“ПОСТРОЕНИЕ ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА”

1.Краткие теоретические сведения

Лексический анализатор (сканер) решает задачу синтаксического анализа на уровне лексем.

Под лексемой понимается некоторый класс эквивалентности на множестве (словаре) терминальных символов формальной грамматики, порождающей операторы языка программирования.

В задачи лексического анализатора входят: а) выделение лексем (атомов, терминальных символов) языка из потока литер, проверка правильности их написания; б) замена цепочки литер, соответствующей опознанной лексеме, определенным внутренним кодом, так называемым дескриптором; в) ведение таблиц соответствия объектов программы системе кодирования: таблиц констант, таблиц имен переменных, таблиц размещения лексем по строкам; г) устранение пробелов, комментариев и аналогичных элементов визуального восприятия программы, и представление исходной программы последовательностью дескрипторов.

На контрольную работу выносятся пункты а), б), и г) из перечня задач.

Под дескриптором понимают пару вида (код_класса_лексемы, ссылка_на_значение).

В языке программирования традиционно присутствуют следующие классы лексем: имена переменных (идентификаторы), константы, знаки операций, знаки пунктуации (скобки, разделители списка параметров, разделители операторов).

Иногда, с целью упрощения процедур синтаксического анализа, классы знаков операций и знаков пунктуации сводят до одного символа в каждом классе.

Лексический анализ выполняется с использованием детерминированных конечных автоматов. Поэтому для построения сканера необходимо разработать конечный автомат, выполняющий классификацию цепочек литер, и в заключительных состояниях помещающий необходимый дескриптор в файл или массив выходных данных.

Процедура построения сканера включает следующие шаги:

1. Определение лексем языка программирования.
2. Назначение кодов лексемам.
3. Проектирование конечного автомата, на базе которого будет осуществляться анализ.
4. Составление перечня диагностических сообщений об ошибках в тех случаях, когда функция перехода конечного автомата не определена.
5. Определение алгоритмов и процедур, связанных с обработкой информации в пределах множества состояний конечного автомата.

6. Разработка тестовых последовательностей.
7. Программирование и отладка сканера.

1.Пример выполнения контрольной работы

1.1. Пусть исходные данные имеют вид, представленный ниже

Служебные слова:	STA – начало программы. STO – окончание.
Описание идентификатора:	начинается буквой латинского алфавита, а заканчивается буквой E.
Однолитерные разделители:	+,- ,* ,/.
Двулитерные разделители:	: =.
Константа:	восьмеричная дробь, например – 617.214
Фрагмент программы для анализа:	STA xE := yE+1.1 y123E := xE*yE - 2.257 STO

1.2 Определение лексем и назначение им кодов

Коды лексемам назначим, руководствуясь соображениями об их уникальности и удобстве эффективной обработки, поэтому класс однолитерных разделителей имеет, например, в коде префикс 5.

STA	100
STO	200
идентификатор, <iden>	300
константа, <data>	400
+	501
-	502
*	503
/	504
:=	600

1.3 Построение минимального детерминированного конечного автомата

Точное описание конечного автомата удается получить на основании цепочек литер с использованием регулярных выражений. Правила конструирования регулярных выражений изложены в [9, 10, 12].

Общая структура регулярного выражения примерно такова:

$Cc_1L_1 \cup Cc_2L1 \cup <iden>L_1 \cup <data>L_2 \cup d_1 \cup d_2 \cup \ldots \cup d_j \cup \ldots \cup d_n \cup L_3$.

В записи обозначено:

Cc_i – i-тое служебное слово.

L_1 – лексема, состоящая в том, что текущая литера не буква или не цифра.

L_2 – текущая литера не восьмеричная цифра.

d_j – разделители j-того типа.

L_3 – литера, не принадлежащая алфавиту конечного автомата.

Служебные слова являются самоопределяющимися цепочками, составленными из литер{S,T,A,O} – STA, STO.

