МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
**"Южно-Уральский государственный университет
(национальный исследовательский университет)"**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ОТЧЕТ

по учебной практике

бакалавра направления 02.03.02 "Фундаментальная информатика
и информационные технологии"

Выполнил:_____
студент группы КЭ-101
М.И. Скоблюк

Проверил:_____
Ст. преподаватель кафедры СП
Н.С. Силкина
Дата:_____, Оценка:_____

Челябинск-2021

Министерство науки и высшего образования Российской Федерации
Южно-Уральский государственный университет
Кафедра системного программирования

УТВЕРЖДАЮ

Зав. кафедрой
системного программирования

_____ Л.Б. Соколинский

ЗАДАНИЕ
по учебной практике

Цель работы

Необходимо разработать распознаватель заданной символьной цепочки. Символьная цепочка задается с помощью формул Бэкуса-Наура.

Исходные данные к работе

1. *Йенсен К., Вирт Н.* Паскаль. Руководство пользователя и описание языка. М.: Компьютер, 1995.
2. *Льюис Ф., Розенкранц Д., Стирнз Р.* Теоретические основы проектирования компиляторов. М.: Мир, 1979.

Перечень подлежащих разработке вопросов

1. Выполнить анализ требований и разработать спецификации программы.
2. Провести проектирование программы.
3. Реализовать спроектированные модули.
4. Провести тестирование и отладку реализованных модулей.

Сроки

Дата выдачи задания: "28" июня 2021 г.
Срок сдачи законченной работы: "23" июля 2021 г.

Руководитель:

Ст. преподаватель кафедры СП _____ Силкина Н.С.
подпись

Задание принял к исполнению:

_____ Скоблюк М.И.
подпись

ОГЛАВЛЕНИЕ

1. Спецификация

Вариант С:

<цепочка>::=<описание переменных>

<описание переменных>::=**VAR** <список идентификаторов>:<тип>;

<список идентификаторов>::=<идентификатор> | <идентификатор>,<список идентификаторов>

<идентификатор>::= <буква> | <идентификатор><буква> |

<идентификатор><цифра>

<буква>::=A | B | C | D | E | F | ... | Z

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

<тип>::=**array** [<нижняя граница>..<верхняя граница>] **of** <стандартный тип>

<нижняя граница>::=<идентификатор> | <целая константа>

<целая константа>::=<целое со знаком> | <целое без знака>

<целое со знаком>::=<знак><целое без знака>

<знак>::=+ | -

<целое без знака>::=<цифра> | <цифра><целое без знака>

<верхняя граница>::=<нижняя граница>

<стандартный тип>::=**BOOLEAN** | **BYTE** | **CHAR** | **INTEGER** | **LONGINT** | **REAL** | **STRING** | **WORD**

Семантические ограничения:

Помимо этого, на цепочку накладывается следующее семантическое ограничение: идентификатор, входящий в цепочку, не должен совпадать с ключевыми словами языка Pascal. Регистр входных данных имеет значение.

Описание входных данных:

Цепочка записана в текстовом файле INPUT.txt, который состоит из одной строки. Длина цепочки не превышает 80 символов.

Описание выходных данных:

Результат распознавания необходимо записать в текстовый файл OUTPUT.txt одно из следующих сообщений: **ACCEPT**, если цепочка допустима, и **REJECT**, если цепочка недопустима.

Таблица 1. Примеры входных и выходных данных

INPUT.txt	OUTPUT.txt
`var arr: array[+1..N] of Real;`	ACCEPT
`var a1, a2, a3: array[-1..10] of Integer;`	ACCEPT
`var Inp, Out: array [Min..Max] of Real;`	ACCEPT
`var arr: array[1..N] of Real`	REJECT
`var a: array[-1..10] of Int;`	REJECT
`var a: array[-1..] of Integer;`	REJECT
`var a: array[+-1..10] of  Boolean;`	REJECT
`var arr:  arr[1..2]  of Real;`	REJECT
`var while : array [of..else] of Integer;`	REJECT
`var 123: array[1..3] of Char;`	REJECT
`var arr: array[-2.2..5] of True;`	REJECT
`var a1, a2: array[0..-1] of Integer;`	REJECT
`var a1, a2, a3: array[-1. .10] of Integer;`	REJECT

2. Проектирование

2.1. Модульная структура

Модульная структура представляет из себя программу, в которой содержатся шесть подпрограмм. Они же, в свою очередь, выполняют данные им задачи, которые описаны ниже.

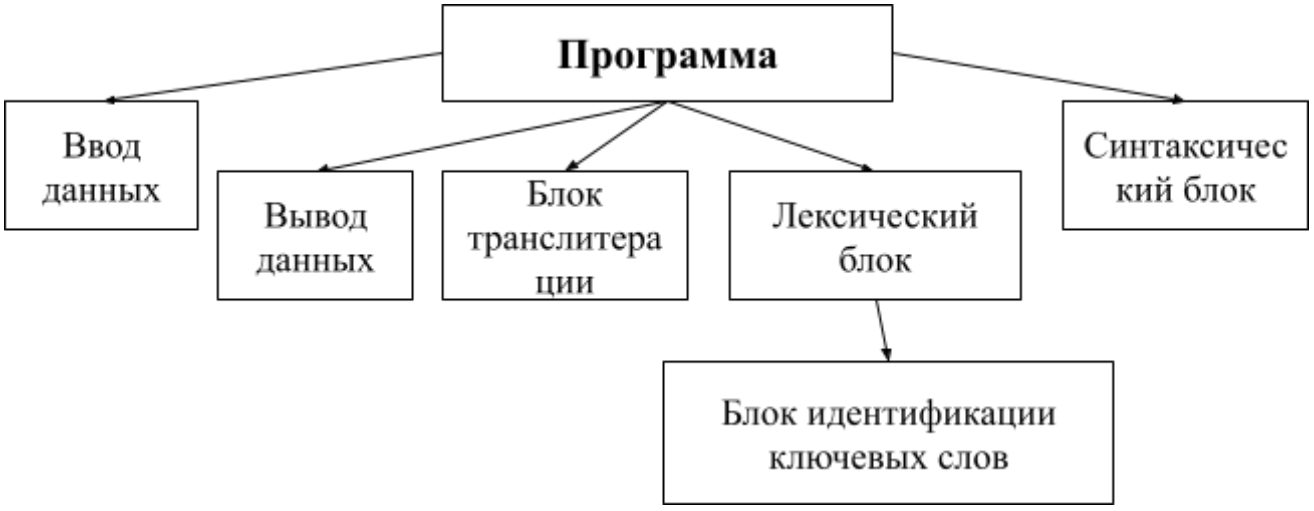

Рис. 1. Схема модульной структуры программы

Программа – основной блок, который содержит в себе нижеописанные подпрограммы.

