

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий
Кафедра Программной инженерии
Специальность 1-40 01 01 Программное обеспечение информационных техноло-
гий
Специализация Программирование интернет-приложений

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

«Разработка компилятора SVY-2020»

Выполнил студент Савченко Владислав Юрьевич
(Ф.И.О.)
Руководитель проекта пр.ст. Пахолко Алена Степановна
(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. Пахолко Алена Степановна
(учен. степень, звание, должность, подпись, Ф.И.О.)
Нормоконтролер пр.ст. Пахолко Алена Степановна
(учен. степень, звание, должность, подпись, Ф.И.О.)
Курсовой проект защищен с оценкой _____

Минск 2020

Содержание

Введение	4
1 Спецификация языка программирования.....	5
1.1 Характеристика языка программирования.....	5
1.2 Алфавит языка.....	5
1.3 Символы сепараторы.....	5
1.4 Применяемые кодировки	6
1.5 Типы данных	6
1.6 Преобразование типов данных	7
1.7 Идентификаторы	7
1.8 Литералы.....	7
1.9 Объявление данных	7
1.10 Инициализация данных.....	8
1.11 Инструкции языка.....	8
1.12 Операции языка.....	8
1.13 Выражения и их вычисление	9
1.14 Программные конструкции языка.....	9
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	16
3.1 Структура лексического анализатора.....	16
3.2. Контроль входных символов	17
3.3 Удаление избыточных символов.....	17
3.4 Перечень ключевых слов	18
3.5 Основные структуры данных	20
3.6 Принцип обработки ошибок.....	21
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.....	22

4. Разработка синтаксического анализатора	23
4.1 Структура синтаксического анализатора	23
4.2 Контекстно-свободная грамматика, описывающая синтаксис языка	23
4.3 Построение конечного магазинного автомата.....	25
4.4 Основные структуры данных	26
4.5 Описание алгоритма синтаксического разбора	26
4.6 Структура и перечень сообщений синтаксического анализатора	27
4.7. Параметры синтаксического анализатора и режимы его работы	27
4.8. Принцип обработки ошибок	27
4.9. Контрольный пример.....	27
5 Разработка семантического анализатора.....	28
5.1 Структура семантического анализатора.....	28
5.2 Функции семантического анализатора	28
5.3 Структура и перечень сообщений семантического анализатора.....	28
5.4 Принцип обработки ошибок	29
5.5 Контрольный пример.....	29
6. Преобразование выражений	30
6.1 Выражения, допускаемые языком.....	30
6.2 Польская запись и принцип её построения.....	30
6.3 Программная реализация обработки выражений	30
6.4 Контрольный пример.....	31
7. Генерация кода	32
7.1 Структура генератора кода	32
7.2 Представление типов данных в оперативной памяти	32
7.3 Статическая библиотека.....	33
7.4 Особенности алгоритма генерации кода	33
7.5 Входные параметры генератора кода	33
7.6 Контрольный пример.....	34
8. Тестирование транслятора	35
8.1 Тестирование проверки на допустимость символов	35
8.2 Тестирование лексического анализатора	35
8.3 Тестирование синтаксического анализатора	35
8.4 Тестирование семантического анализатора	36
Заключение	38
Литература	39
Приложение А	40
Приложение Б.....	440
Приложение В	47
Приложение Г	54
Приложение Д	62

Введение

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

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

Транслятор SVY-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;
- разработка структуры транслятора;
- разработка лексического и семантического анализаторов;
- разработка синтаксического анализатора;
- преобразование выражений;
- генерация кода на 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 SVY-2020 является процедурным, универсальным, строго типизированным, компилируемым и не объектно-ориентированным языком.

1.2 Алфавит языка

Алфавит языка SVY-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 функции или условной конструкции/цикла
(...)	Блок фактических или формальных параметров функции, а также приоритет арифметических операций

Окончание таблицы 1.1

Сепаратор(ы)	Назначение
,	Разделитель параметров функций
#	Символ, отделяющий условные конструкции/циклы
+ - */	Арифметические операции
> < & !	Логические операции (операции сравнения: больше, меньше, проверка на равенство, на неравенство), используемые в условии цикла/условной конструкции.
;	Разделитель программных конструкций
} {	Операторы сдвигов
=	Оператор присваивания

1.4 Применяемые кодировки

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

1.5 Типы данных

В языке SVY-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 SVY-2020

Тип данных	Описание типа данных
Целочисленный тип данных <i>number</i>	Фундаментальный тип данных. Используется для работы с числовыми значениями. В памяти занимает 1 байт. Максимальное значение: 127. Минимальное значение: -128. Инициализация по умолчанию: значение 0.
Строковый тип данных <i>string</i>	Фундаментальный тип данных. Используется для работы с символами, каждый символ в памяти занимает 1 байт. Максимальное количество символов: 255. Инициализация по умолчанию: строка нулевой длины "".

1.6 Преобразование типов данных

Преобразование типов данных в языке SVY-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 - 8 символов. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Префикс занимает 8 дополнительных символов. В случае превышения заданной длины, идентификаторы усекаются до длины, равной 16 символов (8 символов на имя идентификатора, 8 символов на префикс). Данные правила действуют для всех типов идентификаторов. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, параметр функции. Имена идентификаторов-функций не должны совпадать с именами команд ассемблера (это не касается имён идентификаторов-переменных).

1.8 Литералы

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

Таблица 1.3 – Описание литералов

Тип литерала	Описание
Целочисленные литералы в десятичном представлении	Последовательность цифр [0...9] с предшествующим знаком минус или без него (знак минус не должен отделяться пробелом)
Строковые литералы	Набор символов (от 1 до 255), заключённых в двойные кавычки

Ограничения на строковые литералы языка SVY-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; если литерал отрицательный, после знака «-» не может быть знака ноль.

1.9 Объявление данных

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

при объявлении.

Пример объявления числового типа с инициализацией:

now number value1 = -2

Пример объявления переменной символьного типа с инициализацией:

now string str1 = "hello world"

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

1.10 Инициализация данных

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

1.11 Инструкции языка

Инструкции языка SVY-2020 представлены в таблице 1.4

Таблица 1.4 – Инструкции языка SVY-2020

Инструкция	Запись на языке SVY-2020
Объявление переменной	<i>now</i> <тип данных> <идентификатор>;
Объявление переменной инициализацией	<i>now</i> <тип данных> <идентификатор> = <значение>; Значение – инициализатор конкретного типа. Может быть только литералом или идентификатором
Возврат из функции или процедуры	Для функций, возвращающих значение: <i>conclusion</i> <идентификатор/литерал>; Для процедур: <i>conclusion</i> ;
Вывод данных	<i>zapishi</i> <идентификатор/литерал>;
Вывод данных	<i>read</i> <идентификатор >;
Вызов функции или процедуры	<идентификатор функции> (<список параметров>); Список параметров может быть пустым.
Перевод строки	<i>newline</i> ;
Присваивание	<идентификатор> = <выражение>;

1.12 Операции языка

В языке SVY-2020 предусмотрены следующие операции с данными. Приоритетность операции умножения и деления выше приоритета операций сложения и

вычитания. Для установки наивысшего приоритета используются круглые скобки. Операции языка представлены в таблице 1.5.

Таблица 1.5 – Операции языка SVY-2020

Тип оператора	Оператор
Арифметические	1. + – сложение 2. - – вычитание 3. * – умножение 4. / – деление 5. = – присваивание
Строковые	1. = – присваивание
Логические	1. > – больше 2. < – меньше 3. ! – проверка на неравенство
Сдвиговые	1. } – сдвиг вправо 2. { – сдвиг влево

1.13 Выражения и их вычисление

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

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

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

1.14 Программные конструкции языка

Программа на языке SVY-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 SVY-2020

Конструкция	Запись на языке SVY-2020
Главная функция(точка входа)	<i>poejali</i> [...]

Окончание таблицы 1.6

Конструкция	Запись на языке SVY-2020
Цикл	<i>condition:</i> <идентификатор1> <оператор> <идентификатор2> # <i>cicle</i> [...] #
Внешняя функция	<тип данных> <i>operation</i> <идентификатор> (<тип> <идентификатор>, ...) [... <i>conclusion</i> <идентификатор/литерал>;]
Внешняя процедура	<i>procedure operation</i> <идентификатор> (<тип> <идентификатор>, ...) [... <i>conclusion</i> ;]
Условная конструкция	<i>condition:</i> <идентификатор1> <оператор> <идентификатор2> # <i>istrue</i> [...] <i>isfalse</i> [...] # При истинности условия выполняется код внутри блока <i>istrue</i> , иначе – код внутри блока <i>isfalse</i> . В коде так же представлен закрывающим условную конструкцию символом ‘#’.
Внешняя процедура	<i>procedure operation</i> <идентификатор> (<тип> <идентификатор>, ...) [... <i>conclusion</i> ;]

1.15 Области видимости идентификаторов

В языке SVY-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 префикс – 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 SVY-2020 выполняются следующие семантические проверки приведённых в таблице 1.8.

Таблица 1.8 – Семантические проверки

Номер	Правило
1	Наличие функции – точки входа в программу
2	Единственность точки входа
3	Переопределение идентификаторов
4	Использование идентификаторов без их объявления
5	Проверка соответствия типа функции и возвращаемого параметра
6	Правильность передаваемых в функцию параметров: количество, типы
7	Правильность строковых выражений
8	Превышение размера строковых и числовых литералов
9	Правильность составленного условия цикла/условного оператора

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 оттранслированным в язык ассемблера коде переменные имеют глобальную область видимости.

