МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет иформационных технологий
Кафедра программной инженерии
Специальность 6-05-0612-01 Программная инженерия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

«Разработка компилятора PYA-2024»

Выполнил студент Павлович Ян Андреевич
(Ф.И.О.)

Руководитель проекта асс. Ромыш Александра Сергеевна
(учен. степень, звание, должность, подпись, Ф.И.О.)

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

Консультанты асс. Ромыш Александра Сергеевна
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Ромыш Александра Сергеевна
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2024

Содержание

Введение

Задачей данного курсового проекта была поставлена разработка транслятора своего языка программирования PYA-2024. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над числами.

Главной задачей транслятора заключается в том, чтобы сделать исходный код на данном языке программирования понятной компьютеру. Для решения этой задачи был выбран способ трансляции исходного кода моего языка программирования в исходный код на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений с помощью обратной польской нотации;

– генерация кода на язык ассемблера;

– тестирование транслятора;

Способы решения каждой задачи будут описаны в соответствующих главах курсового проекта.

1. Спецификация языка программирования

1.1 Характеристика языка программирования

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

1.2 Определение алфавита языка программирования

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

1.3 Применяемые сепараторы

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

Таблица 1.1 – Символы-сепараторы

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

Символы сепараторы включают в себя символы пробела, блоки функций или циклов, блоки параметров функций и тд.

1.4 Применяемые кодировки

Для написания программ на языке PYA-2024 используется кодировка Windows – 1251, представленная на рис.1.1.

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

Рисунок 1.1 Алфавит вводных символов

Согласно кодировке Windows-1251 был разработан алфавит языка PYA-2024.

1.5 Типы данных

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

Таблица 1.2 – Типы данных языка PYA-2024

Типы данных	Характеристика
Целочисленный тип данных **numb**	Является целочисленным типом данных. Этот тип данных занимает 4 байта. Предназначен для арифметических операций над числами. Инициализация по умолчанию: 0. Поддерживаемые операции: + (бинарный) – Оператор сложения; - (бинарный) – Оператор вычитания; * (бинарный) – Оператор умножения; / (бинарный) – Оператор деления; = (бинарный) – Оператор присваивания %(бинарный) – Остаток от деления В качестве операторов условия или условия цикла можно использовать следующие операторы: > (бинарный) – Оператор “больше”; < (бинарный) – Оператор “меньше” } (бинарный) – Оператор “больше либо равно” { (бинарный) – Оператор “меньше либо равно”
Строковый тип данных **stroke**	Фундаментальный тип данных. Используется для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 255. Инициализация по умолчанию: строка нулевой длины ". Операции над данными строкового типа: =(бинарный)оператор присваивания
Символьный тип данных **symbol**	Фундаментральный тип данных. Используется для работы с символом, занимающим 1 байт. Инициализация по умолчанию: символ нулевой длины “”. Операции над данными символьного типа: =(бинарный)оператор присваивания.
Логический тип данный **boolean**	Является логическим типом данных. Переменные данного типа могу принимать 2 значения:true или false. Операции над данными логического типа: |(бинарный) – Логическое или &(бинарный) – Логическое И

Эти типы данных предоставляют разработчикам инструменты для выполнения разнообразных операций, от арифметических до операций сравнения

и условий, что делает их ключевыми строительными блоками для создания разнообразных программных решений.

1.6 Преобразование типов данных

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

1.7 Идентификаторы

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

Правило составления идентификатора:

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

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

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

1.8 Литералы

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

Таблица 1.3 – Литералы

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

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

1.9 Объявление данных

Для объявления переменной используется ключевое слово **new,** после которого указывается тип данных и имя идентификатора.
Пример объявления числового типа данных с инициализации:
new numb x = 14;
Пример объявления строкового типа данных с инициализацией:
new stroke str = 'привет мир';
Пример объявления логического типа данные с инициализацией:
new boolean b = true;
new boolean c = false;
Пример объявления символьного типа данных с инициализацией:
new symbol s = “S”;

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

1.10 Инициализация данных

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

1.11 Инструкции языка

Инструкции языка PYA-2024 представлены в таблице 1.4.

Таблица 1.4 – Инструкции языка

Инструкция	Реализация
Объявление переменной	**New** <тип данных><идентификатор>;
Возврат значения из функции	**return** <идентификатор> | <литерал>;
Вывод данных	**write** <идентификатор> | <литерал>; **writeline** <идентификатор> | <литерал>;
Вызов функции	<идентификатор функции>(<список параметров>);
Присваивание	<идентификатор> = <выражение>;

Программа обычно представляет собой последовательность инструкций.

1.12 Операции языка

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

Таблица 1.5 – Операции языка PYA-2024

Тип оператора	Оператор
Арифметические	+ - сложение (приоритет 5) - - разность (приоритет 5) * - умножение (приоритет 4) / - деление (приоритет 4) % - остаток от деления (приоритет 4)
Логические	> - больше (приоритет 7) < - меньше (приоритет 7) ! – не равно (приоритет 8) } – больше или равно (приоритет 7) { - меньше или равно (приоритет 7) &-логическое и (приоритет 9) | - логическое или (приоритет 10)

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

1.13 Выражения и их вычисления

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

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

Перед генерацией кода каждое выражение приводится к записи в польской записи для удобства дальнейшего вычисления выражения на языке ассемблера. Преобразование выражений приведено в главе 5.

1.14 Конструкции языка

Программа на языке PYA-2024 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 PYA-2024 представлены в таблице 1.6.

Таблица 1.6 – Конструкции языка PYA-2024

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

За проверку реализации всех конструкций языка отвечает семантический анализатор.

1.15 Область видимости идентификаторов

Область видимости: сверху вниз. Переменные, объявленные в одной функции не доступны в другой. Все операции и объявления происходят внутри какого-либо блока или тела функции. Каждая переменная или параметр функции получают область видимости – название функции, в которой они находятся. Все идентификаторы являются локальными и обязаны быть объявленными внутри какой-либо функции. Глобальных переменных нет. Параметры видны только внутри функции, в которой объявлены.