Ввод данных – считывание строки из INPUT.txt в память.

Вывод данных – вывод результата в OUTPUT.txt.

Блок транслитерации – подпрограмма, преобразующая исходную символьную цепочку в цепочку лексем вида ("символ цепочки", "класс символа цепочки ").

Лексический блок – подпрограмма, преобразующая цепочку лексем, полученную от транслитератора, в цепочку лексем вида ("символ входного языка", "класс символа входного языка").

Блок идентификации ключевых слов – блок, устанавливающий присутствуют ли в двумерном списке лексем ключевые слова языка Паскаль.

Синтаксический блок – подпрограмма, которая получает цепочку лексем вида ("символ входного языка", "класс символа входного языка") и устанавливает, соответствует ли она заданным формулам Бэкуса-Наура.

2.2 Интерфейсы модулей

Программа:

Имя модуля: def main()

Входные параметры: -

Выходные параметры: -

Семантика: модуль, вызывающий остальные подпрограммы

Ввод данных:

Имя модуля: def input_file(file_name: str)

Входные параметры: file_name - путь к файлу

Выходные параметры: str - строка, содержащаяся в файле INPUT.txt

Семантика: модуль, считывающий и возвращающий строку из файла INPUT.txt

Блок транслитерации:

Имя модуля: def transliterator(string: str)

Входные данные: string - строка, полученная из файла INPUT.txt.

Выходные данные: flag - флаг, принимает значения True/False или 0/1, в зависимости от того, присутствуют ли в цепочке символы с неописанными классами(если встречается символ, класс которого не описан, работа модуля прекращается и флаг зануляется); transliteration_list - двумерный список размера n * 2 вида ["символ цепочки"] ["класс символа цепочки"], где n – длина исходной строки. Например, символьную цепочку “A1:= 5; ” блок транслитерации должен преобразовать в следующий двумерный массив лексем:

[‘A’][‘letter’]

[‘1’][‘digit’]

[‘:’][‘colon’]

[‘=’][‘equally’]

[‘ ’][‘space’]

[‘5’][‘digit’]

[‘;’][‘semicolon’]

(пробелы в начале и в конце опускаются)

Семантика: модуль, получающий строку из файла INPUT.txt и генерирующий по содержанию этой строки - двумерный список размера n * 2 вида ["символ цепочки"] ["класс символа цепочки"], где n – длина исходной строки

Лексический блок:

Имя модуля: def lexical(transliteration_list: list)

Входные данные: transliteration_list - – двумерный список, в котором находится строка, разбитая на символы с присвоенными им классами

Выходные данные: flag - флаг, устанавливающий правилен ли порядок символов цепочки; lexical_list - двумерный список размера n * 2

Семантика: модуль отвечает за преобразование двумерного списка лексем, полученного от блока транслитерации, в двумерный список лексем вида ["слово входного языка"] ["класс слова входного языка"]. В рассматриваемом примере лексический блок должен выдать следующую цепочку лексем:

[‘A1’][‘identifier’]

[‘:’][‘colon’]

[‘=’][‘equally’]

[‘ ’][‘space’]

[‘5’][‘digit’]

[‘;’][‘semicolon’]

Блок идентификации ключевых слов:

Имя модуля: def identification(lexical_list: list)

Входные данные: lexical_list – двумерный список, идентификаторы которого будут просматриваться на совпадения с ключевыми словами языка Pascal

Выходные данные: identified_list - двумерный список, идентификаторы, классы которых были изменены в случае совпадения с ключевыми словами языка Pascal

Семантика: модуль устанавливает, присутствуют ли в двумерном списке лексем ключевые слова языка Pascal

Синтаксический блок:

Имя модуля: def syntax(identified_list: list)

Входные данные: identified_list - двумерный список, содержащий лексемы и их классы.

Выходные данные: flag - флаг, показывающий соответствует ли цепочка заданным формулам Бэкуса-Наура

Семантика: модуль устанавливает, соответствует ли двумерный список заданным формулам Бэкуса-Наура. Данный блок использует для работы только классы символов входного языка. В рассматриваемом примере синтаксический блок, рассматривая цепочку вида должен сообщить, что она синтаксически правильна

Вывод данных:

Имя модуля: def output_file(file_name: str, answer: str)

Входные данные: answer - строка, от которой будет зависеть, что выведется в файл OUTPUT.txt: ACCEPT или REJECT; file_name - путь к файлу

Выходные данные: -

Семантика: модуль, получающий на вход путь к файлу и строку, от которой зависит, что выведется в файл OUTPUT.txt

3. Кодирование

3.1. Структура текста программы

Схема структуры программы приведена для языка программирования Python.

