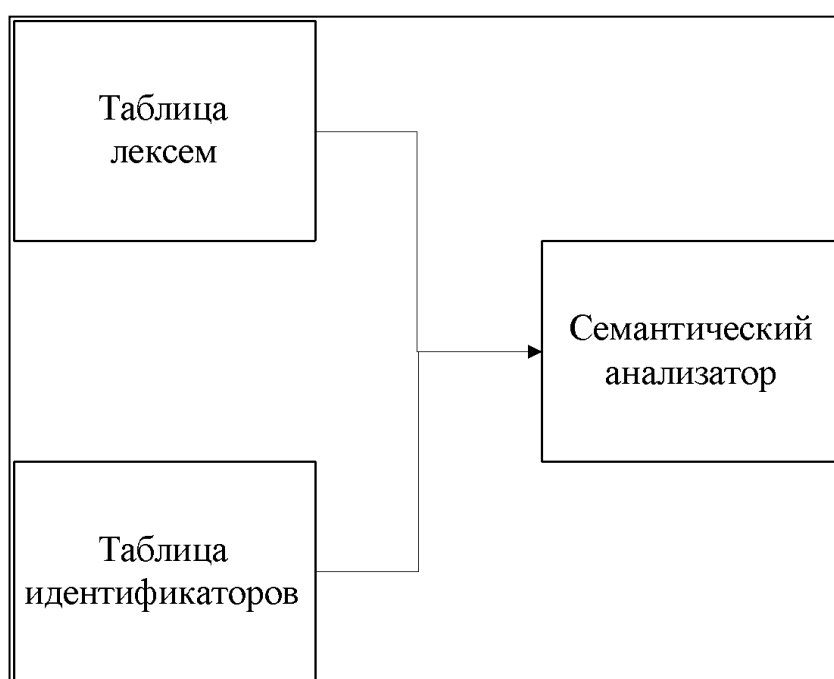


Рисунок 5.1. – Структура семантического анализатора

5.2 Функции семантического анализатора

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

5.3 Структура и перечень сообщений семантического анализатора

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

```

ERROR_ENTRY(400, "Return не может возвращать функцию"),
ERROR_ENTRY(401, "Ошибка в аргументах вызываемой функции функции"),
ERROR_ENTRY(402, "Нельзя присвоить значение данного типа"),
ERROR_ENTRY(403, "Недопустимый тип аргумента цикла"),
ERROR_ENTRY(404, "Недопустимое количество аргументов функции"),
ERROR_ENTRY_NODEF10(405), ERROR_ENTRY_NODEF10(406),
ERROR_ENTRY_NODEF(407), ERROR_ENTRY_NODEF(408),
ERROR_ENTRY_NODEF(409), ERROR_ENTRY_NODEF10(410),
ERROR_ENTRY_NODEF10(420), ERROR_ENTRY_NODEF10(430),
ERROR_ENTRY_NODEF10(440), ERROR_ENTRY_NODEF10(450),
ERROR_ENTRY_NODEF10(460), ERROR_ENTRY_NODEF10(470),
ERROR_ENTRY_NODEF10(480), ERROR_ENTRY_NODEF10(490),

```

Рисунок 5.2 – Перечень сообщений семантического анализатора

5.4 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входными параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением. Анализ останавливается после того, как будут найдены все ошибки.

5.5 Контрольный пример

Соответствие примеров некоторых ошибок в исходном коде и диагностических сообщений об ошибках приведено в таблице 5.1.

Таблица 5.1. – Примеры диагностики ошибок

Исходный код	Текст сообщения
<pre> main{ poo int a; a = 10; poo string b; b = 'start'; twirl(a<b) { } ... } </pre>	<p>Ошибка N400: Семантическая ошибка: Return не может содержать функцию Строка: 2</p>
<pre> main{ poo int b; b= 9; poo string y; y = b; ... } </pre>	<p>Ошибка N402: Нельзя присвоить значение данного типа, Строка: 5</p>

<pre>int func ok{ ... return mod(9, 3); }</pre>	<p>Ошибка N403: Недопустимый тип аргумента цикла</p>
---	--

6. Вычисление выражений

6.1 Выражения, допускаемые языком

В языке SES-2020 допускаются вычисления выражений целочисленного типа данных с поддержкой вызова функций внутри выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1 – Приоритеты операций

Операция	Значение приоритета
+	1
-	1
}	0
{	0

6.2 Польская запись и принцип её построения

Все выражения языка SES-2020 преобразовываются к обратной польской записи.

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

- 1) исходная строка: выражение;
- 2) результирующая строка: польская запись;
- 3) стек: пустой;
- 4) исходная строка просматривается слева направо;
- 5) операнды переносятся в результирующую строку;
- 6) операция записывается в стек, если стек пуст;
- 7) операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
- 8) отрывающая скобка помещается в стек;
- 9) закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.

6.3 Программная реализация обработки выражений

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

6.4 Контрольный пример

Пример преобразования выражений из контрольных примеров к обратной польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления и преобразования к ассемблерному коду. В приложении Г приведены изменённые таблицы лексем и идентификаторов, отображающие результаты преобразования выражений в польский формат.

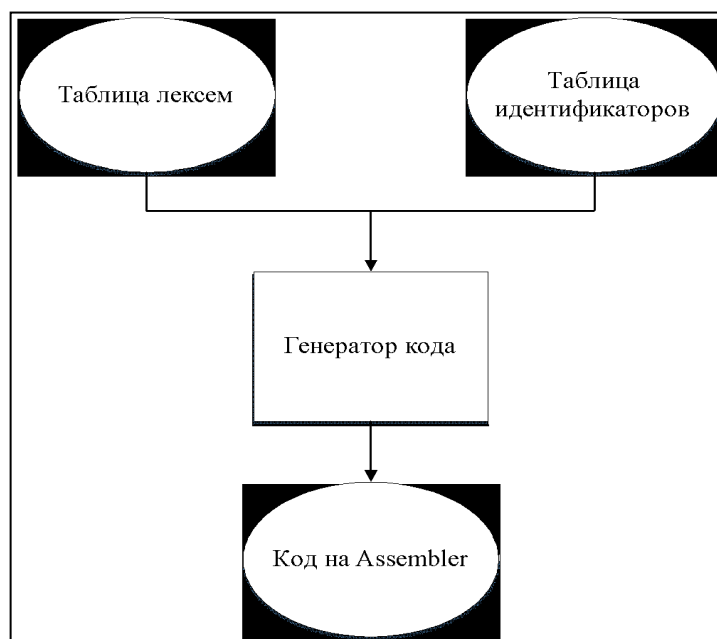
Таблица 6.2. – Преобразование выражений к ПОЛИЗ

Выражение	Обратная польская запись для выражения
$i=i+i-i+i$	$I=iiii+-+$
$i=i+l-l$	$i=ill+-$

7. Генерация кода

7.1 Структура генератора кода

В языке SES-2020 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится выходной файл на языке ассемблера, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода SES-2020 представлена на рисунке 7.1.