1.16 Семантические проверки

В языке программирования PYA-2024 выполняются следующие семантические проверки:

1. Наличие функции main – точки входа в программу;

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

1.17 Распределение оперативной памяти на этапе выполнения

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

1.18 Стандартная библиотека и ее состав

В языке PYA-2024 присутствует стандартная библиотека, которая подключается автоматически на этапе трансляции исходного кода в язык ассемблера. Содержимое стандартной библиотеки представлено в таблице 1.8.

Таблица 1.8 – Стандартная библиотека языка PYA-2024

Функция	Описание
numb Rand(numb b)	Целочисленная функция, возвращает псевдослучайное число в определенном диапазоне b.
numb Strlen(stroke str)	Целочисленная функция, возвращает размер строки
numb Input(numb a)	Целочисленная функция, возвращает число, введенное пользователем.
numb Abs(numb b)	Целочисленная функция, возвращает число, введенное пользователем, по модулю.
numb Sqrt(numb s)	Целочисленная функция, возвращает корень числа, введенного пользователем.

Также статическая библиотека содержит функции вывода в поток, которые используются при генерации кода в Assembler.

1.19 Вывод и ввод данных

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

1.20 Точка входа

В языке PYA-2024 каждая программа должна содержать главную функцию (точку входа) **main**, с первой инструкции которой начнется последовательное выполнение команд программы. Должна иметься только одна точка входа main.

1.21 Препроцессор

Команды препроцессора в языке PYA-2024 отсутствуют.

1.22 Соглашения о вызове

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

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

1.23 Объектный код

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

1.24 Классификация сообщений транслятора

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Сообщения об ошибках имеют специфический постфикс, зависящий от этапа, на котором обнаружена ошибка. Список постфиксов приведен в таблице 1.9.

Таблица 1.9 – Список префиксов ошибок в языке PYA-2024

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

Перечень всех ошибок находится в приложениях Б, В и Г.

1.25 Контрольный пример

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

2. Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

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

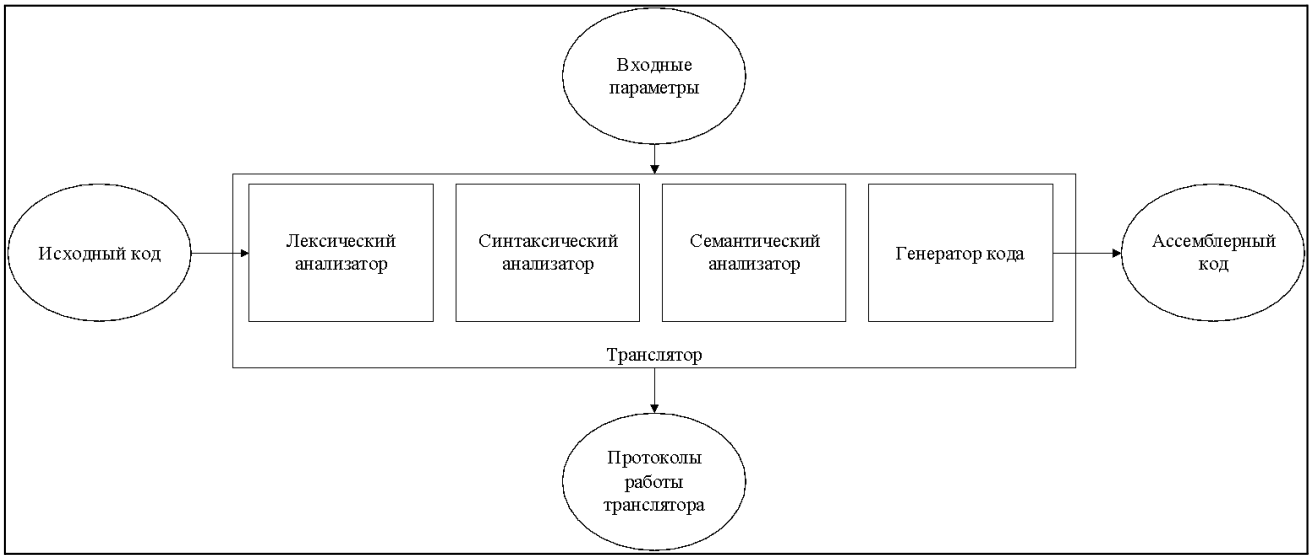

Рисунок 2.1 Структура транслятора языка программирования PYA-2024

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

Цели лексического анализатора:
- убрать все лишние пробелы;
- выполнить распознавание лексем;
- построить таблицу лексем и таблицу идентификаторов;
- при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

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

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

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

2.2 Перечень входных параметров транслятора

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

Таблица 2.1 – Входные параметры транслятора языка PYA-2024

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

-IT:<путь к IT-файлу>	Файл содержащий таблицу идентификаторов	Значение по умолчанию: <имя in-файла>.IT

Данные параметры указываются как аргументы команды в свойствах отладки и при запуске проекта через консоль разработчика.

2.3 Перечень протоколов, формируемых транслятором и их содержимое

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

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

Таблица 2.2 – Протоколы, формируемые транслятором языка PYA-2024

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

Файлы всех протоколов находятся в папке PYA-2024.