Эскизно идентификатор (переменная) будет выглядеть так

$$\boldsymbol{Б\{Б \cup Ц\}Е},$$

где ***Б*** - любая латинская буква, ***Ц*** – любая десятичная цифра.

Восьмеричная дробная константа описывается следующим образом:

$$\{ц\}\,ц \bullet \{ц\}\,ц,$$

где ***ц*** – восьмеричная цифра из диапазона 0÷7.

С учетом того, что конечный автомат должен отличать отдельные буквы и цифры на фоне других, имеем:

1. Множество букв ***Б*** = {S, T, A, O, E, δ}.

δ - буква латинского алфавита, не совпадающая по начертанию с S,T,A,O,E.

2. Множество цифр ***Ц*** = {8, 9, *ц*}
3. Множество «не буква, не цифра» L_1 ={ +,-,*, /, :=, •}$\cup$ L_3
4. Множество «не восьмеричная цифра» L_2 = Б$\cup${8,9}$\cup L1$

Окончательно получаем дизъюнктивные члены выражения, описывающие конечный автомат:

1. $STAL_1$ – первое служебное слово.
2. $STOL_1$ – второе служебное слово.
3. (S$\cup$T$\cup$A$\cup$O$\cup$E$\cup\delta$) {S$\cup$T$\cup$A$\cup$O$\cup$E$\cup\delta\cup$8$\cup$9$\cup$*ц*}EL_1 – переменная.
4. *{ц}ц•{ц}ц* L_2- константа.
5. +$\cup$ - $\cup$ * $\cup$ / $\cup$:= $\cup$ L_3 – однолитерные и двулитерные разделители.

Регулярное выражение подвергается минимизации по алгоритму, изложенному в [8].

Процесс разметки и минимизации представлен на рисунке 1.1. На фрагменте 1.1, а, показано выражение до минимизации, на 1.1, б, – окончательно минимизированное выражение.

В результирующем выражении видно, что по цепочкам литер, в которых начало идентификатора полностью или частично совпадает со служебным сло-

вом, автомат является недетерминированным. Это образы переходов $\delta(1,T) = \{2, 13\}$, $\delta(\{2, 13\}, A) = \{3, 14\}$ и т.д.

Для приведения недетерминированного конечного автомата к детерминированному, можно воспользоваться классическими алгоритмами приведения [7 – 10], но можно воспользоваться описанием КА в виде графа и с его помощью привести таблицу переходов ДКА. Фрагмент графа конечного автомата представлен на рисунке 1.1, а в таблице 1.1 – представлена функция переходов конечного автомата.

Фрагмент графа конечного автомата, отображенный на рисунке 1.2, показан в столбцах таблицы 1.1 с номерами 1 и 2 значениями, заключенными в скобки. Прочие элементы указанных столбцов записываются согласно разметке части регулярного выражения, относящегося к описанию идентификатора регулярного выражения, заключенного в фигурные скобки (смотрите рисунок 1.1,б, основные места с 12-го по 20-е).

Приступим к минимизации таблицы, описывающий конечный автомат. Приемы минимизации изложены в [4, 9, 11], согласно им, под минимизацию подпадают столбцы таблицы, соответствующие состояниям конечного автомата, не различимым по входной строке и по выходному сигналу. В качестве выходных сигналов выберем следующие: сигнал рабочего состояния для всех состояний, кроме заключительных, соответствующих опознанным лексемам, это 4, 6, 21, 25 – 28, 31, 32. В каждом из этих состояний генерируется свой уникальный сигнал. Последние состояния можно удалить из таблицы как неинформативные, хотя функции перехода КА в эти состояния оставлены.

Множества неразличимых состояний, минимизируемые состояния, суть $\{7 – 15, 17 – 20\}$. Конечные состояния $\{4, 6, 21, 25 – 28, 31, 32\}$ тоже устраним из таблицы переходов, а соответствующую позицию функции $\delta(q_r,a_r)$ определим используя коды, присвоенные этим лексемам на этапе кодирования.