1.18 Стандартная библиотека и её состав

В языке SVY-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>string</i> concat(<i>string</i> s1, <i>string</i> s2);	Строковая функция, выполняет объединение строк s1 и s2 в указанном порядке. Возвращает значение в переменной buffer.
<i>number</i> atoi(<i>string</i> str);	Целочисленная функция. Преобразует строку в число
<i>number</i> length(<i>string</i> str);	Целочисленная функция. Вычисляет и возвращает длину строки str

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 *zapishi*. Эти функции представлены в таблице 1.10.

Таблица 1.10 – Дополнительные функции стандартной библиотеки

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

1.19 Ввод и вывод данных

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

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

1.20 Точка входа

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

1.21 Препроцессор

Препроцессор в языке программирования SVY-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 SVY-2020 транслируется в язык ассемблера, а затем - в объектный код.

1.24 Классификация сообщений транслятора

В случае возникновения ошибки в коде программы на языке SVY-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

Интервал	Описание ошибок
0 – 200	Системные ошибки
200 – 299	Ошибки лексического анализа
300 – 399	Ошибки семантического анализа
600 – 699	Ошибки синтаксического анализа
400-499, 700-999	Зарезервированные коды ошибок

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 SVY-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.

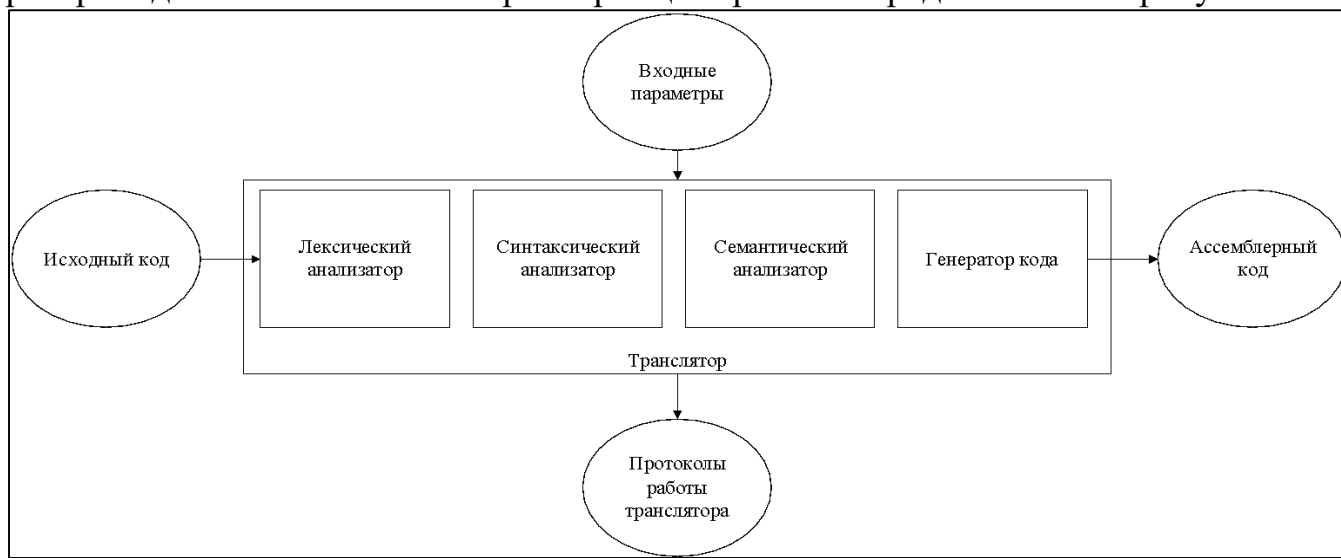


Рисунок 2.1 – Структура транслятора языка программирования SVY-2020

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 SVY-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 SVY-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 SVY-2020

Входной параметр	Описание	Значение по умолчанию
-in:<имя_файла>	Входной файл с расширением .txt, в котором содержится исходный код на SVY-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 SVY-2020. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора.	<имя_файла>.log

2.3 Перечень протоколов, формируемых транслятором и их содержимое

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

Таблица 2.2 – Протоколы, формируемые транслятором языка SVY-2020

Формируемый протокол	Описание протокола
Файл журнала, заданный параметром "-log:"	Файл содержит в себе краткую информацию об исходном коде на языке SVY-2020. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора.
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, преобразующий единый массив текстовых символов в массив токенов.

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

Функции лексического анализатора:

- удаление «пустых» символов 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.

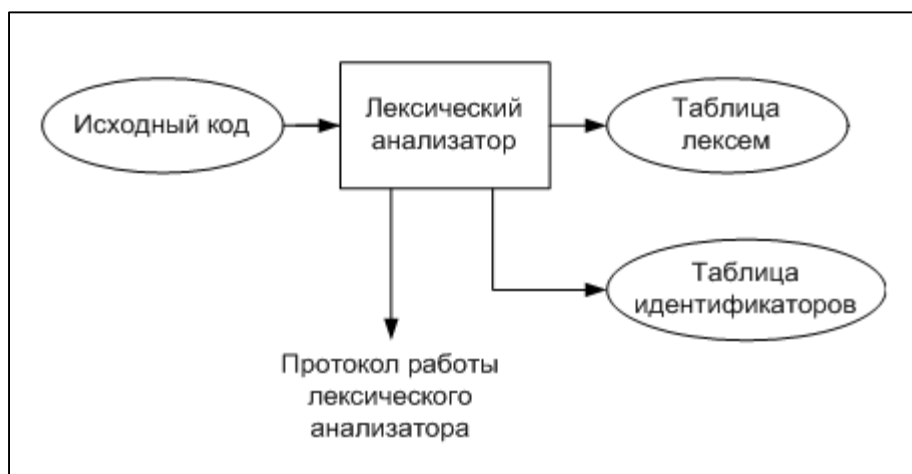


Рисунок 3.1 – Структура лексического анализатора

3.2. Контроль входных символов

Исходный код на языке программирования SVY-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.

[illegible]

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
Игнорируемый	I
Литерал	Q
Сепаратор	S
Перевод строки	N
Пробел, табуляция	P

3.3 Удаление избыточных символов

Удаление избыточных символов не предусмотрено, так как после проверки на допустимость символов исходный код на языке программирования SVY-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
roexali	m	Главная функция.
now	n	Объявление переменной.
operation	f	Объявление функции.
procedure	p	Указывается перед словом operation. Ключевое слово для процедур – функций, не возвращающих значения.
zapishi	@	Ввод данных.
newline	^	Оператор вывода символа перевода строки.
istrue	r	Истинная ветвь условного оператора.
isfalse	w	Ложная ветвь условного оператора.
number, string	t	Названия типов данных языка.
Идентификатор	i	Длина идентификатора – 8 символов.
Литерал	l	Литерал любого доступного типа.
conclusion	e	Выход из функции/процедуры.
condition:	?	Указывает начало цикла/условного оператора.
#	#	Разделение конструкций в цикле/условном операторе.
;	;	Разделение выражений.
,	,	Разделение параметров функций.
+	+	Знаки операций.
-	-	
*	*	
/	/	
>	>	Знаки логических операторов
<	<	
!	!	
[[Начало блока/тела функции.
]]	Закрытие блока/тела функции.
}	}	Знаки сдвиговых операций.
{	{	

Окончание таблицы 3.2

Токен	Лексема	Пояснение
((Передача параметров в функцию, приоритет операций.
))	Закрытие блока для передачи параметров, приоритет операций.
=	=	Знак присваивания.

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 GRAPH_POEXALI 8, \
    FST::NODE(1,FST::RELATION('p',1)),\
    FST::NODE(1,FST::RELATION('o',2)),\
    FST::NODE(1,FST::RELATION('e',3)),\
    FST::NODE(1,FST::RELATION('x',4)),\
    FST::NODE(1,FST::RELATION('a',5)),\
    FST::NODE(1,FST::RELATION('l',6)),\
    FST::NODE(1,FST::RELATION('i',7)),\
    FST::NODE()

```

Рисунок 3.4 – Пример реализации графа конечного автомата для токена *poexali* (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 C++ со структурой таблицы лексем представлен на рисунке 3.3. Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.4.

```

namespace LT    //таблица лексем
{
    struct Entry
    {
        char lexema;
        int sn;
        int idxTI;

        Entry();
        Entry(char lexema, int snn, int idxti = NULLDX_TI);
    };