3. Разработка лексического анализатора

3.1 Структура лексического анализатора

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

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

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

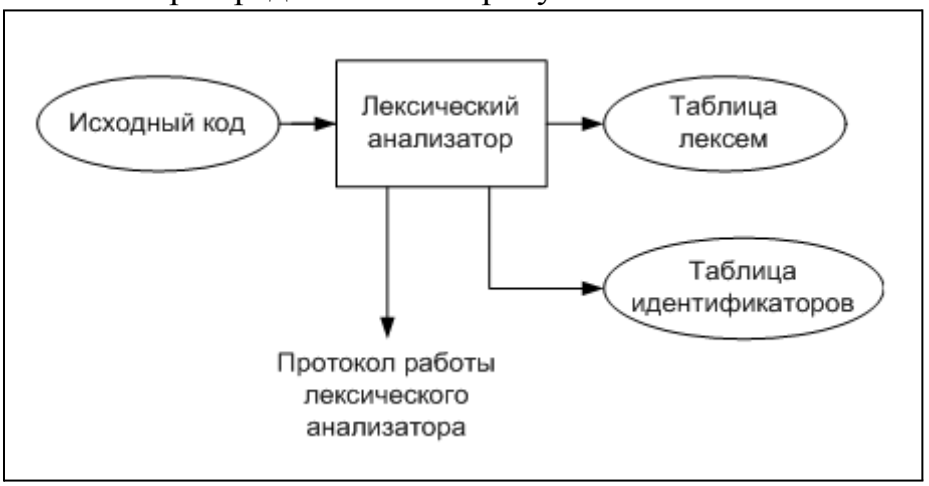

Рисунок 3.1 Структура лексического анализатора

Функции лексического анализатора:

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

3.2. Контроль входных символов

Таблица контроля входных символов представлена в приложении Б

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

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

3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

-посимвольно считываем файл с исходным кодом программы;

-в отличие от других символов-разделителей, не записываем пробелы и символы табуляции в таблицу лексем;

-продолжаем считывание файла с исходным кодом программы до встречи с лексемой, отличной от пробела или символа табуляции.

3.4 Перечень ключевых слов

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

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

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

Токен	Лексема	Пояснение
,	,	Разделение параметров функций.
[	[	Начало блока/тела функции.
]	]	Закрытие блока/тела функции.
(	(	Передача параметров в функцию,
)	)	Закрытие блока для передачи параметров, приоритет операций.
=	=	Знак присваивания.

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

Input	U	Указывает на стандартную функцию Input
Strlen	S	Указывает на стандартную функцию Strlen
correctly	c	Указывает на достоверность условного выражения
wrong	w	Указывает на недостоверность условного выражения
Abs	A	Указывает на стандартную функцию Abs
Sqrt	C	Указывает на стандартную функцию Sqrt

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

3.5 Основные структуры данных

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

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

3.6 Принцип обработки ошибок

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

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

3.7 Структура и перечень сообщений лексического анализа

Перечень сообщений представлен в приложении Б.

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

3.8 Параметры лексического анализатора

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

3.9 Алгоритм лексического анализа

Последовательность выполнения алгоритма работы лексического анализатора представлен ниже.

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

3.10 Контрольный пример

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

4. Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

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

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

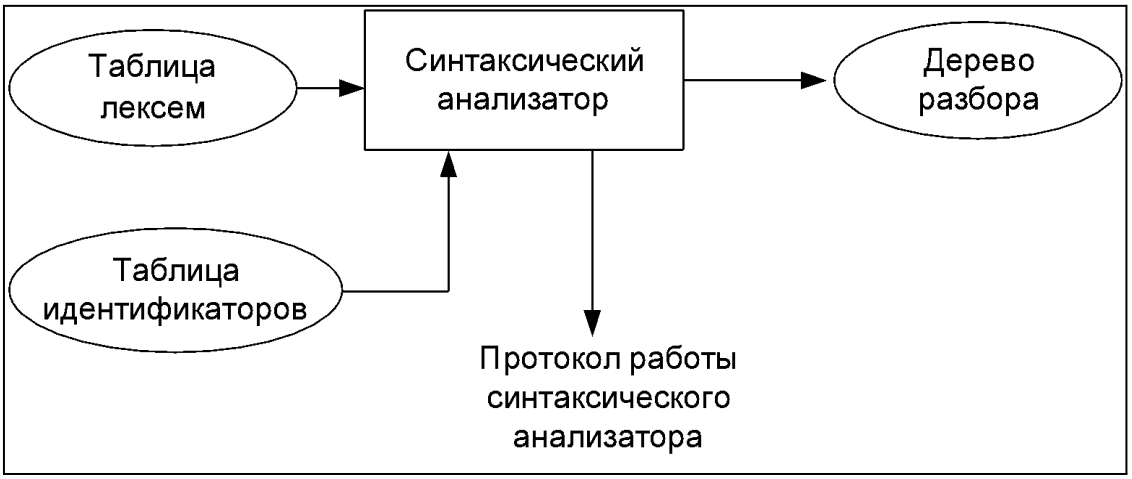

Рисунок 4.1 – Структура синтаксического анализатора

На рисунке 4.1 представлено описание структуры синтаксического анализатора языка.

4.2 Контекстно-свободная грамматика, описывающая синтаксис

В синтаксическом анализаторе транслятора языка PYA-2024 используется контекстно-свободная грамматика $G = \langle T, N, P, S \rangle$, где

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

1) $A \to a\alpha$, где $a \in T, \alpha \in (T \cup N) \cup \{\lambda\}$; (или

$\alpha \in (T \cup N)^*$, или $\alpha \in V^*$);

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

	N->i=E; N->pE; N->pE;N N->?KN N->?K N->rM; N->pi;rM	
Символ	Правила	Какие правила порождает
O	O –>| O –>& O –>> O –>< O –>} O –>{	Правила построения логических операторов
Q	Q–>(M) Q–>(M)VE	Правила передачи параметров в функцию статической библиотеки
L	L –>SQ L –>UQ L –>AQ L –>CQ	Правила вызова функций статической библиотеки
D	D –>tW	Правило инициализации переменной
M	M –>i M –>l	Правила вывода идентификатора/литерала
E	E –>i E –>l E –>i; E –>l; E –>iVE E –>lVE E –>iOE E –>lOE E–>SQ E –>UQ E –>RQ Ew –>i(G)	Правила построения выражений

Данные правила используются для построения дерева разбора контрольного примера, представленного в приложении А.

4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку

$$M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle .$$

Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2 – Описание компонентов магазинного автомата

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

Структура цепочки конечного автомата представлена в приложении В.