Р и с у н о к 7.1 – С т р у к т у р а г е н е р а т о р а к о д а

7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке SES-2020 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка SES-2020 и языка ассемблера

Тип идентификатора на языке SES-2020	Тип идентификатора на языке ассемблера	Пояснение
int	sdword	Хранит целочисленный тип данных.
string	dword	Хранит указатель на начало строки. Строка должна завешаться нулевым

		СИМВОЛОМ.
--	--	-----------

7.3 Статическая библиотека

В языке SES-2020 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++.

Объявление функций статической библиотеки генерируется автоматически в коде ассемблера. Объявление функций статической библиотеки генерируется автоматически.

Таблица 7.3 – Функции статической библиотеки

Функция	Назначение
void prints(char* str)	Вывод на консоль строки str
void printi(int num)	Вывод на консоль целочисленной переменной num
int octat(int a, int b)	Вычисление остатка от деления числа a на число b
char* square(int a, int b)	Возведение числа a в степень b

7.4 Особенности алгоритма генерации кода

В языке SES-2020 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке

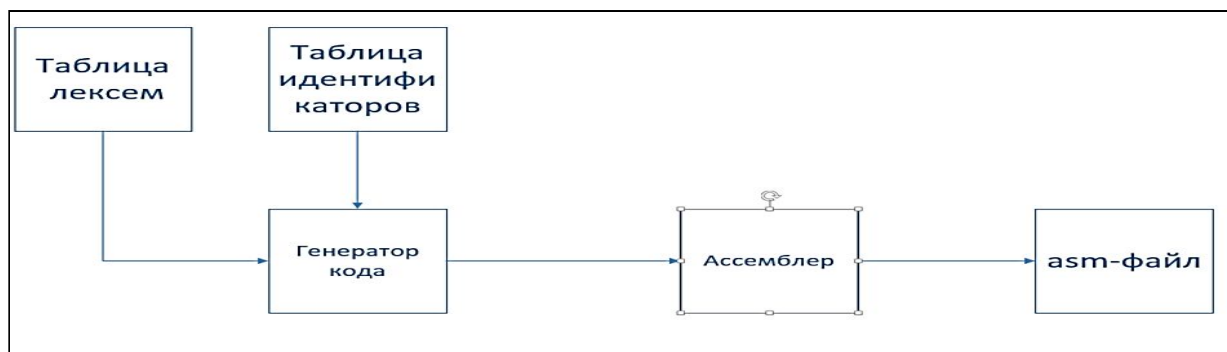


Рисунок 7.2 – Структура генератора кода

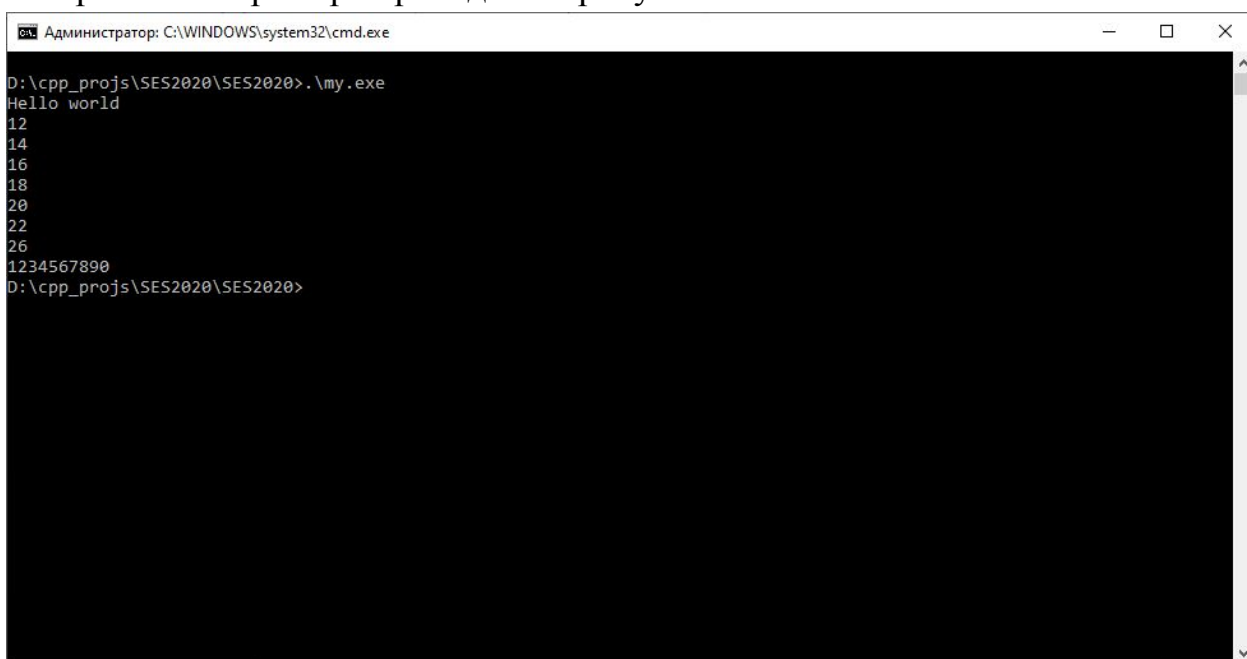
7.5 Входные параметры генератора кода

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке SES-2020. Результаты работы генератора кода выводятся в файл с расширением .asm.