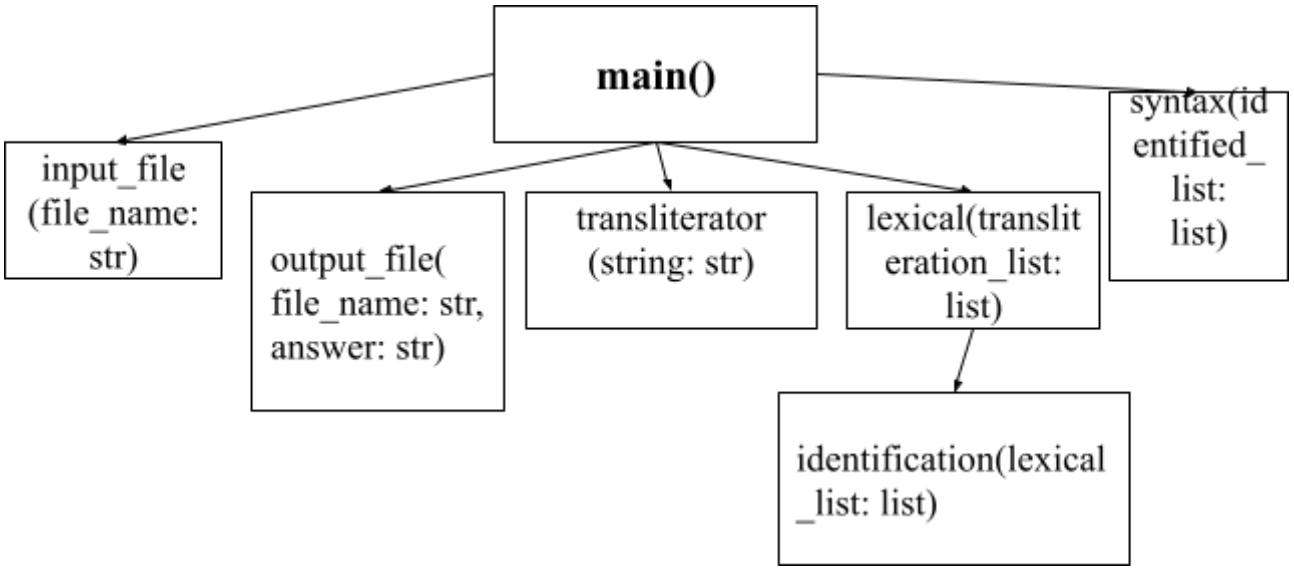

Рис. 2. Схема структуры текста программы

Описание структуры текста программы:

def main() - главный модуль программы.

def input_file(file_name: str) - модуль, реализующий ввод.

def output_file(file_name: str, answer: str) - модуль, реализующий вывод.

def transliterator(string: str) - модуль, реализующий транслитерацию.

def lexical(transliteration_list: list) - модуль, реализующий лексический блок.

def identification(lexical_list: list) - модуль, по поиску ключевых слов.

def syntax(identified_list: list) - модуль, реализующий синтаксический блок.

3.2. Алгоритмы реализации модулей

3.2.1. Блок транслитерации

Таблица 2. Транслитерация символьной цепочки

Символы	Класс лексемы
a-z, A-Z, ”_”	letter
0..9	digit
:	colon
;	semicolon
+-	sign
[	square_bracket_start
]	square_bracket_end
‘ ‘	space
.	dot
,	comma
<другие символы>	other

3.2.2. Лексический блок

Построение обрабатывающего автомата начнем с построения “аналогичного” ему конечного распознавателя. В начале определим все возможные состояния распознавателя и их семантику в Таблице 3.

Таблица 3. Состояния конечного распознавателя лексического блока

№ п/п	Состояние	Семантика
1	start	Момент до начала обработки цепочки.
2	k_word_var	Чтение первого ключевого слова var.
3	space1	Чтение пробелов между ключевым словом var и идентификатором.
4	identifier	Чтение идентификатора.

№ п/п	Состояние	Семантика
5	space2	Чтение пробелов между идентификатором и двоеточием “:”.
6	space3	Чтение пробелов между двоеточием “:” и ключевым словом array.
7	k_word_array	Чтение второго ключевого слова array.
8	space4	Чтение пробелов между ключевым словом array и открывающейся квадратной скобкой “[“.
9	space5	Чтение пробелов между открывающейся квадратной скобкой “[“ и нижней границей.
10	identifier_lower	Чтение идентификатора нижней границы(если указан идентификатор).
11	space_lower	Чтение пробелов между знаком числа нижней границы(если он есть) и самим числом(если указано число)
12	number_lower	Чтение числа нижней границы(если указано число).
13	space6	Чтение пробелов между нижней границей и двумя точками “..”.
14	dots	Чтение двух точек “..”.
15	space7	Чтение пробелов между двумя точками “..” и верхней границей.
16	identifier_upper	Чтение идентификатора верхней границы(если указан идентификатор).
17	space_upper	Чтение пробелов между знаком числа верхней границы(если он есть) и самим числом(если указано число)
18	number_upper	Чтение числа верхней границы(если указано число).
19	space8	Чтение пробелов между верхней границей и закрывающейся квадратной скобкой “]“.
20	space9	Чтение пробелов между закрывающейся квадратной скобкой “]“ и ключевым словом of.

№ п/п	Состояние	Семантика
21	k_word_of	Чтение ключевого слова of.
22	space10	Чтение пробелов между ключевым словом of и ключевым словом Тип данных.
23	k_word_type_of_data	Чтение ключевого слова Тип данных.
24	space11	Чтение пробелов между ключевым словом Тип данных и знаком точка с запятой “;”.
25	semicolon	Чтение знака точка с запятой “;”.
26	E	Ошибка.

Начальное состояние **start**. Допустимое состояния – **semicolon**. Конечный распознаватель, построенный на основе полученных состояний, приведен в Таблице 4.

Таблица 4. Конечный распознаватель лексического блока

	letter	**digit**	**space**	**sign**	**“,”**	**“.”**	**“:”**	**“[“**	**“]”**	**“;”**	
start	k_word_var	E	E	E	E	E	E	E	E	E	0
k_word_var	k_word_var	E	space1	E	E	E	E	E	E	E	0
space1	identifier	E	space1	E	E	E	E	E	E	E	0
identifier	identifier	identifier	space2	E	space1	E	E	E	E	E	0
space2	E	E	space2	E	E	E	space3	E	E	E	0
space3	k_word_array	E	space3	E	E	E	E	E	E	E	0
k_word_array	k_word_array	E	space4	E	E	E	E	space5	E	E	0
space4	E	E	space4	E	E	E	E	space5	E	E	0