Для осуществления корректной минимизации, в таблице 1.2 приводится система перенумерации состояний для таблицы минимального автомата.

Для случаев, когда образ функции переходов КА пуст (см. исходную таблицу 1.1), введём коды ошибок, которые в ходе работы сканера помогут генерировать осмысленные диагностические сообщения.

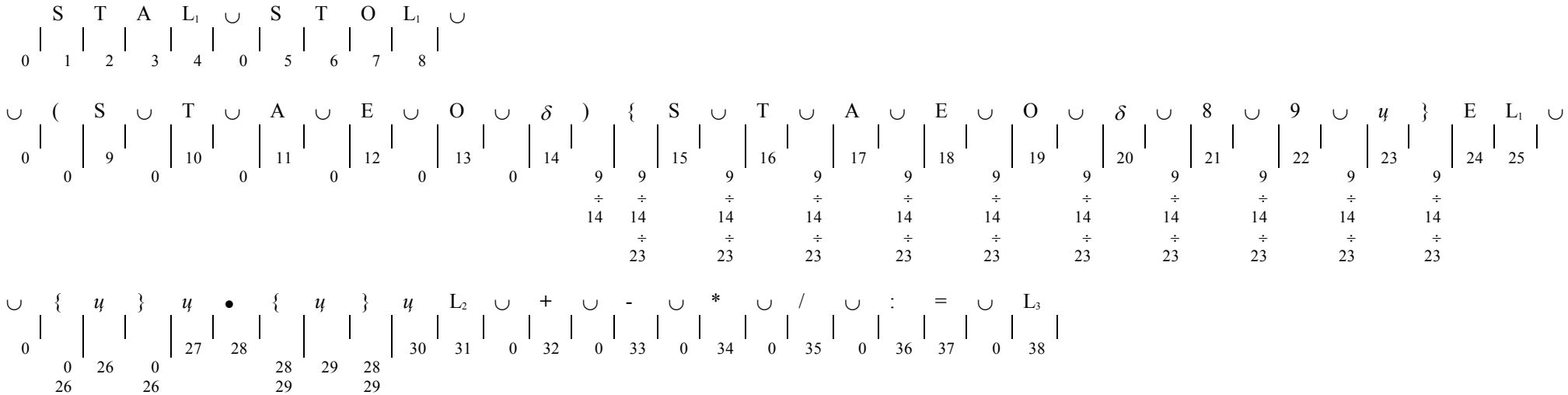
1. Некоректный фрагмент цепочки. Состояния 0, строки соответствующие литерам {8, 9, •, =}, код 801.

2. Ошибка в служебном слове. Состояния {1, 2}, строки, соответствующие L_1, L_3, код 802.

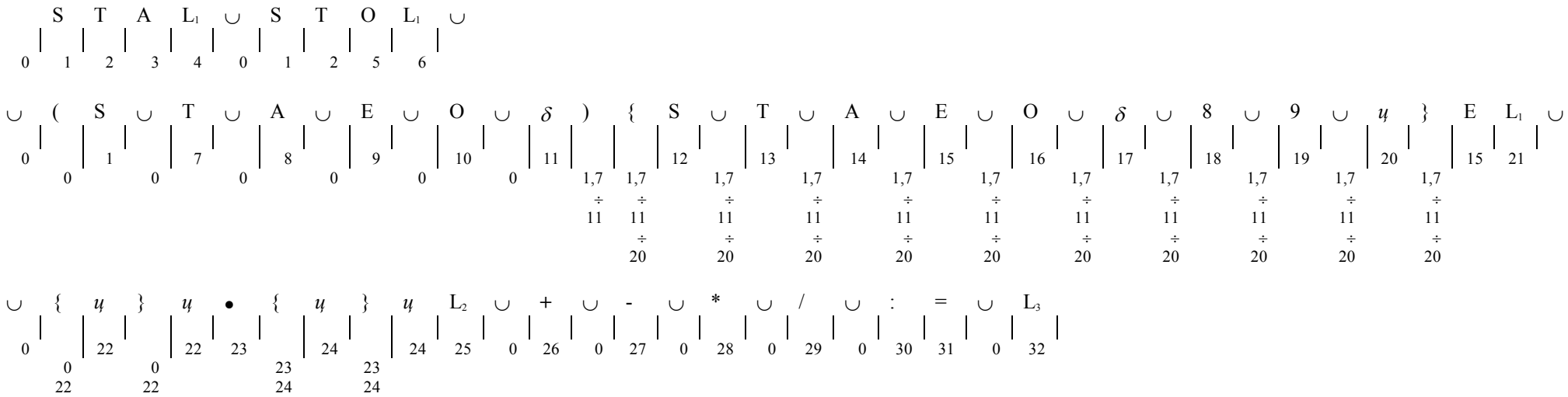
3. Ошибка в написании имени переменной. Состояния {7 – 15, 17 – 20}, строки соответствующие L_1, L_2, код 803.