4.4 Основные структуры данных

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

4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата следующий:

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

4.6 Структура и перечень сообщений синтаксического анализатора

Сообщения генерируемые синтаксическим анализатором представлены в приложении В.

4.7. Параметры синтаксического анализатора и режимы его работы

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

4.8. Принцип обработки ошибок

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

4.9. Контрольный пример

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

5. Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.

Рисунок 5.1. Структура семантического анализатора

Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа.

5.2 Функции семантического анализатора

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

5.3 Структура и перечень семантических ошибок

Сообщения, формируемые семантическим анализатором, представлены в приложении Г.

5.4 Принцип обработки ошибок

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

5.5 Контрольный пример

Таблица 5.5 представляет собой контрольный пример исходного кода с возможными ошибками, которые могут возникнуть при компиляции. В ней демонстрируются сценарии неправильного использования ключевых слов, повторных объявлений идентификаторов и прочих недочетов.

Таблица 5.5 – Таблица возможных ошибок исходного кода

Исходный код	Текст сообщения
main [new Numb b; return 0;];	Ошибка 317: [SEM]# Ошибка в объявление идентификатора (указан неправильный тип) Строка 3 позиция 5
main [numb b; return 0;];	Ошибка 302: [SEM]# В объявлении отсутствует ключевое слово new Строка 3 позиция 6
main [new numb b; new numb b; return 0;];	Ошибка 311: [SEM]# Повторное объявление идентификатора Строка 4 позиция 5
main [new numb b; new numb b; return 0;]; main[new stroke b; new numb e; return 0;]	Ошибка 308: [SEM]# Обнаружено несколько точек входа в main Строка 6 позиция 21

Анализ таких контрольных примеров помогает разработчикам лучше понимать возможные проблемы в своем коде

6. Вычисление выражений

6.1 Выражения, допускаемые языком

В языке PYA-2024 допускаются вычисления выражений целочисленного, а также логического типов данных с поддержкой вызова функций внутри целочисленных выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1. Приоритет операций

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

Чем выше приоритет имеет операция, тем левее она будет находиться после преобразования в польскую запись.

7. Генерация кода

7.1 Структура генератора кода

В языке PYA-2024 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. Структура генератора кода PYA-2024 представлена на рисунке 7.1.

Рисунок 7.1 – Структура генератора кода

В случае возникновения ошибок генерация кода не будет осуществляться.

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 PYA-2024 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка и языка ассемблера

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

При представлении переменной в языке Assembler, необходимый тип идентификатора пишется после нее через двоеточие.

7.3 Статическая библиотека

В языке PYA-2024 предусмотрена статическая библиотека. Статическая

библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера. Стандартная библиотека находится в директории языка и при генерации кода подключается автоматически. Путь к библиотеке генерируется автоматически на стадии генерации кода.

7.4 Особенности алгоритма генерации кода

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

7.5 Входные параметры генератора кода

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке PYA-2024 . Результаты работы генератора кода выводятся в файл с расширением .asm.

7.6 Контрольный пример

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Д. Результат работы контрольного примера приведен в приложении Д.

8. Тестирование транслятора

8.1 Общие положения

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

8.2 Результаты тестирования

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

Таблица 8.1 – Результаты тестирования транслятора

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

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

Заключение

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

1. Сформулирована спецификация языка PYA-2024;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведенная 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 PYA-2024 включает:

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

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

Список использованных источников

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

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

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

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

Приложение А

Листинг 1 – Исходный код программы на языке PYA-2024

```
 numb func max (numb q, numb v, numb z)[
 new numb result;
 new numb k = 123;
 new stroke str1 ;
 state:q<v$
 correctly:[
 state:v<z$
 correctly:[result = 45/15*2;
 str1='some text';]
     wrong:[result = k*q*z;]
    $
   ]
  wrong:[result = 13;]
 $
 new numb len;
 write str1;
 len = strlen(str1);
 write 'Length of str1 is ';
 writeline len;
 return len;
];

numb func circuit(numb start , numb end)[
 new numb a =2;
 new numb iter = 1;
 state: start<end$
  cycle[
   writeline start;
   start = start+2;
   iter=iter*2;
  ]$
 return iter;
];

main
[
 new numb a = 1;
 new numb b =10;
 new numb c =12;
 new symbol S = "a";
 new boolean b1 = false;
 new numb result = max(a,b,c);
 write 'Result of function max:';
 writeline result;
 write 'ff:';
 new numb start = input(start);
 new numb end = input(end);
 new numb k = circuit(start,end);
 write 'Result of function loo :';
```

```
 writeline k;
 new numb ran = rand(1000);
 write 'Rand number :';
 writeline ran;
new numb aresult = abs(2);
writeline aresult;

new numb bresult = sqrt(25);
writeline bresult;

 return 0;
]
```

Приложение Б

Листинг 1 – Таблица контроля входных символов

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

Листинг 2 – Структура таблицы лексем

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

struct LexTable
```

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

Листинг 3 – Пример конечного автомата

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

Листинг 4 – Структура таблицы идентификаторов

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

Листинг 5 – Сообщения об ошибках стадии лексического анализа

```
ERROR_ENTRY(120,"[LEX]# Превышен максимльный размер таблицы
лексем"),
ERROR_ENTRY(121,"[LEX]# Таблица лексем переполнена"),
```

```
ERROR_ENTRY_NODEF(122), ERROR_ENTRY_NODEF(123),
ERROR_ENTRY_NODEF(124),
ERROR_ENTRY_NODEF(125), ERROR_ENTRY_NODEF(126),
ERROR_ENTRY_NODEF(127), ERROR_ENTRY_NODEF(128),
ERROR_ENTRY_NODEF(129),  ERROR_ENTRY_NODEF10(130),
ERROR_ENTRY_NODEF10(140), ERROR_ENTRY_NODEF10(150),
ERROR_ENTRY(160,"[LEX]# Превышено максимальное количество строк в
таблице идентификаторов"),
ERROR_ENTRY(161,"[LEX]# Таблица идентификаторов переполнена"),
ERROR_ENTRY_NODEF(162),
ERROR_ENTRY(163,"[LEX]# Не удалось распознать цепочку"),
ERROR_ENTRY(164,"[LEX]# Размер литерала больше максимально
допустимого"),
ERROR_ENTRY(165,"[LEX]# Отсутствует точка входа main"),
```

Листинг 6 – Таблица идентификаторов контрольного примера