    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, лексический анализатор игнорирует найденную ошибку и продолжает работу с исходным кодом. Перечень сообщений представлен на рисунке 3.5.

```

ERROR_ENTRY(200, "Лексическая ошибка: Недопустимый символ в исходном файле(-in)"),
ERROR_ENTRY(201, "Лексическая ошибка: Неизвестная последовательность символов"),
ERROR_ENTRY(202, "Лексическая ошибка: Превышен размер таблицы лексем"),
ERROR_ENTRY(203, "Лексическая ошибка: Превышен размер таблицы идентификаторов"),
ERROR_ENTRY_NODEF(204), ERROR_ENTRY_NODEF(205), ERROR_ENTRY_NODEF(206), ERROR_ENTRY_NODEF(207), ERROR_ENTRY_NODEF(208), ERROR_ENTRY_NODEF(209),
ERROR_ENTRY_NODEF10(210), 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 входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением. Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

3.8 Параметры лексического анализатора

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся как в файл журнала, 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 заносится 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.

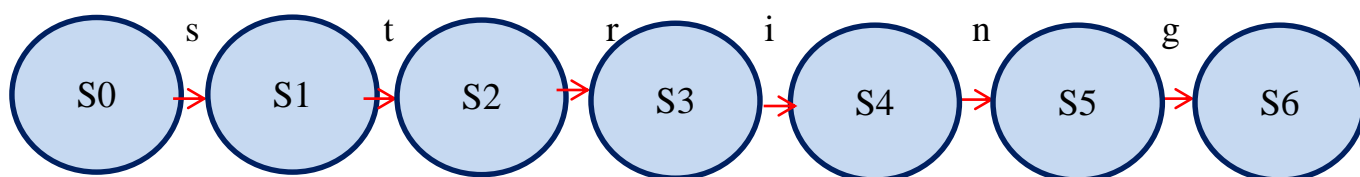


Рисунок 3.2 – Граф переходов для цепочки “string”

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.

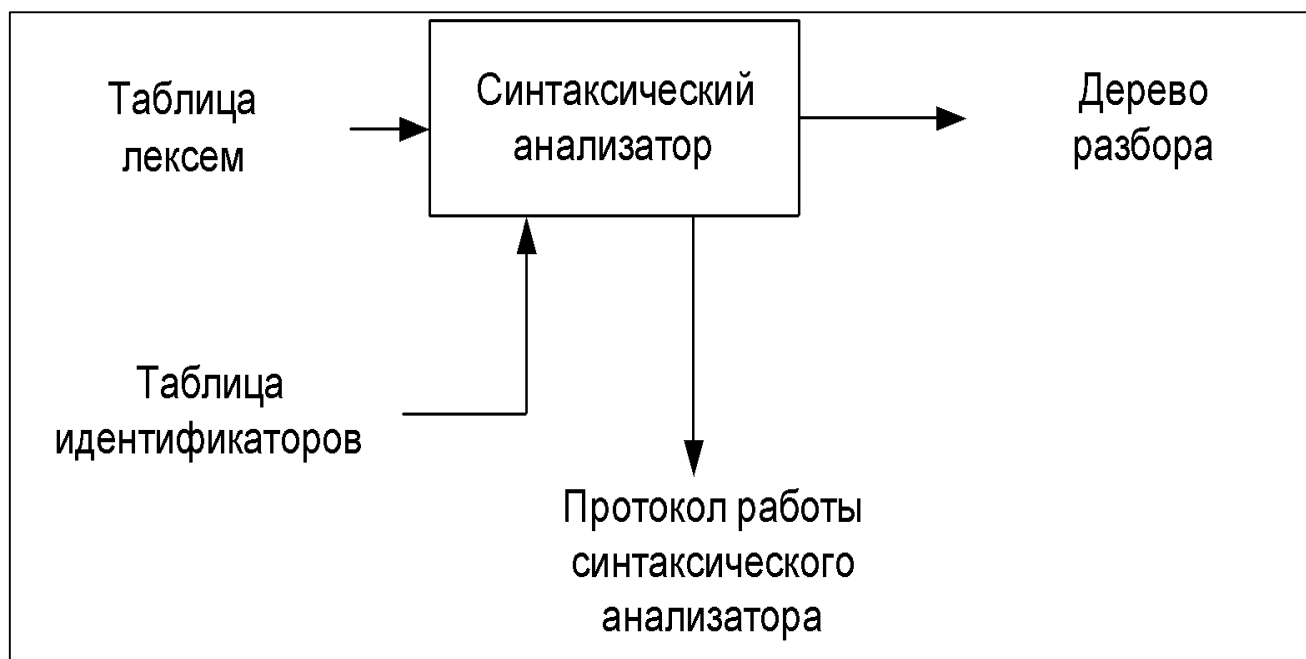


Рисунок 4.1 – Структура синтаксического анализатора.

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка SVY-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->tfiPTS S->pfiPGS S->m[K]	Стартовые правила, описывающее общую структуру программы
P	P->(E) P->()	Правила для параметров объявляемых функций
T	T->[eV;] T->[KeV;]	Правила для тела функций
G	G->[e;] G->[Ke;]	Правила для тела процедур
E	E->ti,E E->ti	Правила для списка параметров функции
F	F->(N) F->()	Правила для вывозов функций(в т.ч. и в выражениях)
N	N->i N>l N->l,N N->I,N	Правила для параметров вызываемых функций
R	R->rY# R>wY# R>cY# R->rYwY# R->wYrY#	Правила составления цикла/условного оператора
Z	Z->iLi Z->iLl Z->lli	Правила для условия цикла/условного оператора
W	W->l W->i W->(W) W->(W)AW W->iF W->iAW W->lAW W->iFAW	Правила для сложных выражений
A	A->+ A->- A->* A->/ A->{ A->}	Правила для арифметических и сдвиговых операторов

Окончание таблицы 4.1

Символ	Правила	Какие правила порождает
V	V->l V->i V->q	Правила для простых выражений
Y	Y->[X]	Правила для тела цикла/условного выражения
L	L->< L->> L->!	Правила для логических операторов
K	K->nti=V;K K->nti;K K->i=W;K K->oV;K K->^;K K->&Z#RK K->iF;K K->nti=V; K->nti; K->i=W; K->oV; K->^; K->&Z#R K->iF;	Программные конструкции
X	X->i=W;X X->oV;X X->^;X X->iF;X X->i=W; X->oV; X->^; X->iF;	Программные конструкции внутри цикла/условного оператора

4.3 Построение конечного магазинного автомата

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

Таблица 4.2 – Описание компонентов магазинного автомата

Компонента	Определение	Описание
Q	Множество состояний автомата	Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата

Окончание таблицы 4.2

V	Алфавит входных символов	Алфавит представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблица 3.1 и 4.1.
Z	Алфавит специальных магазинных символов	Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ \$)
δ	Функция переходов автомата	Функция представляет из себя множество правил грамматики, описанных в таблице 4.1.
q_0	Начальное состояние автомата	Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики
z_0	Начальное состояние магазина автомата	Символ маркера дна стека \$
F	Множество конечных состояний	Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты

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 SVY-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(600, "Синтаксическая ошибка: Неверная структура программы"),
ERROR_ENTRY(601, "Синтаксическая ошибка: Не найден список параметров функции"),
ERROR_ENTRY(602, "Синтаксическая ошибка: Ошибка в теле функции"),
ERROR_ENTRY(603, "Синтаксическая ошибка: Ошибка в теле процедуры"),
ERROR_ENTRY(604, "Синтаксическая ошибка: Ошибка в списке параметров функции"),
ERROR_ENTRY(605, "Синтаксическая ошибка: Ошибка в вызове функции/выражении"),
ERROR_ENTRY(606, "Синтаксическая ошибка: Ошибка в списке фактических параметров функции"),
ERROR_ENTRY(607, "Синтаксическая ошибка: Ошибка при конструировании цикла/условного выражения"),
ERROR_ENTRY(608, "Синтаксическая ошибка: Ошибка в теле цикла/условного выражения"),
ERROR_ENTRY(609, "Синтаксическая ошибка: Ошибка в условии цикла/условного выражения"),
ERROR_ENTRY(610, "Синтаксическая ошибка: Неверный условный оператор"),
ERROR_ENTRY(611, "Синтаксическая ошибка: Неверный арифметический оператор"),
ERROR_ENTRY(612, "Синтаксическая ошибка: Неверное выражение. Ожидаются только идентификаторы/литералы"),
ERROR_ENTRY(613, "Синтаксическая ошибка: Ошибка в арифметическом выражении"),
ERROR_ENTRY(614, "Синтаксическая ошибка: Недопустимая синтаксическая конструкция"),
ERROR_ENTRY(615, "Синтаксическая ошибка: Недопустимая синтаксическая конструкция в теле цикла/условного выражения"),
```

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.