	letter	digit	space	sign	“,”	“.”	“:”	“[“	“]”	“;”	
space5	identifier_lower	number_lower	space5	space_lower	E	E	E	E	E	E	0
identifier_lower	identifier_lower	identifier_lower	space6	E	E	dots	E	E	E	E	0
space_lower	E	number_lower	space_lower	E	E	E	E	E	E	E	0
number_lower	E	number_lower	space6	E	E	dots	E	E	E	E	0
space6	E	E	space6	E	E	dots	E	E	E	E	0
dots	E	E	E	E	E	space7	E	E	E	E	0
space7	identifier_upper	number_upper	space7	space_upper	E	E	E	E	E	E	0
identifier_upper	identifier_upper	identifier_upper	space8	E	E	E	E	E	space9	E	0
space_upper	E	number_upper	space_upper	E	E	E	E	E	E	E	0
number_upper	E	number_upper	space8	E	E	E	E	E	space9	E	0
space8	E	E	space8	E	E	E	E	E	space9	E	0
space9	k_word_of	E	space9	E	E	E	E	E	E	E	0
k_word_of	space10	E	space10	E	E	E	E	E	E	E	0

	letter	digit	space	sign	“,”	“.”	“:”	“[“	“]”	“;”	
space10	k_word_type_of_data	E	space10	E	E	E	E	E	E	E	0
k_word_type_of_data	k_word_type_of_data	E	space11	E	E	E	E	E	E	semicolon	0
space11	E	E	space11	E	E	E	E	E	E	semicolon	0
semicolon	E	E	E	E	E	E	E	E	E	E	1
E	E	E	E	E	E	E	E	E	E	E	0

В полученном конечном распознавателе нет недостижимых состояний. Эквивалентных состояний нет. Дальнейший процесс редукции конечного распознавателя лексического блока приведен в Таблице 5.

Таблица 5. Редукция конечного распознавателя лексического блока

Шаг	Результат(блоки состояний)	Действия
0	P0={start, k_word_var, space1, identifier, space2, space3, k_word_array, space4, space5, identifier_lower, space_lower, number_lower, space6, dots, space7, identifier_upper, space_upper, number_upper, space8, space9, k_word_of, space10, k_word_type_of_data, space11, semicolon, E}	Разбиваем P0 на два блока: допустимые и отвергающие состояния
1	$P1_1$={start, k_word_var, space1, identifier, space2, space3, k_word_array, space4, space5, identifier_lower, space_lower, number_lower, space6, dots, space7, identifier_upper, space_upper, number_upper, space8, space9, k_word_of, space10, k_word_type_of_data, space11, E} $P1_2$={semicolon}	Разбиваем $P1_1$ по входу letter

Шаг	Результат(блоки состояний)	Действия
2	$P2_1$={start, k_word_var}, $P2_2$={space1, identifier}, $P2_3$={space3, k_word_array}, $P2_4$={space5, identifier_lower}, $P2_5$={space7, identifier_upper}, $P2_6$={space9}, $P2_7$={k_word_of}, $P2_8$={space10, k_word_type_of_data}, $P2_9$={space2, space4, space_lower, number_lower, space6, dots, space_upper, number_upper, space8, space11, E}, $P2_{10}$={semicolon}	Разбиваем $P2_1$ по входу space, Разбиваем $P2_2$ по входу digit, Разбиваем $P2_3$ по входу “[“, Разбиваем $P2_4$ по входу digit, Разбиваем $P2_5$ по входу digit, Разбиваем $P2_8$ по входу “;”, Разбиваем $P2_9$ по входу “.”
3	$P3_1$={start}, $P3_2$={k_word_var}, $P3_3$={space1}, $P3_4$={identifier}, $P3_5$={space3}, $P3_6$={k_word_array}, $P3_7$={space5}, $P3_8$={identifier_lower}, $P3_9$={space7}, $P3_{10}$={identifier_upper}, $P3_{11}$={space9}, $P3_{12}$={k_word_of}, $P3_{13}$={space10}, $P3_{14}$={k_word_type_of_data}, $P3_{15}$={space2}, $P3_{16}$={space4, space_lower, number_lower, space6, dots, space_upper, number_upper, space8, space11, E}, $P3_{17}$={semicolon}	Разбиваем $P3_{16}$ по входу “[”

Шаг	Результат(блоки состояний)	Действия
4	$P4_1$={start}, $P4_2$={k_word_var}, $P4_3$={space1}, $P4_4$={identifier}, $P4_5$={space3}, $P4_6$={k_word_array}, $P4_7$={space5}, $P4_8$={identifier_lower}, $P4_9$={space7}, $P4_{10}$={identifier_upper}, $P4_{11}$={space9}, $P4_{12}$={k_word_of}, $P4_{13}$={space10}, $P4_{14}$={k_word_type_of_data}, $P4_{15}$={space2}, $P4_{16}$={space4}, $P4_{17}$={space_lower, number_lower, space6, dots, space_upper, number_upper, space8, space11, E}, $P4_{18}$={semicolon}	Разбиваем $P4_{17}$ по входу dots
5	$P5_1$={start}, $P5_2$={k_word_var}, $P5_3$={space1}, $P5_4$={identifier}, $P5_5$={space3}, $P5_6$={k_word_array}, $P5_7$={space5}, $P5_8$={identifier_lower}, $P5_9$={space7}, $P5_{10}$={identifier_upper}, $P5_{11}$={space9}, $P5_{12}$={k_word_of}, $P5_{13}$={space10}, $P5_{14}$={k_word_type_of_data}, $P5_{15}$={space2}, $P5_{16}$={space4}, $P5_{17}$={number_lower, space6}, $P5_{18}$={dots}, $P5_{19}$={space_lower, space_upper, number_upper, space8, space11, E}, $P5_{20}$={semicolon}	Разбиваем $P5_{17}$ по входу digit, Разбиваем $P5_{19}$ по входу “:”