```
-------------------------------------------------------------------------------------------------------
   №  | Scope      | идентификатор | тип данных | тип идентификатора | индекс в ТЛ | Значение
-------------------------------------------------------------------------------------------------------
0000 | max        | max           | integer    | функция            | 2           | -
0001 | max        | q             | integer    | параметр           | 5           | 0
0002 | max        | v             | integer    | параметр           | 8           | 0
0003 | max        | z             | integer    | параметр           | 11          | 0
0004 | max        | result        | integer    | переменная         | 16          | 0
0005 | max        | k             | integer    | переменная         | 20          | 0
0006 | max        | L0            | integer    | литерал            | 23          | 123
0007 | max        | str1          | string     | переменная         | 26          | ""
0008 | max        | L1            | integer    | литерал            | 49          | 45
0009 | max        | L2            | integer    | литерал            | 51          | 15
0010 | max        | L3            | integer    | литерал            | 53          | 2
0011 | max        | L4            | string     | литерал            | 57          | 'some text'
0012 | max        | L5            | integer    | литерал            | 79          | 13
0013 | max        | len           | integer    | переменная         | 84          | 0
0014 | max        | strlen        | integer    | функция            | 92          | -
0015 | max        | L6            | string     | литерал            | 98          | 'Length of str1 is '
0016 | circuit    | circuit       | integer    | функция            | 109         | -
0017 | circuit    | start         | integer    | параметр           | 112         | 0
0018 | circuit    | end           | integer    | параметр           | 115         | 0
0019 | circuit    | a             | integer    | переменная         | 120         | 0
0020 | circuit    | iter          | integer    | переменная         | 126         | 0
0021 | circuit    | L7            | integer    | литерал            | 129         | 1
0022 | main       | a             | integer    | переменная         | 164         | 0
0023 | main       | b             | integer    | переменная         | 170         | 0
0024 | main       | L8            | integer    | литерал            | 173         | 10
0025 | main       | c             | integer    | переменная         | 176         | 0
0026 | main       | L9            | integer    | литерал            | 179         | 12
0027 | main       | S             | Char       | переменная         | 182         | ""
0028 | main       | L10           | Char       | литерал            | 185         | "a"
0029 | main       | b1            | Boolean    | переменная         | 188         | -
0030 | main       | L11           | Boolean    | литерал            | 191         | false
0031 | main       | L12           | string     | литерал            | 194         | 'ff:'
0032 | main       | start         | integer    | переменная         | 197         | 0
0033 | main       | input         | integer    | функция            | 200         | -
0034 | main       | end           | integer    | переменная         | 206         | 0
0035 | main       | input         | integer    | функция            | 209         | -
0036 | main       | aresult       | integer    | переменная         | 215         | 0
0037 | main       | abs           | integer    | функция            | 218         | -
0038 | main       | dresult       | integer    | переменная         | 227         | 0
0039 | main       | rand          | integer    | функция            | 230         | -
0040 | main       | L13           | integer    | литерал            | 232         | 1000
0041 | main       | bresult       | integer    | переменная         | 239         | 0
0042 | main       | sqrt          | integer    | функция            | 242         | -
0043 | main       | L14           | integer    | литерал            | 244         | 25
-------------------------------------------------------------------------------------------------------
```

Листинг 7 – Таблица лексем контрольного примера

----------Таблица лексем----------

0000	tfi(ti,ti,ti)[	000 - 013
0001	nti;	013 - 017
0002	nti=l;	017 - 023
0003	nti;	023 - 027
0004	?:i<i$	027 - 033
0005	c:[	033 - 036
0006	?:i<i$	036 - 042
0007	c:[i=l/l*l;	042 - 053
0008	i=l;]	053 - 058
0009	w:[i=i*i*i;]	058 - 070
0010	$	070 - 071
0011	]	071 - 072
0012	w:[i=l;]	072 - 080
0013	$	080 - 081
0014	nti;	081 - 085
0015	pi;	085 - 088
0016	i=S(i);	088 - 095
0017	pl;	095 - 098
0018	di;	098 - 101
0019	ri;	101 - 104
0020	];	104 - 106
0021	tfi(ti,ti)[	106 - 117
0022	nti=l;	117 - 123
0023	nti=l;	123 - 129
0024	?:i<i$	129 - 135
0025	v[	135 - 137
0026	di;	137 - 140

```
0027 | i=i+l;                                    | 140 - 146 |

0028 | i=i*l;                                    | 146 - 152 |

0029 | ]$                                        | 152 - 154 |

0030 | ri;                                       | 154 - 157 |

0031 | ];                                        | 157 - 159 |

0032 | m                                         | 159 - 160 |

0033 | [                                         | 160 - 161 |

0034 | nti=l;                                    | 161 - 167 |

0035 | nti=l;                                    | 167 - 173 |

0036 | nti=l;                                    | 173 - 179 |

0037 | nti=l;                                    | 179 - 185 |

0038 | nti=l;                                    | 185 - 191 |

0039 | pl;                                       | 191 - 194 |

0040 | nti=U(i);                                 | 194 - 203 |

0041 | nti=U(i);                                 | 203 - 212 |

0042 | nti=A(l);                                 | 212 - 221 |

0043 | di;                                       | 221 - 224 |

0044 | nti=R(l);                                 | 224 - 233 |

0045 | di;                                       | 233 - 236 |

0046 | nti=C(l);                                 | 236 - 245 |

0047 | di;                                       | 245 - 248 |

0048 | rl;                                       | 248 - 251 |

0049 | ----------------------------------------------------------------
```

Приложение В

Листнинг 1 – Грамматика языка PYA-2024

