

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
"Южно-Уральский государственный университет"
(национальный исследовательский университет)
Факультет Вычислительной математики и информатики
Кафедра системного программирования

УЧЕБНАЯ ПРАКТИКА
СТУДЕНТОВ НАПРАВЛЕНИЯ
010300.62 "ФУНДАМЕНТАЛЬНАЯ ИНФОРМАТИКА
И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ"

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Челябинск 2012

ОГЛАВЛЕНИЕ

Введение.....	3
1. Задание.....	3
2. Методические указания по выполнению задания	4
2.1 Анализ.....	4
2.2 Спецификация.....	4
2.3 Проектирование.....	4
2.3.1 Проектирование модульной структуры	5
2.3.2 Разработка тестов и тестовых программ.....	6
2.4 Кодирование.....	8
2.4.1 Разработка структуры текста программы	8
2.4.2 Разработка алгоритмов	9
2.4.3 Кодирование модулей.....	14
2.5 Тестирование.....	15
2.6 Сопровождение.....	17
2.7 Распределение времени.....	18
3. Указания по подготовке отчета о практике	18
4. Варианты заданий	20
4.1 Общие определения.....	20
4.2 Семантические ограничения	21
4.2.1 Варианты	21
Список рекомендуемой литературы.....	26

Введение

Целью учебной практики является практическое закрепление знаний, полученных при изучении курсов "Язык программирования" и "Основы программирования".

В ходе практики студент обязан *разработать программную систему* в соответствии с заданием и представить *отчет* о прохождении учебной практики.

Данный документ содержит указания по прохождению учебной практики и состоит из пяти разделов и списка литературы.

В *первом разделе* сформулировано *задание* по учебной практике.

Во *втором разделе* приведены *методические указания* по учебной практике, которые регламентируют процесс разработки программной системы, предусмотренной заданием.

В *третьем разделе* даны *указания по подготовке отчета* о прохождении учебной практики.

В *четвертом разделе* находятся *варианты задания* по учебной практике.

Методические указания завершаются *списком литературы*, рекомендуемой при выполнении задания по учебной практике.

1. Задание

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

Например:

```
<цепочка> ::= CONST <идентификатор> = <значение> ;
<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
<значение> ::= <целая константа> | <логическая константа>
<целая константа> ::= <целое со знаком> | <целое без знака>
<целое со знаком> ::= <знак><целое без знака>
<знак> ::= + | -
<целое без знака> ::= <цифра> | <целое без знака>
<логическая константа> ::= TRUE | FALSE
```

Помимо этого на цепочку накладывается следующее *семантическое ограничение*: идентификатор, входящий в цепочку, не должен совпадать с ключевыми словами языка Pascal [1].

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 в одно из следующих сообщений: ACCEPT, если цепочка допустима, и REJECT, если цепочка недопустима.

Примеры входных и выходных данных

INPUT . TXT	OUTPUT . TXT
const N=10;	ACCEPT
const Min=-10;	ACCEPT

INPUT . TXT	OUTPUT . TXT
const Max=+10;	ACCEPT
const On=True;	ACCEPT
const Off=FALSE;	ACCEPT
const MyMaxN = 123456 ;	ACCEPT
const N=10	REJECT
const Min=-+10;	REJECT
const Min=-True;	REJECT
const Off=Flolse;	REJECT
const Min=123.456;	REJECT
const N=10; const Min=-10;	REJECT
const BEGIN=123;	REJECT
const end=False;	REJECT

2. Методические указания по выполнению задания

Методические указания регламентируют процесс разработки программой системы, предусмотренной заданием.

Процесс разработки распознавателя цепочки необходимо разбить на этапы в соответствии с *технологическим циклом разработки* программного обеспечения: анализ, спецификация, проектирование, кодирование, тестирование и сопровождение [2; 5; 6]. Прохождение данных этапов необходимо *документировать*, то есть *одновременно* с выполнением работ очередного этапа готовить соответствующий раздел отчета по учебной практике.