Шаг	Результат(блоки состояний)	Действия
6	$P6_1$={start}, $P6_2$={k_word_var}, $P6_3$={space1}, $P6_4$={identifier}, $P6_5$={space3}, $P6_6$={k_word_array}, $P6_7$={space5}, $P6_8$={identifier_lower}, $P6_9$={space7}, $P6_{10}$={identifier_upper}, $P6_{11}$={space9}, $P6_{12}$={k_word_of}, $P6_{13}$={space10}, $P6_{14}$={k_word_type_of_data}, $P6_{15}$={space2}, $P6_{16}$={space4}, $P6_{17}$={number_lower}, $P6_{18}$={space6}, $P6_{19}$={dots}, $P6_{20}$={space11}, $P6_{21}$={space_lower, space_upper, number_upper, space8, E}, $P6_{22}$={semicolon}	Разбиваем $P6_{21}$ по входу digit
7	$P7_1$={start}, $P7_2$={k_word_var}, $P7_3$={space1}, $P7_4$={identifier}, $P7_5$={space3}, $P7_6$={k_word_array}, $P7_7$={space5}, $P7_8$={identifier_lower}, $P7_9$={space7}, $P7_{10}$={identifier_upper}, $P7_{11}$={space9}, $P7_{12}$={k_word_of}, $P7_{13}$={space10}, $P7_{14}$={k_word_type_of_data}, $P7_{15}$={space2}, $P7_{16}$={space4}, $P7_{17}$={number_lower}, $P7_{18}$={space6}, $P7_{19}$={dots}, $P7_{20}$={space11}, $P7_{21}$={space_lower, space_upper, number_upper}, $P7_{22}$={space8, E}, $P7_{23}$={semicolon}	Разбиваем $P7_{21}$ по входу space Разбиваем $P7_{22}$ по входу “]”

Шаг	Результат(блоки состояний)	Действия
8	$P8_1$={start}, $P8_2$={k_word_var}, $P8_3$={space1}, $P8_4$={identifier}, $P8_5$={space3}, $P8_6$={k_word_array}, $P8_7$={space5}, $P8_8$={identifier_lower}, $P8_9$={space7}, $P8_{10}$={identifier_upper}, $P8_{11}$={space9}, $P8_{12}$={k_word_of}, $P8_{13}$={space10}, $P8_{14}$={k_word_type_of_data}, $P8_{15}$={space2}, $P8_{16}$={space4}, $P8_{17}$={number_lower}, $P8_{18}$={space6}, $P8_{19}$={dots}, $P8_{20}$={space11}, $P8_{21}$={space_lower}, $P8_{22}$={space_upper}, $P8_{23}$={number_upper}, $P8_{24}$={space8}, $P8_{25}$={E}, $P8_{26}$={semicolon}	Дальнейшее разбиение невозможно

В Таблице 6 приведен результат редукции конечного распознавателя лексического блока.

Таблица 6. Минимальный конечный распознаватель лексического блока

	letter	**digit**	**space**	**sign**	**“,”**	**“.”**	**“:”**	**“[“**	**“]”**	**“;”**	
start	k_word_var	E	E	E	E	E	E	E	E	E	0
k_word_var	k_word_var	E	space1	E	E	E	E	E	E	E	0
space1	identifier	E	space1	E	E	E	E	E	E	E	0

	letter	digit	space	sign	“,”	“.”	“:”	“[“	“]”	“;”	
identifier	identifier	identifier	space2	E	space1	E	E	E	E	E	0
space2	E	E	space2	E	E	E	space3	E	E	E	0
space3	k_word_array	E	space3	E	E	E	E	E	E	E	0
k_word_array	k_word_array	E	space4	E	E	E	E	space5	E	E	0
space4	E	E	space4	E	E	E	E	space5	E	E	0
space5	identifier_lower	number_lower	space5	space_lower	E	E	E	E	E	E	0
identifier_lower	identifier_lower	identifier_lower	space6	E	E	dots	E	E	E	E	0
space_lower	E	number_lower	space_lower	E	E	E	E	E	E	E	0
number_lower	E	number_lower	space6	E	E	dots	E	E	E	E	0
space6	E	E	space6	E	E	dots	E	E	E	E	0
dots	E	E	E	E	E	space7	E	E	E	E	0
space7	identifier_upper	number_upper	space7	space_upper	E	E	E	E	E	E	0
identifier_upper	identifier_upper	identifier_upper	space8	E	E	E	E	E	space9	E	0
space_upper	E	number_upper	space_upper	E	E	E	E	E	E	E	0

	letter	digit	space	sign	“,”	“.”	“:”	“[“	“]”	“;”	
number_upper	E	number_upper	space8	E	E	E	E	E	space9	E	0
space8	E	E	space8	E	E	E	E	E	space9	E	0
space9	k_word_of	E	space9	E	E	E	E	E	E	E	0
k_word_of	space10	E	space10	E	E	E	E	E	E	E	0
space10	k_word_type_of_data	E	space10	E	E	E	E	E	E	E	0
k_word_type_of_data	k_word_type_of_data	E	space11	E	E	E	E	E	E	semicolon	0
space11	E	E	space11	E	E	E	E	E	E	semicolon	0
semicolon	E	E	E	E	E	E	E	E	E	E	1
E	E	E	E	E	E	E	E	E	E	E	0

По полученному конечному распознавателю составим обрабатывающий автомат. Для этого введем примитивные процедуры, описанные в Таблице 7.

Таблица 7. Примитивные процедуры обрабатывающего автомата лексического блока

№ п/п	Состояние	Семантика
1	Да	Остановить обработку и допустить цепочку.
2	Нет	Остановить обработку и отвергнуть цепочку
3	Обработать	Добавить входной символ к значению текущей лексемы.
4	Лексема(класс)	Перейти к рассмотрению следующей лексемы с заданным классом.

Для удобства определим процедуры переходов, которые будем обозначать цифрами. Данные процедуры приведены в Таблице 8.

Таблица 8. Процедуры переходов обрабатывающего автомата лексического блока

Действие	**Семантика**
1	Обработать
2	Лексема(identifier); Обработать;
3	Лексема(number); Обработать;
4	Лексема(dot); Обработать;
5	Лексема(square_bracket); Обработать;
6	Лексема(comma); Обработать;
7	Лексема(colon); Обработать;
8	Лексема(sign); Обработать;
9	Лексема(semicolon); Обработать;

На основе минимального конечного распознавателя составим соответствующий обрабатывающий автомат, который приведен в Таблице 9.