7.6 Контрольный пример

Результат генерации ассемблерного кода на основе контрольного

примера из приложения А приведен в приложении Д. Результат работы контрольного примера приведён на рисунке 7.2.



```
Администратор: C:\WINDOWS\system32\cmd.exe
D:\cpp_proj\SES2020\SES2020>. \my.exe
Hello world
12
14
16
18
20
22
26
1234567890
D:\cpp_proj\SES2020\SES2020>
```

Рисунок 7.2 – Результат работы программы на языке SES-2020

8. Тестирование транслятора

8.1 Тестирование проверки на допустимость символов

В языке SES-2020 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование проверки на допустимость символов

Исходный код	Диагностическое сообщение
<code>main{y}</code>	Ошибка 100: Недопустимый символ в исходном коде (-in), строка 17

8.2 Тестирование лексического анализатора

На этапе лексического анализа в языке SES-2020 могут возникнуть ошибки, описанные в пункте 3.7. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

Исходный код	Диагностическое сообщение
<code>a = 10;</code>	Ошибка 209: Неизвестная переменная, строка 1
<code>po0 int aaaaaaaaaaaa;</code>	Ошибка 208: Слишком длинное имя переменной, строка 1
<code>main{ po0 int x1; }</code>	Ошибка 201: Нераспознанная лексема, строка 17

8.3 Тестирование синтаксического анализатора

На этапе синтаксического анализа в языке SES-2020 могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 – Тестирование синтаксического анализатора

Исходный код	Диагностическое сообщение
<code>main{ twirl(a < b < g){} }</code>	Ошибка 302: Ошибка в выражении, строка 2
<code>main{ twirl(a < b) }</code>	Ошибка 301: Ошибочный сепаратор, строка 2
<code>main{...} dtfi(...){...}</code>	Ошибка 302: Ошибка в выражении, строка 2

Окончание таблицы 8.3

Исходный код	Диагностическое сообщение
main {}	Ошибка 301: Ошибочный сепаратор, строка 3

8.4 Тестирование семантического анализатора

Семантический анализ в языке SES-2020 содержит множество проверок по семантическим правилам, описанным в пункте 1.16. Итоги тестирования семантического анализатора на корректное обнаружение семантических ошибок приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

Исходный код	Диагностическое сообщение
int func jo(int a) { return a } main {return jo(10)}	Ошибка 400: Return не может содержать функцию, строка 6
int func functio(int x, int y) { return x; } main { poo int a; a = functio(10,'123'); return a; }	Ошибка 401: Ошибка в аргументах вызываемой функции функции, строка 8
poo int a; a='hello';	Ошибка 402: Нельзя присвоить значение данного типа, строка 18
twirl('a' < 'f') [out();]	Ошибка 403: Недопустимый тип аргумента цикла, строка 17
int func jo(int a) {return a}	Ошибка 404: Отсутствует точка входа, строка -1
poo int a; poo int a;	Ошибка 405: Переопределение переменной, строка 2

Заключение

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования SES-2020 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка SES-2020;
 2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
 3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
 4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
 5. Осуществлена программная реализация синтаксического анализатора;
 6. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
 7. Разработан транслятор кода на язык ассемблера;
 8. Проведено тестирование всех вышеперечисленных компонентов.
- Окончательная версия языка SES-2020 включает:
1. 2 типа данных;
 2. Поддержка операторов вывода и перевода строки;
 3. Возможность вызова функций стандартной библиотеки;
 4. Наличие 4 арифметических операторов для вычисления выражений;
 5. Поддержка функций, процедур, операторов цикла и условия;
 6. Структурированная и классифицированная система для обработки ошибок пользователя.

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

Список использованных источников

1. Курс лекций по ЯП Наркевич А.С.
2. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – М.: Вильямс, 2003. – 768с.
3. Прата, С. Язык программирования C++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 с.
4. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.
5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с

Приложение А

Листинг 1 – Исходный код программы на языке SES-2020

```
int func functio(int x, int y)
{
    poo int z;
    z = x + y;
    return z;
}

string func nct(string a, string b)
{
    poo string cn;
    cn = a;
    return cn;
}

main
{
    poo int x;
    poo int y;
    poo int z;
    poo int g;
    poo string sa;
    poo string sb;
    poo string sc;
    g = mod(8, 3);
    x = 1;
    y = square(5, 2);
    sa = '1234567890';
    sb = '1234567890';
    z = functio(x,y);
    sc = nct(sa,sb);
    out('complete');
    poo int ab;
    poo int vv;
    ab = 10;
    vv = 20;
    twirl(ab <= vv)
    [
        out();
        ab = ab + 2;
        out(ab);
    ]
}
```

```
    out();  
    out(z);  
    out(sc);  
    return 0;  
}
```

```

ERROR_ENTRY(0, "Недопустимый код ошибки"), // код ошибки вне диапазона 0 - ERROR_MAX_ENTRY
ERROR_ENTRY(1, "Системный сбой"),
ERROR_ENTRY(2, "Не удалось открыть out файл"),
ERROR_ENTRY(3, "Превышена длина входного параметра"),
ERROR_ENTRY(4, "Параметр -in должен быть задан"),
ERROR_ENTRY(5, "Ошибка при открытии файла с исходным кодом (-in)"),
ERROR_ENTRY(6, "Ошибка при создании файла протокола (-log)"),
ERROR_ENTRY_NODEF(7), ERROR_ENTRY_NODEF(8), ERROR_ENTRY_NODEF(9),
ERROR_ENTRY_NODEF10(10), ERROR_ENTRY_NODEF10(20), ERROR_ENTRY_NODEF10(30), ERROR_ENTRY_NODEF10(40), ERROR_ENTRY_NODEF10(50),
ERROR_ENTRY_NODEF10(60), ERROR_ENTRY_NODEF10(70), ERROR_ENTRY_NODEF10(80), ERROR_ENTRY_NODEF10(90),
ERROR_ENTRY(100, "Недопустимый символ в исходном коде (-in)"),
ERROR_ENTRY_NODEF(101),
ERROR_ENTRY_NODEF(102), ERROR_ENTRY_NODEF(103), ERROR_ENTRY_NODEF(104), ERROR_ENTRY_NODEF(105), ERROR_ENTRY_NODEF(106), ERROR_ENTRY_NODEF(107),
ERROR_ENTRY_NODEF(108), ERROR_ENTRY_NODEF(109), ERROR_ENTRY_NODEF10(110), ERROR_ENTRY_NODEF10(120), ERROR_ENTRY_NODEF10(130), ERROR_ENTRY_NODEF10(140),
ERROR_ENTRY_NODEF10(150), ERROR_ENTRY_NODEF10(160), ERROR_ENTRY_NODEF10(170), ERROR_ENTRY_NODEF10(180), ERROR_ENTRY_NODEF10(190),
ERROR_ENTRY(200, "Таблица лексем переполнена"),
ERROR_ENTRY(201, "Нераспознанная лексема"),
ERROR_ENTRY(202, "Таблица идентификаторов переполнена"),
ERROR_ENTRY(203, "Перезапись идентификатора"),
ERROR_ENTRY(204, "Выход за границу таблицы лексем (или идентификаторов)"),
ERROR_ENTRY(205, "Не удалось создать файл с лексемами (или идентификаторами)"),
ERROR_ENTRY(206, "Слишком длинный литерал"),
ERROR_ENTRY(207, "Неверный формат строкового литерала"),
ERROR_ENTRY(208, "Слишком длинное имя переменной"),
ERROR_ENTRY(209, "Неизвестная переменная"),
ERROR_ENTRY(210, "Слишком большой целочисленный литерал"),
ERROR_ENTRY_NODEF(211), ERROR_ENTRY_NODEF(212), ERROR_ENTRY_NODEF(213), ERROR_ENTRY_NODEF(214), ERROR_ENTRY_NODEF(215),
ERROR_ENTRY_NODEF(216), ERROR_ENTRY_NODEF(217), ERROR_ENTRY_NODEF(218), ERROR_ENTRY_NODEF(219),
ERROR_ENTRY_NODEF10(220), ERROR_ENTRY_NODEF10(230), ERROR_ENTRY_NODEF10(240), ERROR_ENTRY_NODEF10(250), ERROR_ENTRY_NODEF10(260),
ERROR_ENTRY_NODEF10(270), ERROR_ENTRY_NODEF10(280), ERROR_ENTRY_NODEF10(290),
ERROR_ENTRY(300, "Неверная структура программы"),
ERROR_ENTRY(301, "Ошибочный сепаратор"),
ERROR_ENTRY(302, "Ошибка в выражении"),
ERROR_ENTRY(303, "Ошибка в параметрах функции"),
ERROR_ENTRY(304, "Ошибка в параметрах вызываемой функции"), ERROR_ENTRY_NODEF(305),
ERROR_ENTRY_NODEF(306), ERROR_ENTRY_NODEF(307), ERROR_ENTRY_NODEF(308), ERROR_ENTRY_NODEF(309), ERROR_ENTRY_NODEF10(310), ERROR_ENTRY_NODEF10(320),
ERROR_ENTRY_NODEF10(330), ERROR_ENTRY_NODEF10(340), ERROR_ENTRY_NODEF10(350), ERROR_ENTRY_NODEF10(360), ERROR_ENTRY_NODEF10(370),
ERROR_ENTRY_NODEF10(380), ERROR_ENTRY_NODEF10(390),
ERROR_ENTRY(400, "Return не может содержать функцию"),
ERROR_ENTRY(401, "Ошибка в аргументах вызываемой функции функции"),
ERROR_ENTRY(402, "Нельзя присвоить значение данного типа"),
ERROR_ENTRY(403, "Недопустимый тип аргумента цикла"),
ERROR_ENTRY(404, "Отсутствует точка входа"),
ERROR_ENTRY(405, "Переопределение переменной"), ERROR_ENTRY_NODEF(406),
ERROR_ENTRY_NODEF(407), ERROR_ENTRY_NODEF(408), ERROR_ENTRY_NODEF(409), ERROR_ENTRY_NODEF10(410),
ERROR_ENTRY_NODEF10(420), ERROR_ENTRY_NODEF10(430), ERROR_ENTRY_NODEF10(440), ERROR_ENTRY_NODEF10(450),
ERROR_ENTRY_NODEF10(460), ERROR_ENTRY_NODEF10(470), ERROR_ENTRY_NODEF10(480), ERROR_ENTRY_NODEF10(490),
ERROR_ENTRY_NODEF100(500), ERROR_ENTRY_NODEF100(600), ERROR_ENTRY_NODEF100(700), ERROR_ENTRY_NODEF100(800)