4. Ошибочная константа. Состояние 22, строки, соответствующие множеству L_2 \ ”•”. Состояние 23, все строки, кроме «ц», код 804.

5. Ошибка в операторе присваивания. Состояние 30, все строки, кроме «=», код 805.



а)



б)

Рисунок 1.1 – Минимизация конечного автомата по регулярному выражению

Таблица 1.1 – Функция перехода ДКА, построенная по минимизированному регулярному выражению

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
S	1	12	12	12	Служебное слово STA	12	Служебное слово STO	12	12	12	12	12	12	12	12	12	12	12	12	12	12	Идентификатор			25	Восьмеричная константа	Знак +	Знак -	Знак *	Знак /		Оператор присваивания	Синтаксическая ошибка
T	7	(2)	13	13		13		13	13	13	13	13	13	13	13	13	13	13	13	13	13				25								
A	8	14	(3)	14		14		14	14	14	14	14	14	14	14	14	14	14	14	14	14				25								
O	9	16	(5)	16		15		16	16	16	16	16	16	16	16	16	16	16	16	16	16				25								
E	10	15	15	15		16		15	15	15	15	15	15	15	15	15	15	15	15	15	15				25								
δ	11	17	17	17		17		17	17	17	17	17	17	17	17	17	17	17	17	17	17				25								
8		18	18	18		18		18	18	18	18	18	18	18	18	18	18	18	18	18	18				25								
9		19	19	19		19		19	19	19	19	19	19	19	19	19	19	19	19	19	19				25								
ц	22	20	20	20		20		20	20	20	20	20	20	20	20	20	20	20	20	20	20		22	24	24								
+	26			4		6										21									25								
-	27			4		6										21									25								
*****	28			4		6										21									25								
/	29			4		6										21									25								
:	30			4		6										21									25								
·				4		6										21							23		25								
=				4		6										21									25						31		
L3	32			4		6										21									25								

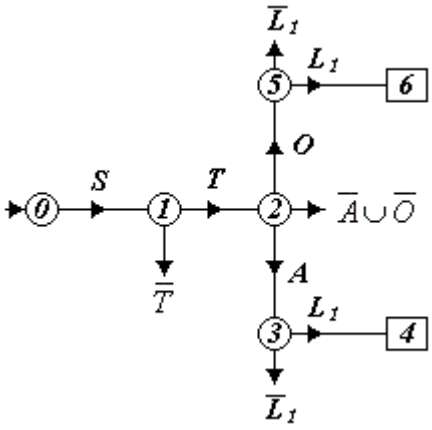


Рисунок 1.2 – Фрагмент КА, относящийся к разбору служебных слов

Функция переходов минимального детерминированного конечного автомата приводится в таблице 1.3.

В алфавит входных символов конечного автомата добавлен пробел, пополняющий множество L_1(не буква, не цифра) для упрощения подготовки тестовых данных с позиции их наглядности (читабельности).