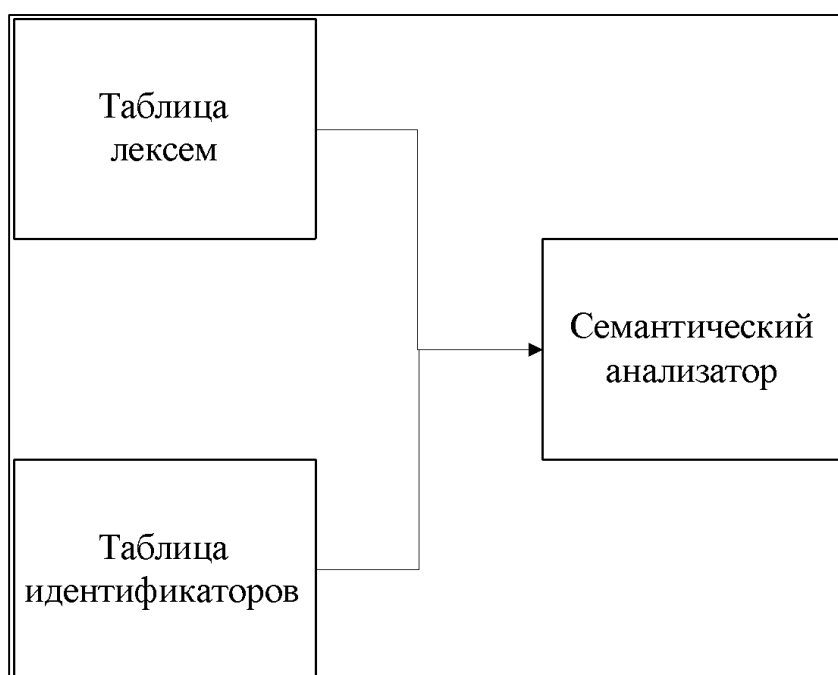


Рисунок 5.1. – Структура семантического анализатора

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(300, "Семантическая ошибка: Необъявленный идентификатор"),
ERROR_ENTRY(301, "Семантическая ошибка: Отсутствует точка входа main"),
ERROR_ENTRY(302, "Семантическая ошибка: Обнаружено несколько точек входа main"),
ERROR_ENTRY(303, "Семантическая ошибка: В объявлении не указан тип идентификатора"),
ERROR_ENTRY(304, "Семантическая ошибка: В объявлении отсутствует ключевое слово new"),
ERROR_ENTRY(305, "Семантическая ошибка: Попытка переопределения идентификатора"),
ERROR_ENTRY(306, "Семантическая ошибка: Превышено максимальное количество параметров функции"),
ERROR_ENTRY(307, "Семантическая ошибка: Слишком много параметров в вызове"),
ERROR_ENTRY(308, "Семантическая ошибка: Кол-во ожидаемых функций и передаваемых параметров не совпадают"),
ERROR_ENTRY(309, "Семантическая ошибка: Несовпадение типов передаваемых параметров"),
ERROR_ENTRY(310, "Семантическая ошибка: Использование пустого строкового литерала недопустимо"),
ERROR_ENTRY(311, "Семантическая ошибка: Обнаружен символ '\\\"'. Возможно, не закрыт строковый литерал"),
ERROR_ENTRY(312, "Семантическая ошибка: Превышен размер строкового литерала"),
ERROR_ENTRY(313, "Семантическая ошибка: Недопустимый целочисленный литерал"),
ERROR_ENTRY(314, "Семантическая ошибка: Типы данных в выражении не совпадают"),
ERROR_ENTRY(315, "Семантическая ошибка: Тип функции и возвращаемого значения не совпадают"),
ERROR_ENTRY(316, "Семантическая ошибка: Недопустимое строковое выражение справа от знака '='"),
ERROR_ENTRY(317, "Семантическая ошибка: Неверное условное выражение"),
ERROR_ENTRY(318, "Семантическая ошибка: Деление на ноль"),

```

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> роехали[number b = 11; запishi b;] </pre>	Ошибка N304: Семантическая ошибка: В объявлении отсутствует ключевое слово new Строка: 2
<pre> роехали[now number b = 9; now string y = b;] </pre>	Ошибка N314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 3
<pre> роехали [now number b = 9;] роехали [now string y = "privet";] </pre>	Ошибка N302: Семантическая ошибка: Обнаружено несколько точек входа роехали Строка: 0

6. Вычисление выражений

6.1 Выражения, допускаемые языком

В языке SVY-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
()	3
*	2
/	2
+	1
-	1
}	0
{	0

6.2 Польская запись и принцип её построения

Все выражения языка SVY-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
$i[2]=(((l[3]+l[4])-i[0])*l[5])/l[6];$	$i[2]=l[3]l[4]+i[0]-l[5]*l[6]/$
$i[23]=(i[23]+l[26])*l[26]$	$i[23]=i[23]l[26]+l[26]*$
$i[3]=(((l[4]+l[5])-i[0])*l[6])$	$i[3]=l[4]l[5]+i[0]-l[6]*$

7. Генерация кода

7.1 Структура генератора кода

В языке SVY-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 SVY-2020 представлена на рисунке 7.1.

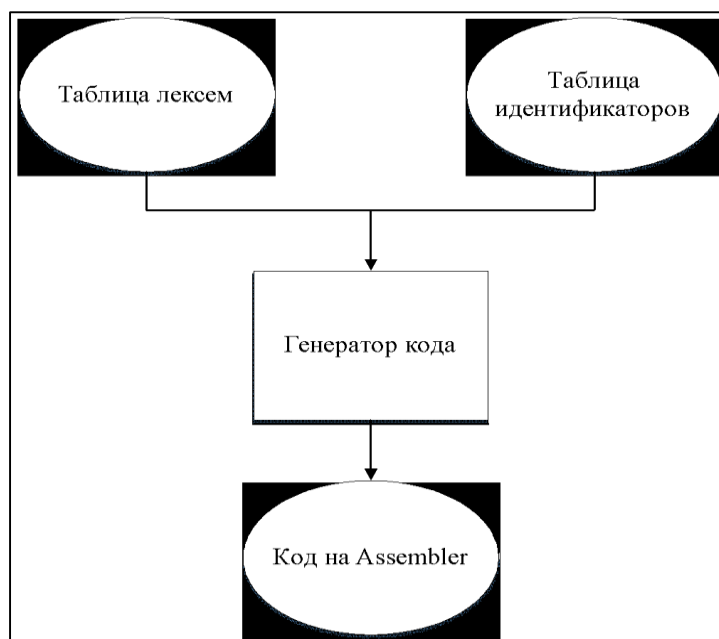


Рисунок 7.1 – Структура генератора кода

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 SVY-2020 и на языке ассемблера приведены в таблице 7.1.

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

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

7.3 Статическая библиотека

В языке SVY-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 outlstr(char* str)	Вывод на консоль строки str
void outnum(int num)	Вывод на консоль целочисленной переменной num
int lenght(char* buffer, char* str)	Вычисление длины строки
char* concat(char* buffer, char* str1, char* str2)	Объединение строк str1 и str2
int atoi(char* ptr)	Преобразование строки в число

7.4 Особенности алгоритма генерации кода

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

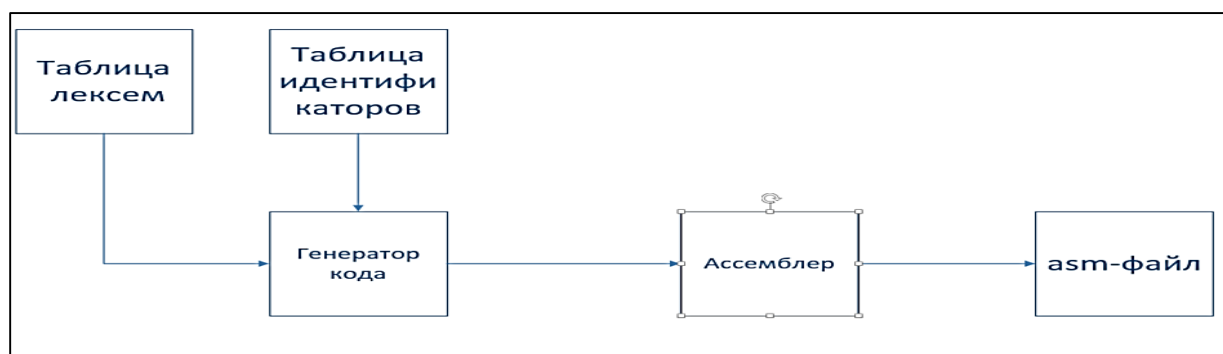



Рисунок 7.2 – Структура генератора кода

7.5 Входные параметры генератора кода

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке SVY-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.



```
Microsoft Visual Studio
Консоль отладки Microsoft Visual Studio

iz srtoki v chislo:15
-31
sdvig vlevo:18
-9 output PROTO : DWORD
5 16 38 82
LenghtA:8 PROTO : DWORD
concat:Nebeute pogaluista

D:\Kursach\SVY2020\Generation\Debug\Generation.exe (процесс 6748) завершил работу с кодом 0.
Что бы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно.