```

Рисунок 1 – Таблица ошибок языка SES-2020

Приложение Б

Листинг 1 – Таблица идентификаторов контрольного примера

----- Л и т е р а л ы -----				
Т и п д а н н ы х :		З н а ч е н и е : Д л и н а		
с т р о к и :				
integer		8		
integer		3		
integer		1		
integer		5		
integer		2		
string	1234567890		10	
string	complete		8	
integer		10		
integer		20		
integer		0		

----- Ф у н к ц и и -----				
И д е н т и ф и к а т о р : Т и п в о з в р а щ а е м о г о з н а ч е н и я :				
functio	integer			
nct	string			
main	integer			
mod	integer			
square	integer			

----- П е р е м е н н ы е -----				
Б л о к - р о д и т е л ь :		И д е н т и ф и к а т о р : Т и п		
д а н н ы х : Т и п и д е н т и ф и к а т о р а :				
З н а ч е н и е : Д л и н а с т р о к и :				
п а р а м е т р	functio	xfunctio	integer	
			0	
п а р а м е т р	functio	yfunctio	integer	
			0	
п е р е м е н н а я	functio	zfunctio	integer	
			0	
п а р а м е т р	nct	anct	string	
				0
п а р а м е т р	nct	bnct	string	
				0
п е р е м е н н а я	nct	cnct	string	
				0
п е р е м е н н а я	main	xmain	integer	
			0	
п е р е м е н н а я	main	ymain	integer	
			0	
п е р е м е н н а я	main	zmain	integer	
			0	
п е р е м е н н а я	main	gmain	integer	
			0	
п е р е м е н н а я	main	samain	string	
				0
п е р е м е н н а я	main	sbmain	string	
				0
п е р е м е н н а я	main	scmain	string	
				0
п е р е м е н н а я	main	abmain	integer	
			0	
п е р е м е н н а я	main	vvmain	integer	
			0	

И д е н т и ф и к а т о р		И н д е к с п е р в о г о в х о ж д е н и я в Т Л :		

functio	2
xfunctio	5
yfunctio	8
zfunctio	13
nct	27
anct	30
bnct	33
cnnct	38
main	48
xmain	52
ymain	56
zmain	60
gmain	64
samain	68
sbmain	72
scmain	76
mod	23
lit0	82
lit1	84
lit2	89
square	25
lit3	95
lit4	97
lit5	102
lit6	128
abmain	133
vvmain	137
lit7	141
lit8	145
lit9	185

Листинг 2 – Таблица лексем после контрольного примера

-----Таблица

лексем-----

```

1    tfi(ti,ti)
2    {
3    dti;
4    i=i+i;
5    ri;
6    }
7
8    tfi(ti,ti)
9    {
10   dti;
11   i=i;
12   ri;
13   }
14
15   m
16   {
17   dti;
18   dti;
19   dti;
20   dti;
```

```
21   dti;
22   dti;
23   dti;
24   i=k(l,l);
25   i=l;
26   i=s(l,l);
27   i=l;
28   i=l;
29   i=i(i,i);
30   i=i(i,i);
31   o(l);
32
33   dti;
34   dti;
35   i=l;
36   i=l;
37
38   w(ici)
39   [
40   o();
41   i=i+l;
42   o(i);
43   ]
44
45   o();
46   o(i);
47   o(i);
48   rl;
49   }
```