Таблица 1.2 – Соответствие старых и новых состояний МДКА

Множество старых состояний	**Новые состояния**
0	0
1	1
2	2
3	3
4	Исключено, код 100
5	4
6	Исключено, код 200
7-14,16-20	5
15	6
21	Исключено, код 300
22	7
23	8
24	9
25	Исключено, код 400
26	Исключено, код 501
27	Исключено, код 502
28	Исключено, код 503
29	Исключено, код 504
30	10
31	Исключено, код 600
32	Исключено, код 700

1.4 Методология разработки программы сканера

В общем случае алгоритм работы программы моделирования автомата может выглядеть таким образом:
организация цикла чтения из файла по одной литере до достижения его конца. Номер состояния КА предварительно должен быть положен 0 (нулю), файл, в который будут записаны дескрипторы, открыт. В теле цикла выполняются следующие действия:

1. По текущей литере определяется номер строки таблицы функции переходов.

2. Пара (номер строки, номер состояния) однозначно определяет значение функции перехода.

3. Код значения функции перехода анализируется на принадлежность к : а) кодам рабочих состояний; б) кодам заключительных состояний; в) кодам ошибок.

Для вариантов б) и в) автомат устанавливается в начальное состояние. В зависимости от результатов анализа формируется выходной файл (случай б), диагностическое сообщение (случай в) или никаких действий не выполняется (случай а).

Таблица 1.3 – Переходы минимального ДКА

№		0	1	2	3	4	5	6	7	8	9	10	
0	S	1	5	5	5	5	5	5	804	804	400	805	
1	T	5	2	5	5	5	5	5	804	804	400	805	
2	A	5	5	3	5	5	5	5	804	804	400	805	
3	O	5	5	4	5	5	5	5	804	804	400	805	L_2
4	E	6	6	6	6	6	6	6	804	804	400	805	
5	δ	5	5	5	5	5	5	5	804	804	400	805	
6	8	801	5	5	5	5	5	5	804	804	400	805	
7	9	801	5	5	5	5	5	5	804	804	400	805	
8	Ц	7	5	5	5	5	5	5	7	9	9	805	
9	+	501	802	802	100	200	803	300	804	804	400	805	
10	-	502	802	802	100	200	803	300	804	804	400	805	
11	*	503	802	802	100	200	803	300	804	804	400	805	
12	/	504	802	802	100	200	803	300	804	804	400	805	L_1,
13	:	10	802	802	100	200	803	300	804	804	400	805	L_2
14	•	801	802	802	100	200	803	300	8	804	400	805	
15	=	801	802	802	100	200	803	300	804	804	400	600	
16		0	802	802	100	200	803	300	804	804	400	805	
17	L3	700	802	802	100	200	803	300	804	804	400	805	

Обратим внимание, что при опознавании сканером служебного слова, переменной или константы номер состояния КА полагается равным начальному, а **чтение очередной литеры** из входного файла **не выполняется!**

Для эффективного осуществления кодирования алгоритма, предполагается использовать операторы `switch`, `do…while`, `if` и встроенные функции `isalpha()`, `isdigit()` для экспресс – определения принадлежности литеры и `toupper()` для принудительной установки регистра букв в положение прописной буквы.

Фрагмент программы представлен ниже.

```
N_sost = 0;
if (( c = getc( ))! = EOF) do {
            if (isalpha(c)) /* Это буква? */
                switch(toupper(c) {
                    case “S”: N_str=0;break;
                    …   …   …
                    case “E”: N_str=4;break;
                    default: N_str=5; } /* δ */
            else if (isdigit(c)) /* Это цифра? */
                    switch (c) {
                    case “8”: N_str=6;break;
                    case “9”: N_str=7;break;
                    default: N_str=8; } /* 8-я цифра */
                else switch (c) {
                /* Это ни буква, ни цифра ? */
                    case “+”: N_str=9;break;
                    …   …   …
                    case “  ”: N_str=16;break;
                    default: N_str=17; }
    N_sost=upr_table[N_str][N_sost];
            /* Получение состояния перехода */
    if ((N_sost>99)&&(N_sost<500)) {
            /* Анализ кода состояния */
        fprintf(Prog_code,”%d”,N_sost);N_sost=0; }
    else if (N_sost>500)&&(N_sost<800) {
            fprintf(Prog_code,”%d”,N_sost);
            N_sost=0;c=getc( );}
        else if (N_sost>800) {
            switch(N_sost) {
                case 801:
                printf(“\n Неправильное начало”);break;
                case 805:
                printf(“\n Ожидалось равно”);break;
                default:
```

```
            printf("\n Неизвестная входная литера");
                    }
N_sost=0; c= getc( ); }
else c= getc( );
            } while (c!=EOF);
```