.const
    newline byte 13, 10, 0
    VALS1 byte 'LenghtA:', 0
    VALS2 byte 'concat:', 0
    VALS3 sdword 9
    VALS4 sdword -9
    VALS5 byte 'Nebeute ', 0
    VALS6 byte 'pogaluista', 0
    VALS7 byte '15December', 0
    VALS8 byte 'iz srtoki v chislo:', 0
    VALS9 sdword 40
    VALS10 sdword 1
    VALS11 byte 'sdvig vlevo:', 0
    VALS12 sdword 5
```

Рисунок 7.2 – Результат работы программы на языке SVY-2020

8. Тестирование транслятора

8.1 Тестирование проверки на допустимость символов

В языке SVY-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
роехали [ë]	Ошибка N200: Лексическая ошибка: Недопустимый символ в исходном файле(-in) Строка: 2 Позиция в строке: 2

8.2 Тестирование лексического анализатора

На этапе лексического анализа в языке SVY-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
роехали [now number x11;]	Ошибка N201: Лексическая ошибка: Неизвестная последовательность символов Строка: 3

8.3 Тестирование синтаксического анализатора

На этапе синтаксического анализа в языке SVY-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
роехали now number x;]	Ошибка 600: строка 1, Синтаксическая ошибка: Неверная структура программы
string operation fi([роехали[]	Ошибка 601: строка 1, Синтаксическая ошибка: Не найден список параметров функции
string operation fi() [newline; write] роехали[]	Ошибка 602: строка 1, Синтаксическая ошибка: Ошибка в теле функции
procedure operation fi() [newline write;] роехали[]	Ошибка 603: строка 1, Синтаксическая ошибка: Ошибка в теле процедуры

Окончание таблицы 8.3

Исходный код	Диагностическое сообщение
procedure operation fi(number number)[] роехали []	Ошибка 604: строка 1, Синтаксическая ошибка: Ошибка в списке параметров функции
string operation fi(num- ber x)[conclusion 3;] роехали [newline;fi(5,5; newline;]	Ошибка 605: строка 2, Синтаксическая ошибка: Ошибка в вызове функции/выражении
string operation fi(num- ber x)[conclusion 3;] роехали[newline;fi(5,5,5 5);]	Ошибка 606: строка 2, Синтаксическая ошибка: Ошибка в списке фактических параметров функции
роехали [now number x; condition: x > 2 # cicle #]	Ошибка 607: строка 1, Синтаксическая ошибка: Ошибка при конструировании цикла/условного выра- жения
роехали [now number x; condition: x > 2 # cicle #]	Ошибка 608: строка 1, Синтаксическая ошибка: Ошибка в теле цикла/условного выражения
роехали [condition: 1 = 2 #]	Ошибка 609: строка 1, Синтаксическая ошибка: Ошибка в условии цикла/условного выражения
роехали [condition: 1 = 2 #]	Ошибка 610: строка 1, Синтаксическая ошибка: Невер- ный условный оператор
роехали [now number x; x = x ! x;]	Ошибка 611: строка 1, Синтаксическая ошибка: Невер- ный арифметический оператор
роехали [now number x; zapishi now;]	Ошибка 612: строка 1, Синтаксическая ошибка: Невер- ное выражение. Ожидаются только идентифика- торы/литералы
роехали [now number x; x = 1 ++;]	Ошибка 613: строка 1, Синтаксическая ошибка: Ошибка в арифметическом выражении
роехали [newline; 4;]	Ошибка 614: строка 1, Синтаксическая ошибка: Недо- пустимая синтаксическая конструкция
роехали [now number a; condition: a < 3 # istrue [newline; 3;] #]	Ошибка 615: строка 1, Синтаксическая ошибка: Недо- пустимая синтаксическая конструкция в теле цикла/условного выражения

8.4 Тестирование семантического анализатора

Семантический анализ в языке SVY-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
poexali [a = 1]	Ошибка N300: Семантическая ошибка: Необъявленный идентификатор Строка: 1
string function fi()[]	Ошибка N301: Семантическая ошибка: Отсутствует точка входа main Строка: 0
poexali [] poexali []	Ошибка N302: Семантическая ошибка: Обнаружено несколько точек входа main Строка: 0
poexali [a = 1;]	Ошибка N304: Семантическая ошибка: В объявлении отсутствует ключевое слово new Строка: 1
poexali [now number t; now string t;]	Ошибка N305: Семантическая ошибка: Попытка переопределения идентификатора Строка: 3
procedure operation fi()[] poexali [fi("a","b","c","d")]	Ошибка N307: Семантическая ошибка: Слишком много параметров в вызове Строка: 1
string operation fi(string x, string y, string z, string s) poexali []	Ошибка N306: Семантическая ошибка: Превышено максимальное количество параметров функции Строка: 1
string operation fi(string x)[conclusion "a";] poexali [fi("a", "b");]	Ошибка N308: Семантическая ошибка: Кол-во ожидаемых функцией и передаваемых параметров не совпадают Строка: 2
string operation fi(string x)[conclusion "a";] poexali [fi("a", "b");]	Ошибка N309: Семантическая ошибка: Несовпадение типов передаваемых параметров Строка: 2
poexali [now string x="";]	Ошибка N310: Семантическая ошибка: Использование пустого строкового литерала недопустимо Строка: 1
poexali [now string x=";]	Ошибка N311: Семантическая ошибка: Обнаружен символ "". Возможно, не закрыт строковый литерал Строка: 1
poexali [now number x=995995959595959;]	Ошибка N313: Семантическая ошибка: Недопустимый целочисленный литерал Строка: 1
poexali [now number x; x = 5 + "abc";]	Ошибка N314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 1
string operation fi()[return 5;] poexali [newline;]	Ошибка N315: Семантическая ошибка: Тип функции и возвращаемого значения не совпадают Строка: 1
poexali [now string x; x = "abc" + "d";]	Ошибка N316: Семантическая ошибка: Недопустимое строковое выражение справа от знака '=' Строка: 1
poexali [condition: "string"& 6# istruel[zapishi "string";]]	Ошибка N317: Семантическая ошибка: Неверное условное выражение Строка: 1
poexali [now number a =5; a = a/0; zapishi a;]	Ошибка N318: Семантическая ошибка: Деление на ноль Строка: 4

Заключение

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

1. Сформулирована спецификация языка SVY-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 SVY-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 SVY-2020

```
procedure operation stand (string poexali
ada, string bred)
[
    now number forlen;
    forlen = lenght(ada);
    zapishi "LenghtA:";
    zapishi forlen;
    newline;
    now string str;
    str = concat(ada,bred);
    zapishi "concat:";
    zapishi str;
    newline;
    conclusion;
]
number operation min(number x, number y)
[
    now number result;
    condition: x < y #
    istrue [result = x;]
    isfalse [result = y;]#
    conclusion result;
]
[
    now number x = 9;
    now number y = -9;
    now string strokx = "Nebeute ";
    now string stroky =
    "pogaluista";
    now string strokz = "15December";
    now number chil;
    now number temp;
    zapishi "iz srtoki v
chislo:";
    chil = atoi(strokz);
    zapishi chil;
    newline;
    now number pre;
    pre = x - 40 ;
    zapishi pre;
    newline;
    now number result;
    result = x{1;
    zapishi "sdvig vlevo:";
    zapishi result;
    newline;
    temp = min (x,y);
```


zapishi temp;	xru = (xru + 3)*2;
newline;]#
now number xru = 5;	zapishi " ";
condition: xru < 82 #	zapishi xru;
cicle [newline;
zapishi xru;	stand(strokk,stroky);
zapishi " ";]

Приложение Б

Листинг 1 – Таблица идентификаторов контрольного примера

----- ТАБЛИЦА ИДЕНТИФИКАТОРОВ -----					

N	СТРОКА В ТЛ	ТИП ИДЕНТИФИКАТОРА		ИМЯ	ЗНАЧЕНИЕ (ПА-
РАМЕТРЫ)					
0	2	proc	function	stand	P0:STRING
P1:STRING					
1	5	string	parameter	standada	
2	8	string	parameter	standbred	
3	13	number	variable	standforlen	0
4	17	number	LIB FUNC	lenght	P0:STRING
5	23	string	literal	VALS1	[8]LenghtA:
6	32	string	variable	standstr	[0]
7	36	string	LIB FUNC	concat	P0:STRING
P1:STRING					
8	44	string	literal	VALS2	[7]concat:
9	56	number	function	min	P0:NUMBER
P1:NUMBER					
10	59	number	parameter	minx	
11	62	number	parameter	miny	
12	67	number	variable	minresult	0
13	97	number	variable	poexalix	0
14	99	number	literal	VALS3	9
15	103	number	variable	poexaliy	0
16	105	number	literal	VALS4	-9
17	109	string	variable	poexalistrokx	[0]
18	111	string	literal	VALS5	[8]Nebeute
19	115	string	variable	poexalistroky	[0]
20	117	string	literal	VALS6	[10]pogaluista
21	121	string	variable	poexalistrokz	[0]
22	123	string	literal	VALS7	[10]15December
23	127	number	variable	poexalichil	0
24	131	number	variable	poexalitemp	0
25	134	string	literal	VALS8	[19]iz srtoki
v chislo:					
26	138	number	LIB FUNC	atoi	P0:STRING
27	150	number	variable	poexalipre	0
28	156	number	literal	VALS9	40
29	165	number	variable	poexaliresult	0
30	171	number	literal	VALS10	1
31	174	string	literal	VALS11	[12]sdvig
vlevo:					
32	197	number	variable	poexalixru	0
33	199	number	literal	VALS12	5
34	204	number	literal	VALS13	82

35	212	string	literal	VALS14	[1]
36	219	number	literal	VALS15	3
37	222	number	literal	VALS16	2

Листинг 2 – Таблица лексем после контрольного примера

```

1 | pfi[0](ti[1],ti[2])
2 | [
3 | nti[3];
4 | i[3]=i[4](i[1]);
5 | ol[5];
6 | oi[3];
7 | ^;
8 | nti[6];
9 | i[6]=i[7](i[1],i[2]);
10 | ol[8];
11 | oi[6];
12 | ^;
14 | e;
15 | ]
16 | tfi[9](ti[10],ti[11])
17 | [
18 | nti[12];
19 | ?i[10]<i[11]#
20 | w[i[12]=i[10];]
21 | r[i[12]=i[11];]#
22 | ei[12];
23 | ]
24 | m
25 | [
26 | nti[13]=l[14];
27 | nti[15]=l[16];
28 | nti[17]=l[18];
29 | nti[19]=l[20];
30 | nti[21]=l[22];
31 | nti[23];
32 | nti[24];
33 | ol[25];
34 | i[23]=i[26](i[21]);
35 | oi[23];
36 | ^;
37 | nti[27];
38 | i[27]=i[13]-l[28];
39 | oi[27];
40 | ^;
41 | nti[29];
42 | i[29]=i[13]{l[30];
43 | ol[31];

```

```

44 | oi[29];
45 | ^;
46 | i[24]=i[9](i[13],i[15]);
47 | oi[24];
48 | ^;
49 | nti[32]=l[33];
50 | ?i[32]<l[34]#
51 | c[
52 | oi[32];
53 | ol[35];
54 | i[32]=(i[32]+l[36])*l[37];
55 | ]#
56 | ol[35];
57 | oi[32];
58 | ^;
59 | i[0](i[17],i[19]);
60 | ]

```

Приложение В

Листинг 1 – Грамматика языка SVY-2020

```
Greibach greibach(NS('S'), TS('$'), 16,  
  
    Rule(NS('S'), GRB_ERROR_SERIES, 3,  
        Rule::Chain(6, TS('t'), TS('f'), TS('i'), NS('P'), NS('T'),  
NS('S'))),  
        Rule::Chain(6, TS('p'), TS('f'), TS('i'), NS('P'), NS('G'),  
NS('S'))),  
        Rule::Chain(4, TS('m'), TS('['), NS('K'), TS(']'))  
    ),  
  
    Rule(NS('T'), GRB_ERROR_SERIES + 2, 2,  
        Rule::Chain(5, TS('['), TS('e'), NS('V'), TS(';'), TS(']'))),  
        Rule::Chain(6, TS('['), NS('K'), TS('e'), NS('V'), TS(';'),  
TS(']'))  
    ),  
  