Приложение В

Листинг 1 – Грамматика языка SES-2020

Greibach greibach(

NS('S'), TS('\$'), // стартовый символ, дно стека

6,

Rule(NS('S'), GRB_ERROR_SERIES + 0, // неверная структура
программы

2, // S-> m{NrE;}; | tfi(F){NrE;};S | m{NrE;};S |
tfi(F){NrE;};

Rule::Chain(4, TS('m'), TS('{'), NS('N'), TS('}')),

Rule::Chain(10, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'),
TS('{'), NS('N'), TS('}'), NS('S'))

),

Rule(NS('N'), GRB_ERROR_SERIES + 1, // ошибочный сепаратор

12, // N-> dti; | rE; | i=E; | dtfi(F); | dti;N | rE;N | i=E;N
| dtfi(F);N | pE; pE;N

Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),

Rule::Chain(3, TS('r'), NS('E'), TS(';')),

Rule::Chain(3, TS('r'), NS('E'), TS(';')),

Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),

Rule::Chain(5, TS('o'), TS('('), NS('E'), TS(')'), TS(';')),

Rule::Chain(4, TS('o'), TS('('), TS(')'), TS(';')),

Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),

Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),

Rule::Chain(6, TS('o'), TS('('), NS('E'), TS(')'), TS(';'), NS('N')),

Rule::Chain(5, TS('o'), TS('('), TS(')'), TS(';'), NS('N')),

Rule::Chain(7, TS('w'), TS('('), NS('E'), TS(')'), TS('['), NS('N'),
 TS(']')),
 Rule::Chain(8, TS('w'), TS('('), NS('E'), TS(')'), TS('['), NS('N'),
 TS(']'), NS('N'))
),

Rule(NS('E'), GRB_ERROR_SERIES + 2, // ошибка в выражении

28, // E-> i | 1 | (E) | i(W) | iM | IM | (E)M | i(W)M

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('I')),

Rule::Chain(3, TS('('), NS('E'), TS(')'),

Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')'),

Rule::Chain(2, TS('i'), NS('M')),

Rule::Chain(2, TS('I'), NS('M')),

Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),

Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M')),

Rule::Chain(3, TS('i'), TS('c'), TS('i')),

Rule::Chain(3, TS('i'), TS('c'), TS('I')),

Rule::Chain(3, TS('I'), TS('c'), TS('I')),

Rule::Chain(3, TS('I'), TS('c'), TS('i')),

Rule::Chain(6, TS('s'), TS('('), TS('i'), TS(','), TS('i'), TS(')'),

Rule::Chain(6, TS('s'), TS('('), TS('i'), TS(','), TS('I'), TS(')'),

Rule::Chain(6, TS('s'), TS('('), TS('I'), TS(','), TS('I'), TS(')'),

Rule::Chain(6, TS('s'), TS('('), TS('I'), TS(','), TS('i'), TS(')'),

Rule::Chain(7, TS('s'), TS('('), TS('i'), TS(','), TS('i'), TS(')'),
 NS('M')),

Rule::Chain(7, TS('s'), TS('('), TS('i'), TS(','), TS('I'), TS(')'),

```

NS('M')),
Rule::Chain(7, TS('s'), TS('('), TS('l'), TS(','), TS('l'), TS(')'),
NS('M')),
Rule::Chain(7, TS('s'), TS('('), TS('l'), TS(','), TS('i'), TS(')'),
NS('M')),
Rule::Chain(6, TS('k'), TS('('), TS('i'), TS(','), TS('i'), TS(')'),
Rule::Chain(6, TS('k'), TS('('), TS('i'), TS(','), TS('l'), TS(')'),
Rule::Chain(6, TS('k'), TS('('), TS('l'), TS(','), TS('l'), TS(')'),
Rule::Chain(6, TS('k'), TS('('), TS('l'), TS(','), TS('i'), TS(')'),
Rule::Chain(7, TS('k'), TS('('), TS('i'), TS(','), TS('i'), TS(')'),
NS('M')),
Rule::Chain(7, TS('k'), TS('('), TS('i'), TS(','), TS('l'), TS(')'),
NS('M')),
Rule::Chain(7, TS('k'), TS('('), TS('l'), TS(','), TS('l'), TS(')'),
NS('M')),
Rule::Chain(7, TS('k'), TS('('), TS('l'), TS(','), TS('i'), TS(')'),
NS('M'))
),

```

```

Rule(NS('M'), GRB_ERROR_SERIES + 2, // ошибка в выражении
4, // M-> vE | vEM
Rule::Chain(2, TS('+'), NS('E')),
Rule::Chain(2, TS('-'), NS('E')),
Rule::Chain(3, TS('-'), NS('E'), NS('M')),
Rule::Chain(3, TS('+'), NS('E'), NS('M'))
),

```

```

Rule(NS('F'), GRB_ERROR_SERIES + 3, // ошибка в
параметрах функции

```

```

2,    // F-> ti | ti,F
Rule::Chain(2, TS('t'), TS('i')),
Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))
),

Rule(NS('W'), GRB_ERROR_SERIES + 4, // ошибка в вызываемой
функции

4,    // W-> i | l | i,W | l,W
Rule::Chain(1, TS('i')),
Rule::Chain(1, TS('l')),
Rule::Chain(3, TS('i'), TS(','), NS('W')),
Rule::Chain(3, TS('l'), TS(','), NS('W'))
)

);

```

Листинг 2 Структура магазинного автомата
namespace MFST

```

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

```

```

};

struct Mfst
{
    Parm::PARM parm;
    enum RC_STEP {
        NS_OK,
        NS_NORULE,
        NS_NORULECHAIN,
        NS_ERROR,
        TS_OK,
        TS_NOK,
        LENTA_END,
        SURPRISE
    };
    struct MfstDiagnosis
    {
        short lenta_position;
        RC_STEP rc_step;
        short nrule;
        short nrule_chain;
        MfstDiagnosis();
        MfstDiagnosis(
            short plenta_position,
            RC_STEP prc_step,
            short pnrule,
            short pnrule_chain
        );
    } diagnosis[MFST_DIAGN_NUMBER];
    GRBALPHABET* lenta;
    short lenta_position;
    short nrule;
    short nrulechain;
    short lenta_size;
    GRB::Greibach greibach;
    LT::LexTable lex;
    MFSTSTSTACK st;
    std::vector<MfstState> storestate;
    Mfst();
    Mfst(
        LT::LexTable plex,
        GRB::Greibach pgreibach,
        Parm::PARM parm
    );
};