Таблица 9. Обрабатывающий автомат лексического блока

	letter	**digit**	**space**	**sign**	**“,”**	**“.”**	**“:”**	**“[“**	**“]”**	**“;”**	-|
start	**2** k_word _var										

	letter	**digit**	**space**	**sign**	**“,”**	**“.”**	**“:”**	**“[“**	**“]”**	**“;”**	-|
k_word_var	**1** k_word_var		space1								
space1	**2** identifier		space1								
identifier	**1** identifier	**1** identifier	space2		**6** space1		**7** space3				
space2			space2				**7** space3				
space3	**2** k_word_array		space3								
k_word_array	**1** k_word_array		space4					**5** space5			
space4			space4					**5** space5			
space5	**2** identifier_lower	**3** number_lower	space5	**8** space_lower							
identifier_lower	**1** identifier_lower	**1** identifier_lower	space6			**4** dots					
space_lower		**3** number_lower	space_lower								
number_lower		**3** number_lower	space6			**4** dots					

	letter	**digit**	**space**	**sign**	**“,”**	**“.”**	**“:”**	**“[“**	**“]”**	**“;”**	**-|**
space6			space6			**4** dots					
dots						**4** space7					
space7	**2** identifier_upper	**3** number_upper	space7	**8** space_upper							
identifier_upper	**1** identifier_upper	**1** identifier_upper	space8						**5** space9		
space_upper		**3** number_upper	space_upper								
number_upper		**3** number_upper	space8						**5** space9		
space8			space8						**5** space9		
space9	**2** k_word_of		space9								
k_word_of	**1** space10		space10								
space10	**2** k_word_type_of_data		space10								
k_word_type_of_data	**1** k_word_type_of_data		space11							**9** semicolon	
space11			space11							**9** semicolon	
semicolon											Yes

Пустым клеткам соответствует вызов примитивной процедуры No.

3.2.3. Синтаксический блок

Конечный распознаватель синтаксического блока рассматриваемого приведен в Таблице 10. Пустым клеткам соответствует вызов примитивной процедуры No, которая отвергает цепочку.

Таблица 10. Конечный распознаватель синтаксического блока

	k_word_var	**identifier**	**comma**	**colon**	**k_word_array**	**‘[‘**	**sign**	**number**	**dot**	**‘]’**	**k_word_of**	**k_word_type**	**semicolon**
start	var												
var		name											
name			name	colon									
colon					array								
array						square_bracket_start							
square_bracket_start		name_lower					num_lower	num_lower					
name_lower									dots				
num_lower									dots				
dots		name_upper					num_upper	num_upper					
name_upper										square_bracket_end			

	k_word_var	identifier	comma	colon	k_word_array	‘[‘	sign	number	dot	‘]’	k_word_of	k_word_type	semicolon
num_upper										square_bracket_end			
square_bracket_end											of		
of												type	
type													semicolon
semicolon													1

Очевидно, что полученный автомат не подлежит дальнейшей редукции и является минимальным.

3.2.4. Блок идентификации ключевых слов

Реализация функции ***def*** **identification(lexical_list: list)** с помощью линейного поиска(проход по массиву пар лексема-класс и проверка на вхождение в массив ключевых слов языка Pascal, в случае вхождения изменяется класс лексемы). Описание алгоритма линейного поиска: https://kvodo.ru/lineynyiy-poisk.html.

3.3. Размер текста программы (в строках)

Размер текста программы занимает ~500 строк.

4. Тестирование

4.1. Автономное тестирование

Протоколы тестирования для блоков транслитерации, лексического блока, блока идентификации ключевых слов и синтаксического блока приведены соответственно в Таблицах 11, 12, 13 и 14.

Таблица 11. Протокол тестирования блока транслитерации

№ п/п	Входные данные	Выходные данные	Действительный результат	Тест прой ден?
1.	var arr :;	[‘v’, ‘letter’], [‘a’, ‘letter’], [‘r’, ‘letter’], [‘ ’, ‘space’], [‘ ’, ‘space’], [‘a’, ‘letter’], [‘r’, ‘letter’], [‘r’, ‘letter’], [‘ ’, ‘space’], [‘:’, ‘colon’], [‘;’, ‘semicolon’]	[‘v’, ‘letter’], [‘a’, ‘letter’], [‘r’, ‘letter’], [‘ ’, ‘space’], [‘ ’, ‘space’], [‘a’, ‘letter’], [‘r’, ‘letter’], [‘r’, ‘letter’], [‘ ’, ‘space’], [‘:’, ‘colon’], [‘;’, ‘semicolon’]	Да
2.	a1, a2	[‘a’, ‘letter’], [‘1’, ‘digit’], [‘,’, ‘comma’], [‘ ’, ‘space’], [‘a’, ‘letter’], [‘2’, ‘digit’]	[‘a’, ‘letter’], [‘1’, ‘digit’], [‘,’, ‘comma’], [‘ ’, ‘space’], [‘a’, ‘letter’], [‘2’, ‘digit’]	Да
3.	array[-2..N]	['a', 'letter'], ['r', 'letter'], ['r', 'letter'], ['a', 'letter'], ['y', 'letter'], ['[', 'square_bracket_start'], ['-', 'sign'], ['2', 'digit'], ['.', 'dot'], ['.', 'dot'], ['N', 'letter'], [']', 'square_bracket_end']	['a', 'letter'], ['r', 'letter'], ['r', 'letter'], ['a', 'letter'], ['y', 'letter'], ['[', 'square_bracket_start'], ['-', 'sign'], ['2', 'digit'], ['.', 'dot'], ['.', 'dot'], ['N', 'letter'], [']', 'square_bracket_end']	Да
4.	[Min .@. Max] of ;	['[', 'square_bracket_start'], ['M', 'letter'], ['i', 'letter'], ['n', 'letter'], [' ', 'space'], ['.', 'dot'], ['@', 'other']	['[', 'square_bracket_start'], ['M', 'letter'], ['i', 'letter'], ['n', 'letter'], [' ', 'space'], ['.', 'dot'], ['@', 'other']	Да
5.	of Char1;	['o', 'letter'], ['f', 'letter'], [' ', 'space'], ['C', 'letter'], ['h', 'letter'], ['a', 'letter'], ['r', 'letter'], ['1', 'digit'], [';', 'semicolon']	['o', 'letter'], ['f', 'letter'], [' ', 'space'], ['C', 'letter'], ['h', 'letter'], ['a', 'letter'], ['r', 'letter'], ['1', 'digit'], [';', 'semicolon']	Да