    Rule(NS('P'), GRB_ERROR_SERIES + 1, 2,  
        Rule::Chain(3, TS('('), NS('E'), TS(')'))),  
        Rule::Chain(2, TS('('), TS(')'))  
    ),  
  
    Rule(NS('G'), GRB_ERROR_SERIES + 3, 2,  
        Rule::Chain(4, TS('['), TS('e'), TS(';'), TS(']'))),  
        Rule::Chain(5, TS('['), NS('K'), TS('e'), TS(';'), TS(']'))  
    ),  
  
    Rule(NS('F'), GRB_ERROR_SERIES + 5, 2,  
        Rule::Chain(3, TS('('), NS('N'), TS(')'))),  
        Rule::Chain(2, TS('('), TS(')'))  
    ),  
  
    Rule(NS('E'), GRB_ERROR_SERIES + 4, 2,  
  
        Rule::Chain(4, TS('t'), TS('i'), TS(', '), NS('E'))),  
        Rule::Chain(2, TS('t'), TS('i'))  
    ),  
  
    Rule(NS('N'), GRB_ERROR_SERIES + 6, 4,  
  
        Rule::Chain(1, TS('i')),  
        Rule::Chain(1, TS('l')),  
        Rule::Chain(3, TS('i'), TS(', '), NS('N')),  
        Rule::Chain(3, TS('l'), TS(', '), NS('N'))  
    ),  
  
    Rule(NS('R'), GRB_ERROR_SERIES + 7, 5,  
        Rule::Chain(3, TS('r'), NS('Y'), TS('#')),  
        Rule::Chain(3, TS('w'), NS('Y'), TS('#')),
```

```

        Rule::Chain(3, TS('c'), NS('Y'), TS('#')),
        Rule::Chain(5, TS('r'), NS('Y'), TS('w'), NS('Y'), TS('#')),
        Rule::Chain(5, TS('w'), NS('Y'), TS('r'), NS('Y'), TS('#'))
    ),

    Rule(NS('Y'), GRB_ERROR_SERIES + 8, 1,

        Rule::Chain(3, TS('[', NS('X'), TS(']'))
    ),

    Rule(NS('Z'), GRB_ERROR_SERIES + 9, 3,

        Rule::Chain(3, TS('i'), NS('L'), TS('i')),
        Rule::Chain(3, TS('i'), NS('L'), TS('l')),
        Rule::Chain(3, TS('l'), NS('L'), TS('i'))
    ),

    Rule(NS('L'), GRB_ERROR_SERIES + 10, 4,

        Rule::Chain(1, TS('<')),
        Rule::Chain(1, TS('>')),
        Rule::Chain(1, TS('&')),
        Rule::Chain(1, TS('!'))
    ),

    Rule(NS('A'), GRB_ERROR_SERIES + 11, 6,
        Rule::Chain(1, TS('+')),
        Rule::Chain(1, TS('-')),
        Rule::Chain(1, TS('*')),
        Rule::Chain(1, TS('/')),
        Rule::Chain(1, TS('}')),
        Rule::Chain(1, TS('{'))
    ),

    Rule(NS('V'), GRB_ERROR_SERIES + 12, 3,
        Rule::Chain(1, TS('l')),
        Rule::Chain(1, TS('i')),
        Rule::Chain(1, TS('q'))
    ),

    Rule(NS('W'), GRB_ERROR_SERIES + 13, 8,
        Rule::Chain(1, TS('i')),
        Rule::Chain(1, TS('l')),
        Rule::Chain(3, TS('(', NS('W'), TS(')')),
        Rule::Chain(5, TS('(', NS('W'), TS(')'), NS('A'), NS('W')),
        Rule::Chain(2, TS('i'), NS('F')),
        Rule::Chain(3, TS('i'), NS('A'), NS('W')),
        Rule::Chain(3, TS('l'), NS('A'), NS('W')),
        Rule::Chain(4, TS('i'), NS('F'), NS('A'), NS('W'))
    ),

    Rule(NS('K'), GRB_ERROR_SERIES + 14, 14,

```

```

        Rule::Chain(7, TS('n'), TS('t'), TS('i'), TS('='), NS('V'),
TS(';'), NS('K')),
        Rule::Chain(5, TS('n'), TS('t'), TS('i'), TS(';'), NS('K')),
        Rule::Chain(5, TS('i'), TS('='), NS('W'), TS(';'), NS('K')),

        Rule::Chain(4, TS('o'), NS('V'), TS(';'), NS('K')),
        Rule::Chain(3, TS('^'), TS(';'), NS('K')),
        Rule::Chain(5, TS('?'), NS('Z'), TS('#'), NS('R'), NS('K')),
        Rule::Chain(4, TS('i'), NS('F'), TS(';'), NS('K')),

        Rule::Chain(6, TS('n'), TS('t'), TS('i'), TS('='), NS('V'),
TS(';')),
        Rule::Chain(4, TS('i'), TS('='), NS('W'), TS(';')),
        Rule::Chain(4, TS('n'), TS('t'), TS('i'), TS(';')),
        Rule::Chain(3, TS('o'), NS('V'), TS(';')),
        Rule::Chain(2, TS('^'), TS(';')),
        Rule::Chain(4, TS('?'), NS('Z'), TS('#'), NS('R')),
        Rule::Chain(3, TS('i'), NS('F'), TS(';'))

    ),

    Rule(NS('X'), GRB_ERROR_SERIES + 15, 8,
        Rule::Chain(5, TS('i'), TS('='), NS('W'), TS(';'), NS('X')),

        Rule::Chain(4, TS('o'), NS('V'), TS(';'), NS('X')),

        Rule::Chain(3, TS('^'), TS(';'), NS('X')),

        Rule::Chain(4, TS('i'), NS('F'), TS(';'), NS('X')),

        Rule::Chain(4, TS('i'), TS('='), NS('W'), TS(';')),
        Rule::Chain(3, TS('o'), NS('V'), TS(';')),
        Rule::Chain(2, TS('^'), TS(';')),
        Rule::Chain(3, TS('i'), NS('F'), TS(';'))

    )
);

```


Листинг 2 Структура магазинного автомата

```
namespace MFST
{
    struct MfstState
    {
        short lenta_position;
        short nrule;
        short nrulechain;
        MFSTSTACK st;
        MfstState();
        MfstState(
            short pposition,
            MFSTSTACK pst,
            short pnrulechain
        );
        MfstState(
            short pposition,
            MFSTSTACK pst,
            short pnrule,
            short pnrulechain
        );
    };

    struct Mfst
    {
        enum RC_STEP
        {
            NS_OK,
            NS_NORULE,
            NS_NORULECHAIN,
            NS_ERROR,
            TS_OK,
            TS_NOK,
            LENTA_END,
            SURPRISE
        };
        struct MfstDiagnosis //диагностика
        {
            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 grebach;
    Lexer::LEX lex;
    MFSTSTACK st;
    use_container<std::stack<MfstState>> storestate;
    Mfst();
    Mfst(
        Lexer::LEX plex,
        GRB::Greibach pgrebach
    );
    char* getCSt(char* buf);
    char* getCLenta(char* buf, short pos, short n = 25);
    char* getDiagnosis(short n, char* buf);
    bool savestate(const Log::LOG& log);
    bool reststate(const Log::LOG& log);
    bool push_chain(
        GRB::Rule::Chain chain
    );
    RC_STEP step(const Log::LOG& log);
    bool start(const Log::LOG& log);
    bool savediagnosis(
        RC_STEP pprc_step
    );
    void printrules(const Log::LOG& log);
    struct Deduction
    {
        short size;
        short* nrules;
        short* nrulechains;
        Deduction() { size = 0; nrules = 0; nrulechains = 0; };
    } deduction;
    bool savededuction();
};
};
};

```

Листинг 3 Структура грамматики Грейбах

```

struct Greibach //грамматика Грейбах
{
    short size; //количество правил
    GRBALPHABET startN; //стартовый символ
    GRBALPHABET stbottomT; //дно стека
    Rule* rules; //множество правил
    Greibach() { short size = 0; startN = 0; stbottomT = 0; rules = 0; };
};

```

```

Greibach(
GRBALPHABET pstartN,           //стартовый символ
GRBALPHABET pstbootomT,       //дно стека
short psize,                   //количество правил
Rule r, ...                    //правила
);
short getRule(                 //получить правило, возвращается номер
мер правила или -1
GRBALPHABET pnn,              //левый символ правила
Rule& prule                   //возвращаемое правило грамматики
);
Rule getRule(short n);         //получить правило по номеру
};

```

Листинг 4 Разбор исходного кода синтаксическим анализатором

Шаг	Правило	Входная лента	Стек
0	:S->pfiPGS	pfi(ti,ti)[nti;i=i(i);ol;	S\$
1	: SAVESTATE:	1	
1	:	pfi(ti,ti)[nti;i=i(i);ol;	pfiPGS\$
2	:	fi(ti,ti)[nti;i=i(i);ol;o	fiPGS\$
3	:	i(ti,ti)[nti;i=i(i);ol;oi	iPGS\$
4	:	(ti,ti)[nti;i=i(i);ol;oi;	PGS\$
5	:P->(E)	(ti,ti)[nti;i=i(i);ol;oi;	PGS\$
6	: SAVESTATE:	2	
6	:	(ti,ti)[nti;i=i(i);ol;oi;	(E)GS\$
7	:	ti,ti)[nti;i=i(i);ol;oi;^	E)GS\$
8	:E->ti,E	ti,ti)[nti;i=i(i);ol;oi;^	E)GS\$
9	: SAVESTATE:	3	
9	:	ti,ti)[nti;i=i(i);ol;oi;^	ti,E)GS\$
10	:	i,ti)[nti;i=i(i);ol;oi;^;	i,E)GS\$
11	:	,ti)[nti;i=i(i);ol;oi;^;n	,E)GS\$
12	:	ti)[nti;i=i(i);ol;oi;^;nt	E)GS\$
13	:E->ti,E	ti)[nti;i=i(i);ol;oi;^;nt	E)GS\$
14	: SAVESTATE:	4	
14	:	ti)[nti;i=i(i);ol;oi;^;nt	ti,E)GS\$
15	:	i)[nti;i=i(i);ol;oi;^;nti	i,E)GS\$
16	:)[nti;i=i(i);ol;oi;^;nti;	,E)GS\$
17	: TS_NOK/NS_NORULECHAIN		
17	: RESSTATE		
17	:	ti)[nti;i=i(i);ol;oi;^;nt	E)GS\$
18	:E->ti	ti)[nti;i=i(i);ol;oi;^;nt	E)GS\$
19	: SAVESTATE:	4	
19	:	ti)[nti;i=i(i);ol;oi;^;nt	ti)GS\$
20	:	i)[nti;i=i(i);ol;oi;^;nti	i)GS\$
21	:)[nti;i=i(i);ol;oi;^;nti;)GS\$
22	:	[nti;i=i(i);ol;oi;^;nti;i	GS\$
23	:G->[e;]	[nti;i=i(i);ol;oi;^;nti;i	GS\$
24	: SAVESTATE:	5	