Далее приведены *указания* по выполнению работ и *распределению времени* на каждом из перечисленных этапов технологического цикла.

2.1 Анализ

На этапе *анализа* разработчик выясняет у заказчика, *что* именно должна делать программная система (но не *как* она будет это делать). В контексте учебной практики на данном этапе необходимо ознакомиться со своим вариантом задания (см. раздел 3), и выяснить все неясные моменты задания у преподавателя, ведущего вычислительную практику.

2.2 Спецификация

На этапе *спецификации* разработчик подготавливает формальное описание функций будущей программной системы. В контексте учебной практики на данном этапе необходимо подготовить раздел отчета "Спецификация". В данном разделе отчета формулируется задание, дается определение символьной цепочки, распознаватель которой нужно разработать, описывается формат входных и выходных данных, и приводятся примеры входных и соответствующих им выходных данных.

2.3 Проектирование

На этапе *проектирования* необходимо выполнить проектирование модульной структуры программы и разработать набор тестов и соответствующие тестовые программы для проведения тестирования. Важной особенностью этапа проектирования является то, что все работы на данном этапе выполняются *без использования* системы программирования.

2.3.1 Проектирование модульной структуры

Модульная структура представляет собой иерархию процедур и функций (называемых *модулями*), с помощью которых программа решает поставленную задачу. При этом программа является *головным* модулем в данной иерархии. Пример модульной структуры некоторой программы приведен на Рис. 1.

Иерархия модулей отражает межмодульные связи *по управлению*, а не по времени. Это означает, что модульная структура отражает только тот факт, что модуль более высокого уровня иерархии вызывает все непосредственно связанные с ним модули более низкого уровня иерархии, не показывая при этом порядок вызова связанных модулей более низкого уровня иерархии.

Одними из признаков плохого проектирования модульной структуры являются [2]:

- наличие связей между модулями через уровень иерархии (например, от программы к модулю 2.2 на Рис. 1);
- наличие связей от модуля нижнего уровня иерархии к модулю верхнего уровня иерархии (например, от модуля 2.1 к модулю 1.2 или к программе на Рис. 1);
- наличие связей между модулями одного уровня иерархии (например, между модулями 1.1 и 1.2 на Рис. 1).

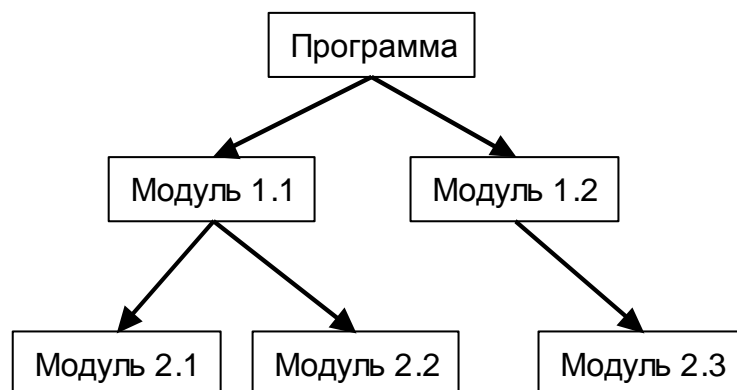


Рис. 1. Пример модульной структуры программы

После построения иерархии модулей программы необходимо выполнить следующие работы:

1. Разработать заголовки и спецификации модулей программы, а также соответствующие структуры данных. *Заголовок модуля* содержит имя модуля, а также имена и типы формальных параметров. *Спецификация модуля* содержит назначение модуля, описание формальных параметров модуля и их семантики (для модулей-функций также описание возвращаемого результата и его семантики) [2; 5; 6].
2. Разработать текст головного модуля программы. Головной модуль должен содержать корректные вызовы модулей первого уровня иерархии.

В ходе работ на данном этапе необходимо подготовить раздел отчета "Проектирование".