Таблица 12. Протокол тестирования лексического блока

№ п/п	Входные данные	Выходные данные	Действительный результат	Тест прой ден?
1.	[['v', 'letter'], ['a', 'letter'], ['r', 'letter'], [' ', 'space'], ['a', 'letter'], ['r', 'letter'], ['r', 'letter'], [':', 'colon'], [' ', 'space'], ['a', 'letter'], ['r', 'letter'], ['r', 'letter'], ['a', 'letter'], ['y', 'letter'], ['[', 'square_bracket_start'], ['+', 'sign'], ['1', 'digit'], ['.', 'dot'], ['.', 'dot'], ['N', 'letter'], [']', 'square_bracket_end'], [' ', 'space'], ['o', 'letter'], ['f', 'letter'], [' ', 'space'], ['R', 'letter'], ['e', 'letter'], ['a', 'letter'], ['l', 'letter'], [';', 'semicolon']]	[('var', 'k_word_var'), ('arr', 'identifier'), (':', 'colon'), ('array', 'k_word_array'), ('[', 'square_bracket_start'), ('+', 'sign'), ('1', 'number_lower'), ('..', 'dots'), ('N', 'identifier_upper'), (']', 'square_bracket_end'), ('of', 'k_word_of'), ('Real', 'k_word_type_of_data'), (';', 'semicolon')]	[('var', 'k_word_var'), ('arr', 'identifier'), (':', 'colon'), ('array', 'k_word_array'), ('[', 'square_bracket_start'), ('+', 'sign'), ('1', 'number_lower'), ('..', 'dots'), ('N', 'identifier_upper'), (']', 'square_bracket_end'), ('of', 'k_word_of'), ('Real', 'k_word_type_of_data'), (';', 'semicolon')]	Да
2.	[['v', 'letter'], ['a', 'letter'], ['r', 'letter'], [' ', 'space'], ['a', 'letter'], ['1', 'digit'], [',', 'comma'], [' ', 'space'], ['a', 'letter'], ['2', 'digit'], [':', 'colon'], [' ', 'space'], ['a', 'letter'], ['r', 'letter'], ['r', 'letter'], ['a', 'letter'], ['y', 'letter'], ['[', 'square_bracket_start'], ['m', 'letter'], ['.', 'dot'], ['.', 'dot'], ['5', 'digit'], [']', 'square_bracket_end'], [' ', 'space'], ['o', 'letter'], ['f', 'letter'], [' ', 'space'], ['i', 'letter'], ['n', 'letter'], ['t', 'letter'], ['e', 'letter'], ['g', 'letter'], ['e', 'letter'], ['r', 'letter'], [';', 'semicolon']]	[('var', 'k_word_var'), ('a1', 'identifier'), (',', 'comma'), ('a2', 'identifier'), (':', 'colon'), ('array', 'k_word_array'), ('[', 'square_bracket_start'), ('m', 'identifier_lower'), ('..', 'dots'), ('5', 'number_upper'), (']', 'square_bracket_end'), ('of', 'k_word_of'), ('integer', 'k_word_type_of_data'), (';', 'semicolon')]	[('var', 'k_word_var'), ('a1', 'identifier'), (',', 'comma'), ('a2', 'identifier'), (':', 'colon'), ('array', 'k_word_array'), ('[', 'square_bracket_start'), ('m', 'identifier_lower'), ('..', 'dots'), ('5', 'number_upper'), (']', 'square_bracket_end'), ('of', 'k_word_of'), ('integer', 'k_word_type_of_data'), (';', 'semicolon')]	Да

№ п/п	Входные данные	Выходные данные	Действительный результат	Тест прой ден?
3.	[['v', 'letter'], ['a', 'letter'], ['r', 'letter'], [' ', 'space'], ['I', 'letter'], ['n', 'letter'], ['p', 'letter'], [',', 'comma'], [' ', 'space'], ['O', 'letter'], ['u', 'letter'], ['t', 'letter'], [':', 'colon'], [' ', 'space'], ['a', 'letter'], ['r', 'letter'], ['r', 'letter'], ['a', 'letter'], ['y', 'letter'], [' ', 'space'], ['[', 'square_bracket_start'], ['M', 'letter'], ['i', 'letter'], ['n', 'letter'], [' ', 'space'], ['.', 'dot'], ['.', 'dot'], ['M', 'letter'], ['a', 'letter'], ['x', 'letter'], [']', 'square_bracket_end'], [' ', 'space'], ['o', 'letter'], ['f', 'letter'], [' ', 'space'], ['R', 'letter'], ['e', 'letter'], ['a', 'letter'], ['l', 'letter'], [';', 'semicolon']]	[('var', 'k_word_var'), ('Inp', 'identifier'), (',', 'comma'), ('Out', 'identifier'), (':', 'colon'), ('array', 'k_word_array'), ('[', 'square_bracket_start'), ('Min', 'identifier_lower'), ('..', 'dots'), ('Max', 'identifier_upper'), (']', 'square_bracket_end'), ('of', 'k_word_of'), ('Real', 'k_word_type_of_data'), (';', 'semicolon')]	[('var', 'k_word_var'), ('Inp', 'identifier'), (',', 'comma'), ('Out', 'identifier'), (':', 'colon'), ('array', 'k_word_array'), ('[', 'square_bracket_start'), ('Min', 'identifier_lower'), ('..', 'dots'), ('Max', 'identifier_upper'), (']', 'square_bracket_end'), ('of', 'k_word_of'), ('Real', 'k_word_type_of_data'), (';', 'semicolon')]	Да