```
Greibach greibach(NS('S'), TS('$'),
          15,
          Rule(NS('Z'), GRB_ERROR_SERIES + 1,
               1,
               Rule::Chain(5, TS('f'), TS('i'), TS('('), NS('G'),
TS(')'))
          ),
          Rule(NS('S'), GRB_ERROR_SERIES + 0,
               3,
               Rule::Chain(7, TS('t'), NS('Z'), TS('['), NS('N'),
TS(']'), TS(';'), NS('S')),
               Rule::Chain(6, TS('t'), NS('Z'), TS('['), NS('N'),
TS(']'), NS('S')),
               Rule::Chain(4, TS('m'), TS('['), NS('N'), TS(']'))
          ),
          Rule(NS('G'), GRB_ERROR_SERIES + 2,
               6,
               Rule::Chain(1, TS('i')),
               Rule::Chain(1, TS('l')),
               Rule::Chain(2, TS('t'), TS('i')),
               Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('G')),
               Rule::Chain(3, TS('i'), TS(','), NS('G')),
               Rule::Chain(3, TS('l'), TS(','), NS('G'))
          ),

          Rule(NS('V'), GRB_ERROR_SERIES + 3,
               4,
               Rule::Chain(1, TS('+')),
               Rule::Chain(1, TS('-')),
               Rule::Chain(1, TS('*')),
               Rule::Chain(1, TS('/'))
          ),
          Rule(NS('O'), GRB_ERROR_SERIES + 4,
               8,
               Rule::Chain(1, TS('|')),
               Rule::Chain(1, TS('&')),
               Rule::Chain(1, TS('!')),
               Rule::Chain(1, TS('%')),
               Rule::Chain(1, TS('>')),
               Rule::Chain(1, TS('<')),
               Rule::Chain(1, TS('}')),
               Rule::Chain(1, TS('{'))
          ),
          Rule(NS('Q'), GRB_ERROR_SERIES + 5,
               2,
               Rule::Chain(3, TS('('), NS('M'), TS(')')),
               Rule::Chain(5, TS('('), NS('M'), TS(')'), NS('V'),
NS('E'))
          ),
```

```
          Rule(NS('L'), GRB_ERROR_SERIES + 6,
               5,
               Rule::Chain(2, TS('R'), NS('Q')),
               Rule::Chain(2, TS('S'), NS('Q')),
               Rule::Chain(2, TS('U'), NS('Q')),
               Rule::Chain(2, TS('A'), NS('Q')),
               Rule::Chain(2, TS('C'), NS('Q'))
          ),

          Rule(NS('D'), GRB_ERROR_SERIES + 7,
               1,
               Rule::Chain(2, TS('t'), NS('W'))
),
          Rule(NS('W'), GRB_ERROR_SERIES +8,
               4,
               Rule::Chain(3, TS('i'), TS('='), NS('M')),
               Rule::Chain(3, TS('i'), TS('='), NS('E')),
               Rule::Chain(3, TS('i'), TS('='), NS('L')),
               Rule::Chain(5, TS('i'), TS('='), NS('E'),
NS('V'),NS('E'))
          ),

          Rule(NS('E'), GRB_ERROR_SERIES + 9,
               12,
               Rule::Chain(1, TS('i')),
               Rule::Chain(1, TS('l')),
               Rule::Chain(2, TS('i'), TS(';')),
               Rule::Chain(2, TS('l'), TS(';')),
               Rule::Chain(3, TS('i'), NS('V'), NS('E')),
               Rule::Chain(3, TS('l'), NS('V'), NS('E')),
               Rule::Chain(3, TS('i'), NS('O'), NS('E')),
               Rule::Chain(3, TS('l'), NS('O'), NS('E')),
               Rule::Chain(2, TS('S'), NS('Q')),
               Rule::Chain(2, TS('U'), NS('Q')),
               Rule::Chain(2, TS('R'), NS('Q')),
               Rule::Chain(4, TS('i'), TS('('), NS('G'), TS(')'))
          ),

          Rule(NS('M'), GRB_ERROR_SERIES + 10,
               2,
               Rule::Chain(1, TS('l')),
               Rule::Chain(1, TS('i'))

          ),

          Rule(NS('K'), GRB_ERROR_SERIES + 11,
               3,
               Rule::Chain(5, TS(':'), NS('E'), TS('$'), NS('A'),
TS('$')),
               Rule::Chain(4, TS(':'), NS('E'), TS('$'), TS('$')),
               Rule::Chain(5, TS(':'), NS('E'), TS('$'), TS('$'),
NS('N'))
          ),
```

```
		Rule(NS('A'), GRB_ERROR_SERIES + 12,
			3,
			Rule::Chain(3, TS('c'), TS(':'), NS('Y')),
			Rule::Chain(3, TS('w'), TS(':'), NS('Y')),
			Rule::Chain(2, TS('v'), NS('Y'))
		),
Rule(NS('Y'),GRB_ERROR_SERIES+13,
			2,
			Rule::Chain(4, TS('['), NS('N'), TS(']'), NS('A')),
			Rule::Chain(3, TS('['), NS('N'), TS(']'))
			),

		Rule(NS('N'), GRB_ERROR_SERIES + 14,
			12,
			Rule::Chain(4, TS('n'), TS('t'), TS('i'), TS(';')),
			Rule::Chain(5, TS('n'), TS('t'), TS('i'), TS(';'),
NS('N')),
			Rule::Chain(4, TS('n'), NS('D'), TS(';'), NS('N')),
			Rule::Chain(3, TS('n'), NS('D'), TS(';')),
			Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'),
NS('N')),
			Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),
			Rule::Chain(3, TS('p'), NS('E'), TS(';')),
			Rule::Chain(4, TS('p'), NS('E'), TS(';'), NS('N')),
	Rule::Chain(3, TS('?'), NS('K'), NS('N')),
			Rule::Chain(2, TS('?'), NS('K')),
			Rule::Chain(3, TS('r'), NS('M'), TS(';')),
			Rule::Chain(4, NS('N'), TS(';'), TS('r'), NS('M'))
		)
	);
```

Листнинг 2 – Структура магазинного автомата