```

```

char* getCSt(char* buf);
char* getCLenta(char* buf, short pos, short n = 25);
char* getDiagnosis(short n, char* buf);
bool saveState();
bool restState();
bool push_chain(
    GRB::Rule::Chain chain
);
RC_STEP step();
bool start();
bool saveDiagnosis(
    RC_STEP pprc_step
);
void printRules();
struct Deduction
{
    short size;
    short* nrules;
    short* nrulechains;
    Deduction() { size = 0; nrules = 0; nrulechains = 0; };
}deduction;
bool saveDeduction();
};
};

```

Листинг 3 Структура грамматики Грейбах
namespace GRB

```
{
    struct Greibach
    {
        short size;
        GRBALPHABET startN;
        GRBALPHABET stbottomT;
        Rule* rules;
        Greibach() { size = 0; startN = 0; stbottomT = 0; rules = 0; };
        Greibach(
            GRBALPHABET pstartN,
            GRBALPHABET pstBottomT
            short psize,
            Rule r, ...
        );
        short getRule(
            GRBALPHABET pnn,
            Rule& prule
        );
        Rule getRule(short n);
    };
    Greibach getGreibach
};
```

Листинг 4 Разбор исходного кода синтаксическим анализатором

Шаг	Правило	Входная лента	Стек
0	S->tfi(F){N}S	tfi(ti,ti){dti;i=i+i;ri;}	S\$
0	SAVESTATE:	1	
0	:	tfi(ti,ti){dti;i=i+i;ri;}	tfi(F){N}S\$
1	:	fi(ti,ti){dti;i=i+i;ri;}t	fi(F){N}S\$
2	:	i(ti,ti){dti;i=i+i;ri;}tf	i(F){N}S\$
3	:	(ti,ti){dti;i=i+i;ri;}tfi	(F){N}S\$
4	:	ti,ti){dti;i=i+i;ri;}tfi(F){N}S\$
5	F->ti	ti,ti){dti;i=i+i;ri;}tfi(F){N}S\$
5	SAVESTATE:	2	
5	:	ti,ti){dti;i=i+i;ri;}tfi(ti){N}S\$
6	:	i,ti){dti;i=i+i;ri;}tfi(t	i){N}S\$
7	:	,ti){dti;i=i+i;ri;}tfi(ti) {N}S\$

```

8 : TS_NOK/NS_NORULECHAIN
8 : RESSTATE
8 :          ti,ti){dti;i=i+i;ri;}tffi(  F){N}S$
9 : F->ti,F          ti,ti){dti;i=i+i;ri;}tffi(  F){N}S$
9 : SAVESTATE:      2
9 :          ti,ti){dti;i=i+i;ri;}tffi(  ti,F){N}S$
10 :          i,ti){dti;i=i+i;ri;}tffi(t  i,F){N}S$
11 :          ,ti){dti;i=i+i;ri;}tffi(ti  ,F){N}S$
12 :          ti){dti;i=i+i;ri;}tffi(ti,  F){N}S$
13 : F->ti          ti){dti;i=i+i;ri;}tffi(ti,  F){N}S$
13 : SAVESTATE:      3
13 :          ti){dti;i=i+i;ri;}tffi(ti,  ti){N}S$
14 :          i){dti;i=i+i;ri;}tffi(ti,t  i){N}S$
15 :          ){dti;i=i+i;ri;}tffi(ti,ti  ){N}S$
16 :          {dti;i=i+i;ri;}tffi(ti,ti)  {N}S$
17 :          dti;i=i+i;ri;}tffi(ti,ti){  N}S$
18 : N->dti;          dti;i=i+i;ri;}tffi(ti,ti){  N}S$
18 : SAVESTATE:      4
18 :          dti;i=i+i;ri;}tffi(ti,ti){  dti;}S$
19 :          ti;i=i+i;ri;}tffi(ti,ti){d  ti;}S$
20 :          i;i=i+i;ri;}tffi(ti,ti){dt  i;}S$
21 :          ;i=i+i;ri;}tffi(ti,ti){dti  ;}S$
22 :          i=i+i;ri;}tffi(ti,ti){dti;  }S$
23 : TS_NOK/NS_NORULECHAIN
23 : RESSTATE
23 :          dti;i=i+i;ri;}tffi(ti,ti){  N}S$
24 : N->dti;N          dti;i=i+i;ri;}tffi(ti,ti){  N}S$
24 : SAVESTATE:      4
24 :          dti;i=i+i;ri;}tffi(ti,ti){  dti;N}S$
25 :          ti;i=i+i;ri;}tffi(ti,ti){d  ti;N}S$
26 :          i;i=i+i;ri;}tffi(ti,ti){dt  i;N}S$
27 :          ;i=i+i;ri;}tffi(ti,ti){dti  ;N}S$
28 :          i=i+i;ri;}tffi(ti,ti){dti;  N}S$
29 : N->i=E;          i=i+i;ri;}tffi(ti,ti){dti;  N}S$
29 : SAVESTATE:      5
29 :          i=i+i;ri;}tffi(ti,ti){dti;  i=E;}S$
30 :          =i+i;ri;}tffi(ti,ti){dti;i  =E;}S$
31 :          i+i;ri;}tffi(ti,ti){dti;i=  E;}S$
32 : E->i          i+i;ri;}tffi(ti,ti){dti,i=  E;}S$

```

```

32 : SAVESTATE:      6
32 :      i+i;ri;} tfi(ti,ti){dti,i= i;}S$
33 :      +i;ri;} tfi(ti,ti){dti,i=i ;}S$
34 : TS_NOK/NS_NORULECHAIN
34 : RESSTATE
34 :      i+i;ri;} tfi(ti,ti){dti,i= E;}S$
35 : E->i(W)      i+i;ri;} tfi(ti,ti){dti,i= E;}S$
35 : SAVESTATE:      6
35 :      i+i;ri;} tfi(ti,ti){dti,i= i(W);}S$
36 :      +i;ri;} tfi(ti,ti){dti,i=i (W);}S$
37 : TS_NOK/NS_NORULECHAIN
37 : RESSTATE
37 :      i+i;ri;} tfi(ti,ti){dti,i= E;}S$
38 : E->iM      i+i;ri;} tfi(ti,ti){dti,i= E;}S$
38 : SAVESTATE:      6
38 :      i+i;ri;} tfi(ti,ti){dti,i= iM;}S$
39 :      +i;ri;} tfi(ti,ti){dti,i=i M;}S$
40 : M->+E      +i;ri;} tfi(ti,ti){dti,i=i M;}S$
40 : SAVESTATE:      7
40 :      +i;ri;} tfi(ti,ti){dti,i=i +E;}S$
41 :      i;ri;} tfi(ti,ti){dti,i=i; E;}S$
42 : E->i      i;ri;} tfi(ti,ti){dti,i=i; E;}S$
42 : SAVESTATE:      8
42 :      i;ri;} tfi(ti,ti){dti,i=i; i;}S$
43 :      ;ri;} tfi(ti,ti){dti,i=i;r ;}S$
44 :      ri;} tfi(ti,ti){dti,i=i;ri }S$
45 : TS_NOK/NS_NORULECHAIN
45 : RESSTATE
45 :      i;ri;} tfi(ti,ti){dti,i=i; E;}S$
46 : E->i(W)      i;ri;} tfi(ti,ti){dti,i=i; E;}S$