Рисунок 1.4 – Эскиз программы сканера

В приведенном фрагменте обозначено: `N_sost` – переменная, содержащая текущий номер состояния конечного автомата или код ошибки или лексемы; `c` – переменная, содержащая текущую литеру на входе конечного автомата; `Prog_code` – управляющая переменная файла, принимающего дескрипторы лексем.

Отметим, что в учебной программе отсутствуют компоненты для формирования таблиц имен и констант, поиска в этих таблицах, а сам дескриптор представлен только кодом класса лексем без ссылки на значение.

Разработанные в ходе проектирования тестовые последовательности литер приведены в таблице 1.4.

Таблица 1.4 – Правильные и ошибочные цепочки

Цепочка литер	*Результат обработки*
STA	Допущено, код 100
+	Допущено, код 501
=	Не допущено, код 801
1.178	Допущено 1.17, код 400. Не допущено 8, код 801
:=	Допущено, код 600
E+STE “пробел”	Допущено, коды 300, 501, 300
SST-	Не допущено SST, код 803. Допущен – , код 502
STO*ST:=	Допущено, код 200, 503, ошибка 802, допущен код 600

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Алгоритмический язык АЛГОЛ 60. Модифицированное сообщение. – М.: Мир, 1982. – 72 с.

2. Алкок Д. Язык Паскаль в иллюстрациях. / Д. Алкок. – М.: Мир, 1991. – 72 с.

3. Ахо А. Теория синтаксического анализа, перевода и компиляции. Синтаксический анализ./ А. Ахо, Дж. Ульман. – М.: Мир, 1978. – 619 с.

4. Ахо А. Компиляторы: принципы, технологии и инструменты/ А. Ахо, Р. Сети, Дж. Ульман. . – М.: Издательский дом «Вильямс», 2002. – 412 с.

5. Вайнгартен Ф. Трансляция языков программирования / Ф. Вайнгартен - М.: Мир, 1977.-190 с.

6. Вирт Н. Алгоритмы и структуры данных / Н. Вирт. - М.: Мир, 1989. - 360 с.

7. Грис Д. Конструирование компиляторов для ЦВМ / Д. Грис - М.: Мир, 1975.-544 с.

8. Льюис Ф. Теоретические основы проектирования компиляторов. / Ф. Льис, Д. Розенкранц, Р. Стирнз. –М.: Мир, 1979. – 564 с.

9. Методические указания к выполнению лабораторных и контрольных работ по дисциплине “Теоретические основы построения компиляторов” для студентов всех форм обучения основного профиля 09.03.02 – “Информационные системы и технологии”. Основные алгоритмы [Текст] / Разраб. В.Ю. Карлусов. – Севастополь: Изд-во СевГУ, 2018. – 44 с.

10. Рейуорд-Смит В.Дж. Теория формальных языков. Вводный курс / В.Дж. Рейуорд-Смит- М.: Мир, 1986.-128 с.

11. Хантер Р. Основные концепции компиляторов. / Р.Хантер. – М.: Издательский дом «Вильямс», 2002. – 256 с.

12. Хопкрофт Дж. Введение в теорию автоматов, языков и вычислений. / Дж. Хопркрофт, Р. Мотвани, Дж. Улман. – М.: Издательский дом «Вильямс», 2002. – 528 с.