```
struct Mfst
	{
		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 grebach;
		LT::LexTable lexTable;
		MFSTSTSTACK st;
		vector<MfstState> storestate;
		Mfst();
		Mfst(LT::LexTable& plexTable, GRB::Greibach pgrebach,
wchar_t parsfile[]);

		char* getCSt(char* buf);
		char* getCLenta(char* buf, short pos, short n = 25);
		char* getDiagnosis(short n, char* buf);
		bool savestate();
		bool resetstate();
		bool push_chain(GRB::Rule::Chain chain);
		RC_STEP step();
		bool start();
		bool savedDiagnosis(RC_STEP prc_step);

		void printRules();

		struct Deducation
		{
			short size;
			short* nrules;
			short* nrulechains;

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

		bool savededucation();
```

```
        ofstream* pars;
    };
```

Листинг 3 – Разбор исходного кода синтаксическим анализатором

```
Шаг: правило      иходная лента     стек
0   : S->tZ[N];S          tfi(ti,ti,ti)[nti;nti=l;n     S$
0   : SAVESTATE:          1
0   :                     tfi(ti,ti,ti)[nti;nti=l;n     tZ[N];S$
1   :                     fi(ti,ti,ti)[nti;nti=l;nt     Z[N];S$
Шаг: правило      иходная лента     стек
2   : Z->fi(G)            fi(ti,ti,ti)[nti;nti=l;nt     Z[N];S$
2   : SAVESTATE:          2
2   :                     fi(ti,ti,ti)[nti;nti=l;nt     fi(G)[N];S$
3   :                     i(ti,ti,ti)[nti;nti=l;nti     i(G)[N];S$
4   :                     (ti,ti,ti)[nti;nti=l;nti;     (G)[N];S$
5   :                     ti,ti,ti)[nti;nti=l;nti;?     G)[N];S$
Шаг: правило      иходная лента     стек
6   : G->ti               ti,ti,ti)[nti;nti=l;nti;?     G)[N];S$
6   : SAVESTATE:          3
6   :                     ti,ti,ti)[nti;nti=l;nti;?     ti)[N];S$
7   :                     i,ti,ti)[nti;nti=l;nti;?:     i)[N];S$
8   :                     ,ti,ti)[nti;nti=l;nti;?:i     )[N];S$
9   : TS_NOK/NS_NORULECHAIN
9   : RESTATE
9   :                     ti,ti,ti)[nti;nti=l;nti;?     G)[N];S$
Шаг: правило      иходная лента     стек
10  : G->ti,G             ti,ti,ti)[nti;nti=l;nti;?     G)[N];S$
10  : SAVESTATE:          3
10  :                     ti,ti,ti)[nti;nti=l;nti;?     ti,G)[N];S$
11  :                     i,ti,ti)[nti;nti=l;nti;?:     i,G)[N];S$
12  :                     ,ti,ti)[nti;nti=l;nti;?:i     ,G)[N];S$
13  :                     ti,ti)[nti;nti=l;nti;?:i<     G)[N];S$
Шаг: правило      иходная лента     стек
…
1913:                                                   $
1914: LENTA_END
1915: ------>LENTA_END
253 всего строк, синтаксический анализ выполнен без ошибок
0   : S->tZ[N];S
1   : Z->fi(G)
4   : G->ti,G
7   : G->ti,G
10  : G->ti
14  : N->nti;N
18  : N->nD;N
19  : D->tW
20  : W->i=M
22  : M->l
24  : N->nti;N
28  : N->?KN
29  : K->:E$A$
```

```
30   : E->iOE
31   : O->< 
```

Листинг 5 – Сообщения об ошибках стадии синтаксического анализа

```
ERROR_ENTRY(600,"[SIN]# Неверная структура программы"),
ERROR_ENTRY(601,"[SIN]# Неверная структура функции"),
ERROR_ENTRY(602,"[SIN]# Ошибка в параметрах функции"),
ERROR_ENTRY(603,"[SIN]# Ошибка в арифметической операции"),
ERROR_ENTRY(604,"[SIN]# Ошибка в логической операции"),
ERROR_ENTRY(605,"[SIN]# Ошибка в параметрах функции статической
библиотеки"),
ERROR_ENTRY(606,"[SIN]# Ошибка в вызове стандартной функции"),
ERROR_ENTRY(607,"[SIN]# Ошибка при инициализация переменной"),
ERROR_ENTRY(608,"[SIN]# Ошибка присвоения значения идентификатору"),
ERROR_ENTRY(609,"[SIN]# Ошибка в выражении"),
ERROR_ENTRY(610,"[SIN]# Ожидается идентификатор или литерал"),
ERROR_ENTRY(611,"[SIN]# Ошибка в структуре условного выражения"),
ERROR_ENTRY(612,"[SIN]# Ошибка построения условного выражения"),
ERROR_ENTRY(613,"[SIN]# Ошибка тела выражения/цикла"),
ERROR_ENTRY(614,"[SIN]# Ошибка в объявлении переменной"),
```

Приложение Г

Листинг 1 – Сообщения об ошибках стадии семантического анализа