```

24 : [nti;i=i(i);ol;oi;^;nti;i [e;]S$
25 : nti;i=i(i);ol;oi;^;nti;i= e;]S$
26 : TS_NOK/NS_NORULECHAIN
26 : RESSTATE
26 : [nti;i=i(i);ol;oi;^;nti;i GS$
27 :G->[Ke;] [nti;i=i(i);ol;oi;^;nti;i GS$
28 : SAVESTATE: 5
28 : [nti;i=i(i);ol;oi;^;nti;i [Ke;]S$
29 : nti;i=i(i);ol;oi;^;nti;i= Ke;]S$
30 :K->nti=V;K nti;i=i(i);ol;oi;^;nti;i= Ke;]S$
31 : SAVESTATE: 6
31 : nti;i=i(i);ol;oi;^;nti;i= nti=V;Ke;]S$
32 : ti;i=i(i);ol;oi;^;nti;i=i ti=V;Ke;]S$
33 : i;i=i(i);ol;oi;^;nti;i=i( i=V;Ke;]S$
34 : ;i=i(i);ol;oi;^;nti;i=i(i =V;Ke;]S$
35 : TS_NOK/NS_NORULECHAIN
35 : RESSTATE
35 : nti;i=i(i);ol;oi;^;nti;i= Ke;]S$
36 :K->nti;K nti;i=i(i);ol;oi;^;nti;i= Ke;]S$
37 : SAVESTATE: 6
37 : nti;i=i(i);ol;oi;^;nti;i= nti;Ke;]S$
38 : ti;i=i(i);ol;oi;^;nti;i=i ti;Ke;]S$
39 : i;i=i(i);ol;oi;^;nti;i=i( i;Ke;]S$
40 : ;i=i(i);ol;oi;^;nti;i=i(i ;Ke;]S$
41 : i=i(i);ol;oi;^;nti;i=i(i, Ke;]S$
42 :K->i=W;K i=i(i);ol;oi;^;nti;i=i(i, Ke;]S$

```

Листинг 4 (прод.) Разбор исходного кода синтаксическим анализатором

```

1068: i,i);] N);]$
1069:N->i,N i,i);] N);]$
1070: SAVESTATE: 117
1070: i,i);] i,N);]$
1071: ,i);] ,N);]$
1072: i);] N);]$
1073:N->i i);] N);]$
1074: SAVESTATE: 118
1074: i);] i);]$
1075: );] );]$
1076: ;] ;]$
1077: ] ]$
1078: $
1079: LENTA_END
1080: ----->LENTA_END

```

Приложение Г

Листинг 1 Программная реализация механизма преобразования в ПОЛИЗ

```
bool __cdecl StartPolishNat(IT::IdTable& idtable, Log::LOG& log, int
lextable_pos, ltvec& v)
{
    vector < LT::Entry > result;
    stack < LT::Entry > s;
    bool ignore = false;

    for (unsigned i = 0; i < v.size(); i++)
    {
        if (ignore)
        {
            result.push_back(v[i]);
            if (v[i].lexema == LEX_RIGHTSK)
                ignore = false;
            continue;
        }
        int priority = PriorityOperation(v[i]);

        if (v[i].lexema == LEX_LEFTSK || v[i].lexema == LEX_RIGHTSK
|| v[i].lexema == LEX_PL || v[i].lexema == LEX_MINUS || v[i].lexema ==
LEX_STAR || v[i].lexema == LEX_DIRSLASH || v[i].lexema == LEX_LEFT ||
v[i].lexema == LEX_RIGHT)
        {
            if (s.empty() || v[i].lexema == LEX_LEFTSK)
            {
                s.push(v[i]);
                continue;
            }

            if (v[i].lexema == LEX_RIGHTSK)
            {
                while (!s.empty() && s.top().lexema !=
LEX_LEFTSK)
                {
                    result.push_back(s.top());
                    s.pop();
                }
                if (!s.empty() && s.top().lexema == LEX_LEFTSK)
                    s.pop();
                continue;
            }
            while (!s.empty() && PriorityOperation(s.top()) >=
priority)
            {
                result.push_back(s.top());
                s.pop();
            }
        }
    }
}
```

```

        }
        s.push(v[i]);
    }

    if (v[i].lexema == LEX_LITERAL || v[i].lexema == LEX_ID)
    {
        if (idtable.table[v[i].idxTI].idtype == IT::IDTYPE::F
|| idtable.table[v[i].idxTI].idtype == IT::IDTYPE::S)
            ignore = true;
        result.push_back(v[i]);
    }
    if (v[i].lexema != LEX_LEFTSK & v[i].lexema != LEX_RIGHTSK
& v[i].lexema != LEX_PL & v[i].lexema != LEX_MINUS & v[i].lexema != LEX_STAR
& v[i].lexema != LEX_DIRSLASH & v[i].lexema != LEX_ID & v[i].lexema !=
LEX_LITERAL & v[i].lexema != LEX_LEFT & v[i].lexema != LEX_RIGHT)
    {
        Log::writeError(log.stream, Error::GetError(1));
        return false;
    }
}