```

Листинг 4 (прод.) Разбор исходного кода синтаксическим анализатором

```

1095: E->i(W)      i);rl;}      E);}$
1095: SAVESTATE:      66
1095:      i);rl;}      i(W));}$
1096:      );rl;}      (W));}$
1097: TS_NOK/NS_NORULECHAIN
1097: RESSTATE

```


1097: i);rl;} E);} \$
 1098: E->iM i);rl;} E);} \$
 1098: SAVESTATE: 66
 1098: i);rl;} iM);} \$
 1099:);rl;} M);} \$
 1100: TNS_NORULECHAIN/NS_NORULE
 1100: RESSTATE
 1100: i);rl;} E);} \$
 1101: E->i(W)M i);rl;} E);} \$
 1101: SAVESTATE: 66
 1101: i);rl;} i(W)M);} \$
 1102:);rl;} (W)M);} \$
 1103: TS_NOK/NS_NORULECHAIN
 1103: RESSTATE
 1103: i);rl;} E);} \$
 1104: E->ici i);rl;} E);} \$
 1104: SAVESTATE: 66
 1104: i);rl;} ici);} \$
 1105:);rl;} ci);} \$
 1106: TS_NOK/NS_NORULECHAIN
 1106: RESSTATE
 1106: i);rl;} E);} \$
 1107: E->icl i);rl;} E);} \$
 1107: SAVESTATE: 66
 1107: i);rl;} icl);} \$
 1108:);rl;} cl);} \$
 1109: TS_NOK/NS_NORULECHAIN
 1109: RESSTATE
 1109: i);rl;} E);} \$
 1110: TNS_NORULECHAIN/NS_NORULE
 1110: RESSTATE
 1110: o(i);rl;} N} \$
 1111: N->o(); o(i);rl;} N} \$
 1111: SAVESTATE: 65
 1111: o(i);rl;} o();} \$
 1112: (i);rl;} ();} \$
 1113: i);rl;});} \$
 1114: TS_NOK/NS_NORULECHAIN
 1114: RESSTATE

1114:	o(i);rl;}	N}\$
1115:	N->o(E);N o(i);rl;}	N}\$
1115:	SAVESTATE: 65	
1115:	o(i);rl;}	o(E);N}\$
1116:	(i);rl;}	(E);N}\$
1117:	i);rl;}	E);N}\$
1118:	E->i i);rl;}	E);N}\$
1118:	SAVESTATE: 66	
1118:	i);rl;}	i);N}\$
1119:);rl;});N}\$
1120:	;rl;}	;N}\$
1121:	rl;}	N}\$
1122:	N->rE; rl;}	N}\$
1122:	SAVESTATE: 67	
1122:	rl;}	rE;}\$
1123:	l;}	E;}\$
1124:	E->l l;}	E;}\$
1124:	SAVESTATE: 68	
1124:	l;}	l;}\$
1125:	;	;}\$
1126:	}	}\$
1127:		\$
1128:	LENTA_END	
1129:	----->LENTA_END	

Приложение Г

Листинг 1 Программная реализация механизма преобразования в ПОЛИЗ

```
int Prior(char ch)
{
    if (ch == '(' || ch == ')')
        return 0;
    else if (ch == ',')
        return 1;
    else if (ch == '+' || ch == '-')
        return 2;
    return -1;
}

void StartPolish(LT::LexTable& lextable, IT::IdTable& idtable, Parm::PARM
parm)
{
    LT::LexTable newTable = LT::Create(lextable.maxsize);
    vector<LT::Entry> temp;
    int semiInd = 0, lextablePos = 0, correction = 0;

    for (int i = 0; i < lextable.size; i++)
    {
        if (lextable.table[i].lexema == LEX_ASSIGN)
        {
            i++;
            correction = 0;
            lextablePos = i;
            int tempSize = 0;
            temp.clear();
            for (; lextable.table[i].lexema != LEX_SEMICOLON; i++)
            {
                temp.push_back(lextable.table[i]);
            }
            semiInd = i;
            tempSize = temp.size();
            ToPolish(temp, idtable, correction);
            if (temp.size() != 0)
                tempSize = temp.size();

            for (int i = lextablePos, k = 0; i < lextablePos + temp.size();
i++, k++)
            {
```

```

        lextable.table[i] = temp[k];
    }
    lextable.table[lextablePos + tempSize] =
lextable.table[semiInd];

    for (int i = 0; i < correction; i++)
    {
        for (int j = lextablePos + temp.size() + 1; j <
lextable.size; j++)          //сдвигаем на лишнее место
        {
            lextable.table[j] = lextable.table[j + 1];
        }
        lextable.size--;
    }

    if (parm.debug)
        printLexT(lextable);
    }
}

```

```

void printLex(LT::LexTable lex)

```

```

{
    for (int i = 0; i < lex.size; i++)
    {
        std::cout << lex.table[i].lexema;
    }
    std::cout << std::endl;
}

```

```

bool ToPolish(vector<LT::Entry>& source, IT::IdTable& idtable, int& correction)

```

```

{
    std::vector<LT::Entry> result;
    std::stack<LT::Entry> stack;
    int elemIndex = 1;
    bool isFunc = false;
    int elemCount = 0;
    LT::Entry func;

    for (int i = 0; i < source.size(); i++)
    {
        switch (source[i].lexema)
        {

```

```

case '-':
case '+':
    if (stack.empty() || stack.top().lexema == '(')
    {
        stack.push(source[i]);
    }

    else
    {
        int symbolPrior = Prior(source[i].lexema);
        while (!stack.empty() && stack.top().lexema != '(' &&
symbolPrior <= Prior(stack.top().lexema))
        {
            result.push_back(stack.top());
            stack.pop();
        }
        stack.push(source[i]);
    }
    break;
case '(':
    stack.push(source[i]);
    break;
case ')':
    while (!stack.empty() && stack.top().lexema != '(')
    {
        result.push_back(stack.top());
        stack.pop();
    }

    if (!stack.empty())
        stack.pop();

    if (isFunc)
    {
        isFunc = false;
        correction += 2 + elemCount - 1 - 2;
        result.push_back({ '@', func.sn, func.idxTI });
        result.push_back({ (char)(elemCount + '0'), func.sn, -1
});

        result.push_back(func);
    }
    break;
case ',':
    while (!stack.empty() && stack.top().lexema != '(')