Таблица 13. Протокол тестирования блока идентификации ключевых слов

№ п/п	Входные данные	Выходные данные	Действительный результат	Тест прой ден?
1.	[('var', 'k_word_var'), ('Inp', 'identifier'), (',', 'comma'), ('Out', 'identifier'), (':', 'colon'), ('array', 'k_word_array'), ('[', 'square_bracket_start'), ('Min', 'identifier_lower'), ('..', 'dots'), ('Max', 'identifier_upper'), (']', 'square_bracket_end'), ('of', 'k_word_of'), ('Real', 'k_word_type_of_data'), (';', 'semicolon')]	[('var', 'var'), ('Inp', 'identifier'), (',', 'comma'), ('Out', 'identifier'), (':', 'colon'), ('array', 'array'), ('[', 'square_bracket_start'), ('Min', 'identifier_lower'), ('..', 'dots'), ('Max', 'identifier_upper'), (']', 'square_bracket_end'), ('of', 'of'), ('Real', 'type'), (';', 'semicolon')]	[('var', 'var'), ('Inp', 'identifier'), (',', 'comma'), ('Out', 'identifier'), (':', 'colon'), ('array', 'array'), ('[', 'square_bracket_start'), ('Min', 'identifier_lower'), ('..', 'dots'), ('Max', 'identifier_upper'), (']', 'square_bracket_end'), ('of', 'of'), ('Real', 'type'), (';', 'semicolon')]	Да
2.	[('vap', 'k_word_var'), ('while', 'identifier'), (':', 'colon'), ('array', 'k_word_array'), ('[', 'square_bracket_start'), ('until', 'identifier_lower'), ('..', 'dots'), ('Max', 'identifier_upper'), (']', 'square_bracket_end'), ('in', 'k_word_of'), ('Int', 'k_word_type_of_data'), (';', 'semicolon')]	[('vap', 'k_word_var'), ('while', 'while'), (':', 'colon'), ('array', 'array'), ('[', 'square_bracket_start'), ('until', 'until'), ('..', 'dots'), ('Max', 'identifier_upper'), (']', 'square_bracket_end'), ('in', 'in'), ('Int', 'k_word_type_of_data'), (';', 'semicolon')]	[('vap', 'k_word_var'), ('while', 'while'), (':', 'colon'), ('array', 'array'), ('[', 'square_bracket_start'), ('until', 'until'), ('..', 'dots'), ('Max', 'identifier_upper'), (']', 'square_bracket_end'), ('in', 'in'), ('Int', 'k_word_type_of_data'), (';', 'semicolon')]	Да
3.	[('var', 'k_word_var'), ('a1', 'identifier'), (':', 'colon'), ('array', 'k_word_array'), ('[', 'square_bracket_start'), ('2', 'number_lower'), ('..', 'dots'), ('3', 'number_upper'), (']', 'square_bracket_end'), ('of', 'k_word_of'), ('Char', 'k_word_type_of_data'), (';', 'semicolon')]	[('var', 'var'), ('a1', 'identifier'), (':', 'colon'), ('array', 'array'), ('[', 'square_bracket_start'), ('2', 'number_lower'), ('..', 'dots'), ('3', 'number_upper'), (']', 'square_bracket_end'), ('of', 'of'), ('Char', 'type'), (';', 'semicolon')]	[('var', 'var'), ('a1', 'identifier'), (':', 'colon'), ('array', 'array'), ('[', 'square_bracket_start'), ('2', 'number_lower'), ('..', 'dots'), ('3', 'number_upper'), (']', 'square_bracket_end'), ('of', 'of'), ('Char', 'type'), (';', 'semicolon')]	Да

Таблица 14. Протокол тестирования синтаксического блока

№ п/п	Входные данные	Выходные данные	Действительный результат	Тест прой ден?
1.	[('var', 'var'), ('arr', 'identifier'), (':', 'colon'), ('array', 'array'), ('[', 'square_bracket_start'), ('-', 'sign'), ('1', 'number_lower'), ('..', 'dots'), ('0', 'number_upper'), (']', 'square_bracket_end'), ('of', 'of'), ('Byte', 'type'), (';', 'semicolon')]	True	True	Да
2.	[('var', 'var'), ('a1', 'identifier'), (',', 'comma'), ('a2', 'identifier'), (':', 'colon'), ('array', 'array'), ('[', 'square_bracket_start'), ('+', 'sign'), ('1', 'number_lower'), ('..', 'dots'), ('N', 'identifier_upper'), (']', 'square_bracket_end'), ('of', 'of'), ('Word', 'type'), (';', 'semicolon')]	True	True	Да
3.	[('var', 'var'), ('Inp', 'identifier'), (':', 'colon'), ('array', 'array'), ('[', 'square_bracket_start'), ('1', 'number_lower'), ('..', 'dots'), ('-', 'sign'), ('2', 'number_upper'), (']', 'square_bracket_end'), ('of', 'of'), ('Char', 'type'), (';', 'semicolon')]	False	False	Да

4.2. Комплексное тестирование

В Таблице 15 приведен протокол тестирования головного модуля программы.

Таблица 15. Протокол тестирования головного модуля программы

№ п/п	Входные данные	Выходные данные	Действительный результат	Тест пройден?
1.	var arr: array[1.. N] of Real;	ACCEPT	ACCEPT	Да
2.	var a1, a2, a3: array[-1..10] of Integer;	ACCEPT	ACCEPT	Да
3.	var Inp, Out: array [Min..Max] of Real;	ACCEPT	ACCEPT	Да
4.	var arr: array[1..N] of Real	REJECT	REJECT	Да
5.	var a: array[-1..10] of Int;	REJECT	REJECT	Да
6.	var a: array[-1..] of Integer;	REJECT	REJECT	Да
7.	var a: array[+-1..10] of Boolean;	REJECT	REJECT	Да
8.	var arr: arr[1..2] of Real;	REJECT	REJECT	Да
9.	var while : array [of..else] of Integer;	REJECT	REJECT	Да
10.	var 123: array[1..3] of Char;	REJECT	REJECT	Да
11.	var arr: array[-2.2..5] of True;	REJECT	REJECT	Да
12.	var a1, a2: array[0..-1] of Integer;	REJECT	REJECT	Да
13.	var a1, a2, a3: array[-1. .10] of Integer;	REJECT	REJECT	Да

5. Заключение

Данная работа посвящена разработке распознавателя символьной цепочки, заданной формулами Бэкуса-Наура. В ходе выполнения работы были пройдены все основные этапы разработки программного обеспечения: анализ, написание спецификации, проектирование, разработка алгоритмов, кодирование, тестирование и сопровождение. Каждому этапу, за исключением сопровождения, в данной работе выделен отдельный пункт. В данных пунктах описана проделанная в каждом случае работа и ее результаты.

Литература

1. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. - М.: Мир, 1979.

2. Йенсен К., Вирт Н. Паскаль. Руководство пользователя и описание языка. - М.: Компьютер, 1995.