while (!s.empty())
{
    result.push_back(s.top());
    s.pop();
}
v = result;
return true;
}

```

Листинг 2 Таблица идентификаторов после преобразования выражений в ПОЛИЗ

N		СТРОКА В ТЛ	ТИП ИДЕНТИФИКАТОРА		ИМЯ	ЗНАЧЕНИЕ
(ПАРАМЕТРЫ)						
0	2		proc	function	stand	P0:STRING
P1:STRING						
1	5		string	parameter	standada	
2	8		string	parameter	standbred	
3	13		number	variable	standforlen	0
4	17		number	LIB FUNC	lenght	P0:STRING
5	23		string	literal	VALS1	[8]LenghtA:
6	32		string	variable	standstr	[0]

7		36		string	LIB FUNC		concat		P0:STRING	
P1:STRING										
8		44		string	literal		VALS2		[7]concat:	
9		56		number	function		min		P0:NUMBER	
P1:NUMBER										
10		59		number	parameter		minx			
11		62		number	parameter		miny			
12		67		number	variable		minresult		0	
13		97		number	variable		poexalix		0	
14		99		number	literal		VALS3		9	
15		103		number	variable		poexaliy		0	
16		105		number	literal		VALS4		-9	
17		109		string	variable		poexalistrokx		[0]	
18		111		string	literal		VALS5		[8]Nebeute	
19		115		string	variable		poexalistroky		[0]	
20		117		string	literal		VALS6		[10]pogaluista	
21		121		string	variable		poexalistrokz		[0]	
22		123		string	literal		VALS7		[10]15December	
23		127		number	variable		poexalichil		0	
24		131		number	variable		poexalitemp		0	
25		134		string	literal		VALS8		[19]iz srtoki v	
chislo:										
26		138		number	LIB FUNC		atoi		P0:STRING	
27		150		number	variable		poexalipre		0	
28		155		number	literal		VALS9		40	
29		165		number	variable		poexaliresult		0	
30		170		number	literal		VALS10		1	
31		174		string	literal		VALS11		[12]sdvig vlevo:	
32		197		number	variable		poexalixru		0	

33		199		number	literal		VALS12		5
34		204		number	literal		VALS13		82
35		212		string	literal		VALS14		[1]
36		217		number	literal		VALS15		3
37		219		number	literal		VALS16		2

Листинг 3 Таблица лексем после преобразования выражений в ПОЛИЗ

N	ЛЕКСЕМА	СТРОКА	ИНДЕКС В ТИ	133	o	33	
				134	l	33	25
0	p	1		135	;	33	
1	f	1		136	i	34	23
2	i	1	0	137	=	34	
3	(1		138	i	34	26
4	t	1		139	(34	
5	i	1	1	140	i	34	21
6	,	1		141)	34	
7	t	1		142	;	34	
8	i	1	2	143	o	35	
9)	1		144	i	35	23
10	[2		145	;	35	
11	n	3		146	^	36	
12	t	3		147	;	36	
13	i	3	3	148	n	37	
14	;	3		149	t	37	
15	i	4	3	150	i	37	27
16	=	4		151	;	37	
17	i	4	4	152	i	38	27
18	(4		153	=	38	
19	i	4	1	154	i	38	13
20)	4		155	l	38	28
21	;	4		156	-	38	
22	o	5		157	;	38	
23	l	5	5	158	o	39	
24	;	5		159	i	39	27
25	o	6		160	;	39	
26	i	6	3	161	^	40	
27	;	6		162	;	40	
28	^	7		163	n	41	
29	;	7		164	t	41	
30	n	8		165	i	41	29
31	t	8		166	;	41	
32	i	8	6	167	i	42	29
33	;	8		168	=	42	
34	i	9	6	169	i	42	13
35	=	9		170	l	42	30
36	i	9	7	171	{	42	

37	(9		172	;	42	
38	i	9	1	173	o	43	
39	,	9		174	l	43	31
40	i	9	2	175	;	43	
41)	9		176	o	44	
42	;	9		177	i	44	29
43	o	10		178	;	44	
44	l	10	8	179	^	45	
45	;	10		180	;	45	
46	o	11		181	i	46	24
47	i	11	6	182	=	46	
48	;	11		183	i	46	9
49	^	12		184	(46	
50	;	12		185	i	46	13
51	e	14		186	,	46	
52	;	14		187	i	46	15
53]	15		188)	46	
54	t	16		189	;	46	
55	f	16		190	o	47	
56	i	16	9	191	i	47	24
57	(16		192	;	47	
58	t	16		193	^	48	
59	i	16	10	194	;	48	
60	,	16		195	n	49	
61	t	16		196	t	49	
62	i	16	11	197	i	49	32
63)	16		198	=	49	
64	[17		199	l	49	33
65	n	18		200	;	49	
66	t	18		201	?	50	
67	i	18	12	202	i	50	32
68	;	18		203	<	50	
69	?	19		204	l	50	34
70	i	19	10	205	#	50	
71	<	19		206	c	51	
72	i	19	11	207	[51	
73	#	19		208	o	52	
74	w	20		209	i	52	32
75	[20		210	;	52	
76	i	20	12	211	o	53	
77	=	20		212	l	53	35
78	i	20	10	213	;	53	
79	;	20		214	i	54	32
80]	20		215	=	54	
81	r	21		216	i	54	32
82	[21		217	l	54	36
83	i	21	12	218	+	54	
84	=	21		219	l	54	37
85	i	21	11	220	*	54	
86	;	21		221	;	54	
87]	21		222]	55	
88	#	21		223	#	55	

89	e	22		224	o	56	
90	i	22	12	225	l	56	35
91	;	22		226	;	56	
92]	23		227	o	57	
93	m	24		228	i	57	32
94	[25		229	;	57	
95	n	26		230	^	58	
96	t	26		231	;	58	
97	i	26	13	232	i	59	0
98	=	26		233	(59	
99	l	26	14	234	i	59	17
100	;	26		235	,	59	
101	n	27		236	i	59	19
102	t	27		237)	59	
103	i	27	15	238	;	59	
104	=	27		239]	60	
105	l	27	16				
106	;	27					
107	n	28					
108	t	28					
109	i	28	17				
110	=	28					
111	l	28	18				
112	;	28					
113	n	29					
114	t	29					
115	i	29	19				
116	=	29					
117	l	29	20				
118	;	29					
119	n	30					
120	t	30					
121	i	30	21				
122	=	30					
123	l	30	22				
124	;	30					
125	n	31					
126	t	31					
127	i	31	23				
128	;	31					
129	n	32					
130	t	32					
131	i	32	24				
132	;	32					

Листинг 4 Соответствие лексем исходному коду программы

```

1  | pfi[0](ti[1],ti[2])
2  | [
3  | nti[3];
4  | i[3]=i[4](i[1]);
5  | ol[5];
6  | oi[3];
7  | ^;
8  | nti[6];
9  | i[6]=i[7](i[1],i[2]);
10 | ol[8];
11 | oi[6];
12 | ^;
14 | e;
15 | ]
16 | tfi[9](ti[10],ti[11])
17 | [
18 | nti[12];
19 | ?i[10]<i[11]#
20 | w[i[12]=i[10];]
21 | r[i[12]=i[11];]#
22 | ei[12];
23 | ]
24 | m
25 | [
26 | nti[13]=l[14];
27 | nti[15]=l[16];
28 | nti[17]=l[18];
29 | nti[19]=l[20];
30 | nti[21]=l[22];
31 | nti[23];
32 | nti[24];
33 | ol[25];
34 | i[23]=i[26](i[21]);
35 | oi[23];
36 | ^;
37 | nti[27];
38 | i[27]=i[13]l[28]-;
39 | oi[27];
40 | ^;
41 | nti[29];
42 | i[29]=i[13]l[30]{;
43 | ol[31];
44 | oi[29];
45 | ^;
46 | i[24]=i[9](i[13],i[15]);
47 | oi[24];
48 | ^;
49 | nti[32]=l[33];
50 | ?i[32]<l[34]#
51 | c[

```

```
52 | oi[32];
53 | ol[35];
54 | i[32]=i[32]l[36]+l[37]*;
55 | ]#
56 | ol[35];
57 | oi[32];
58 | ^;
59 | i[0](i[17],i[19]);
60 | ]
```

Приложение Д

Листинг 1 Результат генерации кода контрольного примера в Ассемблере

```

.586
.model flat, stdcall
includelib libucrt.lib
includelib kernel32.lib
includelib "D:\Kursach\SVY2020\Gener-
ation\Debug\GenLib.lib
ExitProcess PROTO:DWORD
.stack 4096

outnum PROTO : DWORD

outstr PROTO : DWORD

concat PROTO : DWORD, : DWORD

lenght PROTO : DWORD

atoi PROTO : DWORD

.const
    newline byte 13, 10, 0
    VALS1 byte 'LenghtA:',
0
    VALS2 byte 'concat:', 0
    VALS3 sdword 9
    VALS4 sdword -9
    VALS5 byte 'Nebeute ',
0
    VALS6 byte
'pogaluista', 0
    VALS7 byte '15Decem-
ber', 0
    VALS8 byte 'iz srtoki v
chislo:', 0
    VALS9 sdword 40
    VALS10 sdword 1
    VALS11 byte 'sdvig
vlevo:', 0
    VALS12 sdword 5
    VALS13 sdword 82
    VALS14 byte ' ', 0
    VALS15 sdword 3
    VALS16 sdword 2

.data
    temp sdword ?
    buffer byte 256 dup(0)

standforlen sdword 0
standstr dword ?
minresult sdword 0
poexalix sdword 0
poexaliy sdword 0
poexalistrokx dword ?
poexalistroky dword ?
poexalistrokz dword ?
poexalichil sdword 0
poexalitemp sdword 0
poexalipre sdword 0
poexalireult sdword 0
poexalixru sdword 0

.code

;~~~~~ stand ~~~~~
stand PROC,
    standada : dword, standbred :
dword
    push ebx
    push edx

    push standada
    call lenght
    push eax

    pop ebx
    mov standforlen, ebx

    push offset VALS1
    call outstr

    push standforlen
    call outnum

    push offset newline
    call outstr

    push standbred
    push standada
    call concat
    mov standstr, eax

```

push offset VALS2	;~~~~~ POEXALI ~~~~~
call outstr	main PROC
	push VALS3
push standstr	pop ebx
call outstr	mov poexalix, ebx
push offset newline	push VALS4
call outstr	
	pop ebx
	mov poexaliy, ebx
pop edx	
pop ebx	mov poexalistrokx, offset VALS5
ret	mov poexalistroky, offset VALS6
stand ENDP	mov poexalistrokz, offset VALS7
;~~~~~	
	push offset VALS8
	call outstr
;~~~~~ min ~~~~~	
min PROC,	
minx : sdword, miny : sdword	push poexalistrokz
push ebx	call atoi
push edx	push eax
mov edx, minx	pop ebx
cmp edx, miny	mov poexalichil, ebx
j1 right1	
jge wrong1	push poexalichil
right1:	call outnum
push minx	
pop ebx	push offset newline
mov minresult, ebx	call outstr
jmp next1	push poexalix
wrong1:	push VALS9
push miny	pop ebx
	pop eax
	sub eax, ebx
pop ebx	push eax
mov minresult, ebx	
	pop ebx
next1:	mov poexalipre, ebx
pop edx	
pop ebx	push poexalipre
mov eax, minresult	call outnum
ret	
min ENDP	push offset newline
;~~~~~	call outstr
	push poexalix

```

push VALS10
pop ebx
pop eax
mov cl, bl
shl eax, cl
push eax

pop ebx
mov poexaliresult, ebx

```

```

push offset VALS11
call outstr

```

```

push poexaliresult
call outnum

```

```

push offset newline
call outstr

```

```

push poexaliy
push poexalix
call min
push eax

```

```

pop ebx
mov poexalitemp, ebx

```

```

push poexalitemp
call outnum

```

```

push offset newline
call outstr

```

```

push VALS12

```

```

pop ebx
mov poexalixru, ebx

```

```

mov edx, poexalixru
cmp edx, VALS13

```

```

jl cycle2
jmp cyclenext2
cycle2:

```

```

push poexalixru
call outnum

```

```

push offset VALS14
call outstr

```

```

push poexalixru
push VALS15
pop ebx
pop eax
add eax, ebx
push eax
push VALS16
pop ebx
pop eax
imul eax, ebx
push eax

```

```

pop ebx
mov poexalixru, ebx

```

```

mov edx, poexalixru
cmp edx, VALS13

```

```

jl cycle2
cyclenext2:

```

```

push offset VALS14
call outstr

```

```

push poexalixru
call outnum

```

```

push offset newline
call outstr

```

```

push poexalistroky
push poexalistrokx
call stand

```

```

push 0
call ExitProcess
main ENDP
end main

```