```
ERROR_ENTRY(300,"[SEM]# Превышен максимальный размер number типа"),
ERROR_ENTRY(301,"[SEM]# В объявлении не указан тип идентификатора"),
ERROR_ENTRY(302,"[SEM]# В объявлении отсутствует ключевое слово
new"),
ERROR_ENTRY(303,"[SEM]# Возвращаемый функцией тип не соответствует
типу функции"),
ERROR_ENTRY(304,"[SEM]# Ошибка в условии условного выражения"),
ERROR_ENTRY(305,"[SEM]# Типы данных в выражении не совпадают"),
ERROR_ENTRY(306,"[SEM]# Деление на ноль"),
ERROR_ENTRY(307,"[SEM]# Необъявленный идентификатор"),
ERROR_ENTRY(308,"[SEM]# Обнаружено несколько точек входа в main"),
ERROR_ENTRY(309,"[SEM]# Обнаружен \' . Возможно, не закрыт строковый
литерал"),
ERROR_ENTRY(310,"[SEM]# В Symbol типе не может быть более 1
символа"),
ERROR_ENTRY(311,"[SEM]# Повторное объявление идентификатора"),
ERROR_ENTRY(312,"[SEM]# Ошибка в вызове Стандартной функции "),
ERROR_ENTRY(313,"[SEM]# Эта операция не применима к типу
symbol/stroke"),
ERROR_ENTRY(314,"[SEM]# В стандартную функцию передан необъявленный
идентификатор"),
ERROR_ENTRY(315,"[SEM]# Ожидается тип stroke/symbol"),
ERROR_ENTRY(316,"[SEM]# Превышен максимальный размер типа stroke"),
ERROR_ENTRY(317,"[SEM]# Ошибка в объявление идентификатора (указан
неправильный тип)"),
ERROR_ENTRY(318,"[SEM]# Ожидается ;"),
ERROR_ENTRY(319,"[SEM]# Ожидается только одна ;"),
```

Приложение Д

Листинг 1 – Результат генерации кода контрольного примера в Ассемблер

```
.586
.model flat, stdcall
includelib libucrt.lib
includelib kernel32.lib
includelib ../Debug/GenLib.lib
ExitProcess         PROTO:DWORD
Remainder      PROTO : DWORD, :DWORD
Rand           PROTO : DWORD
Abs            PROTO : DWORD
Sqrt           PROTO : DWORD
Input               PROTO : DWORD
Writestroke         PROTO : DWORD
WriteNumb      PROTO : DWORD
Writelinestroke          PROTO : DWORD
WritelineNumb       PROTO : DWORD
Strlen              PROTO : DWORD
.stack 4096
.const
          L0 SDWORD 123
          L1 SDWORD 45
          L2 SDWORD 15
          L3 SDWORD 2
          L4 byte 'some text', 0
          L5 SDWORD 13
          L6 byte 'Length of str1 is ', 0
          L7 SDWORD 1
          L8 SDWORD 10
          L9 SDWORD 12
          L10 byte "a", 0
          TRUE equ 1
          FALSE equ 0
          L11 word 0
          L12 byte 'ff:', 0
          L13 SDWORD 1000
          L14 SDWORD 25
.data
          maxresult sdword 0
          maxk sdword 0
          maxstr1 dword ?
          maxlen sdword 0
          circuita sdword 0
          circuititer sdword 0
```

```
          maina sdword 0
          mainb sdword 0
          mainc sdword 0
          mainS dword ?
          mainb1 word ?
          mainstart sdword 0
          mainend sdword 0
          mainaresult sdword 0
          maindresult sdword 0
          mainbresult sdword 0
.code
;----------- max ------------
max PROC,
     maxq : sdword, maxv : sdword, maxz : sdword
push ebx
push edx
push L0
pop ebx
mov maxk, ebx
mov edx, maxq
cmp edx, maxv
jl right1
jg wrong1
     right1:
mov edx, maxv
cmp edx, maxz
jl right2
jg wrong2
     right2:
push L1
pop ebx
pop eax
cdq
idiv ebx
push eax
push L2
pop ebx
pop eax
imul eax, ebx
push eax
push L3
pop ebx
mov maxresult, ebx
mov maxstr1, offset L4
jmp next2
```

```
     wrong2:
push maxk
pop ebx
pop eax
imul eax, ebx
push eax
push maxq
pop ebx
pop eax
imul eax, ebx
push eax
push maxz
pop ebx
mov maxresult, ebx
next2:
jmp next1
     wrong1:
push L5
pop ebx
mov maxresult, ebx
next1:
push maxstr1
call WriteStroke
push maxstr1
pop ebx
mov maxlen, ebx
push offset L6
call WriteStroke
push maxlen
call WritelineNumb
pop ebx
pop edx
mov eax, maxlen
ret
max ENDP
;------------------------------
;----------- circuit ------------
circuit PROC,
     circuitstart : sdword, circuitend : sdword
push ebx
push edx
push L3
pop ebx
mov circuita, ebx
push L7
```

```
pop ebx
mov circuititer, ebx
mov edx, circuitstart
cmp edx, circuitend
jl repeat3
jmp repeatnext3
repeat3:
push circuitstart
call WritelineNumb
push circuitstart
pop ebx
pop eax
add eax, ebx
push eax
push L3
pop ebx
mov circuitstart, ebx
push circuititer
pop ebx
pop eax
imul eax, ebx
push eax
push L3
pop ebx
mov circuititer, ebx
mov edx, circuitstart
cmp edx, circuitend
jl repeat3
repeatnext3:
pop ebx
pop edx
mov eax, circuititer
ret
circuit ENDP
;------------------------------
;----------- MAIN ------------
main PROC
push L7
pop ebx
mov maina, ebx
push L8
pop ebx
mov mainb, ebx
push L9
pop ebx
```

```
mov mainc, ebx
mov mainS, offset L10
mov cx, L11
mov mainb1, cx
push offset L12
call WriteStroke
push circuitstart
pop ebx
mov mainstart, ebx
push circuitend
pop ebx
mov mainend, ebx
push L3
pop ebx
mov mainaresult, ebx
push mainaresult
call WritelineNumb
push L13
pop ebx
mov maindresult, ebx
push maindresult
call WritelineNumb
push L14
pop ebx
mov mainbresult, ebx
push mainbresult
call WritelineNumb
;------------------------------
push 0
call ExitProcess
main ENDP
end main
```