```

```

        {
            result.push_back(stack.top());
            stack.pop();
        }
        break;
    case 'k':
    case 's':
        while (source[i].lexema != LEX_RIGHTHESIS)
            i++;
        break;
    case 'i':
    case 'l':
        if (isFunc)
        {
            elemCount++;
        }

        if (source.size() - 1 > i ? source[i + 1].lexema == '(' : false &&
idtable.table[source[i].idxTI].idtype == IT::IDTYPE::F)
        {
            isFunc = true;
            elemCount = 0;
            func = source[i];
        }
        else
        {
            result.push_back(source[i]);
        }

        break;
    }
}
while (!stack.empty())
{
    result.push_back(stack.top());
    stack.pop();
}
source = result;
return true;
}

```

Листинг 2 Таблица идентификаторов после преобразования выражений в ПОЛИЗ

----- Литералы -----

Тип данных:	Значение:	Длина строки:
integer	8	
integer	3	
integer	1	
integer	5	
integer	2	
string	1234567890	10
string	complete	8
integer	10	
integer	20	
integer	0	

----- Функции -----

Идентификатор: Тип возвращаемого значения:

functio	integer
nct	string
main	integer
mod	integer
square	integer

----- Переменные -----

Блок-родитель: идентификатора:	Идентификатор:	Тип данных:	Тип
Значение:	Длина строки:		
functio	xfunctio	integer	параметр
0			
functio	yfunctio	integer	параметр
0			
functio	zfunctio	integer	переменная
0			
nct	anct	string	параметр
0			
nct	bnct	string	параметр
0			
nct	cnnct	string	переменная
0			
main	xmain	integer	переменная
0			
main	ymain	integer	переменная
0			
main	zmain	integer	переменная
0			
main	gmain	integer	переменная
0			
main	samain	string	переменная

0		main	sbmain	string	переменная
0		main	scmain	string	переменная
0		main	abmain	integer	переменная
0		main	vvmain	integer	переменная

Идентификатор Индекс первого вхождения в ТЛ:

functio	2
xfunctio	5
yfunctio	8
zfunctio	13
nct	27
anct	30
bnct	33
cnct	38
main	48
xmain	52
ymain	56
zmain	60
gmain	64
samain	68
sbmain	72
scmain	76
mod	23
lit0	82
lit1	84
lit2	89
square	25
lit3	95
lit4	97
lit5	102
lit6	128
abmain	133
vvmain	137
lit7	141
lit8	145
lit9	185

Листинг 3 Таблица лексем после преобразования выражений в ПОЛИЗ

```
1   tfi(ti,ti)
2   {
```



```

3   dti;
4   i=ii+;
5   ri;
6   }
7
8   tfi(ti,ti)
9   {
10  dti;
11  i=i;
12  ri;
13  }
14
15  m
16  {
17  dti;
18  dti;
19  dti;
20  dti;
21  dti;
22  dti;
23  dti;
24  i=k(l,l);
25  i=l;
26  i=s(l,l);
27  i=l;
28  i=l;
29  i=ii@2i;
30  i=ii@2i;
31  o(l);
32
33  dti;
34  dti;
35  i=l;
36  i=l;
37
38  w(ici)
39  [
40  o();
41  i=il+;
42  o(i);
43  ]
44
45  o();
46  o(i);

```

```
47     o(i);  
48     rl;  
49 }
```

Приложение Д

Листинг 1 Результат генерации кода контрольного примера в Ассемблере
.586

```
.model flat, stdcall
includelib libucrt.lib
includelib libucrt.lib
includelib SES2020StaticLib.lib
includelib kernel32.lib
ExitProcess PROTO :DWORD
.stack 4096

printi PROTO : DWORD
prints PROTO : DWORD
octat PROTO : DWORD, : DWORD
elevate PROTO : DWORD, : DWORD
newline PROTO

.const
    lit0 sdword 8
    lit1 sdword 3
    lit2 sdword 1
    lit3 sdword 5
    lit4 sdword 2
    lit5 byte    '1234567890', 0
    lit6 byte    'complete', 0
    lit7 sdword 10
    lit8 sdword 20
    lit9 sdword 0

.data
    zfunctio    sdword    0
    cnnct dword    ?
    xmain       sdword    0
    ymain       sdword    0
    zmain       sdword    0
    gmain       sdword    0
    samain      dword     ?
```

sbmain	dword	?
scmain	dword	?
abmain	sdword	0
vvmain	sdword	0

.code

funcio PROC yfuncio:DWORD, xfuncio:DWORD

```

    push xfuncio
    push yfuncio
    pop eax
    pop ebx
    add eax, ebx
    push eax
    pop ebx
    mov zfuncio, ebx

```

```

    mov eax, zfuncio
    ret

```

funcio ENDP

nct PROC bnct:DWORD, anct:DWORD

```

    push anct
    pop ebx
    mov cnnct, ebx

```

```

    mov eax, cnnct
    ret

```

nct ENDP

main PROC

```

    push lit0
    push lit1
    call octat
    push eax
    pop ebx
    mov gmain, ebx

```

```

    push lit2
    pop ebx

```

mov xmain, ebx

push lit3

push lit4

call elevate

push eax

pop ebx

mov ymain, ebx

push offset lit5

pop ebx

mov samain, ebx

push offset lit5

pop ebx

mov sbmain, ebx

push xmain

push ymain

call functio

push eax

pop ebx

mov zmain, ebx

push samain

push sbmain

call nct

push eax

pop ebx

mov scmain, ebx

push offset lit6

call prints

push lit7

pop ebx

mov abmain, ebx

push lit8

pop ebx

```

        mov vvmain, ebx

twirl_start0:
    mov eax, abmain
    mov ebx, vvmain
    cmp eax, ebx
    jg twirl_end0
    call newline
    push abmain
    push lit4
    pop eax
    pop ebx
    add eax, ebx
    push eax
    pop ebx
    mov abmain, ebx

    push abmain
    call printi
    jmp twirl_start0
twirl_end0:
    call newline
    push zmain
    call printi
    push scmain
    call prints
    push 0
    call ExitProcess
main ENDP
end main

```