При проектировании модульной структуры распознавателя символьной цепочки следует использовать теорию методов трансляции [3]. В частности, модульная структура распознавателя должна содержать следующие модули:

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

Например, символьную цепочку
`const N=10;`

блок транслитерации должен преобразовать в цепочку лексем
(с, буква), (о, буква), (п, буква), (s, буква), (t, буква),
(' ', пробел), (N, буква), (=, равно), (1, цифра), (0, цифра),
(; , ТЧКЗПТ)

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

В рассматриваемом примере лексический блок должен выдать следующую цепочку лексем:

(const, ИДЕНТИФИКАТОР), (N, ИДЕНТИФИКАТОР), (=, РАВНО),
(10, ЦЕЛОЕ), (; , ТЧКЗПТ)

- *Блок идентификации ключевых слов* – подпрограмма, которая устанавливает, какое из ключевых слов языка Pascal соответствует заданному идентификатору, либо сообщает, что заданный идентификатор не является ключевым словом языка Pascal. Идентификация ключевых слов может быть выделена в *отдельный проход* распознавателя символьной цепочки, то есть идентифицирующий блок будет запускаться *после* того, как лексический блок полностью подготовит *всю* цепочку лексем. Другим подходом является *объединение в один проход* распознавателя символьной цепочки лексического блока и блока идентификации. В этом случае идентификация ключевого слова осуществляется *каждый раз*, когда лексический блок выдал *одну* лексему класса "идентификатор" (без ожидания полной подготовки всей цепочки лексем).

В рассматриваемом примере после распознавания слов цепочка лексем, полученная в результате работы лексического блока, примет следующий вид:

(const, КЛСЛОВО_CONST), (N, ИДЕНТИФИКАТОР), (=, РАВНО),
(10, ЦЕЛОЕ), (; , ТЧКЗПТ)

- *Синтаксический блок* – подпрограмма, которая получает цепочку лексем вида ("символ входного языка", "класс символа входного языка") и устанавливает, соответствует ли она заданным формулам Бэкуса-Наура. Поскольку задание предполагает только распознавание исходной цепочки, фактически данный блок использует для работы только классы символов входного языка.

В рассматриваемом примере синтаксический блок, рассматривая цепочку вида КЛСЛОВО_CONST, ИДЕНТИФИКАТОР, РАВНО, ЦЕЛОЕ, ТЧКЗПТ должен сообщить, что она синтаксически правильна.

2.3.2 Разработка тестов и тестовых программ

После того, как разработана модульная структура программы, необходимо разработать тесты для *каждого* модуля, входящего в модульную структуру (в том числе для головного модуля). *Тесты* представляют собой набор входных и соответствующим им выходных данных, с помощью которых можно показать *наличие ошибки* в данном модуле. На этапе тестирования (см. 2.5) модуль необходимо выполнить на входных данных теста и сравнить полученные результаты с выходными данными теста.

Количество тестов зависит от того, насколько полно в тестовом наборе представлены возможные входные данные модуля (тесты должны проверять работу модуля на обычных и граничных данных). *Качество тестов* зависит от того, насколько высока вероятность обнаружения с их помощью ошибок в данном модуле [2; 5; 6].

В Табл. 1 приведен пример набора тестов для рассмотренных в 2.3.1 подпрограмм модульной структуры распознавателя символьной цепочки¹:

¹ Не утверждается, что данный набор содержит достаточное количество тестов для качественного тестирования. Тесты изображены *символически*, без указания точного формата входных и выходных данных.

Табл. 1. Пример набора тестов для модулей распознавателя цепочки

Модуль	Тесты	
	Входные данные	Выходные данные
Блок транслитерации	Dz12;= и	(D, буква) , (z, буква) , (1, цифра) , (2, цифра) , (; , тчкзпт) , (=, равно) , (, пробел) , (и, ошибка)
Лексический блок	(D, СИМ_БУКВА) , (z, СИМ_БУКВА) , (1, СИМ_ЦИФРА) , (2, СИМ_ЦИФРА) , (, СИМ_ПРОБЕЛ) , (1, СИМ_ЦИФРА) , (2, СИМ_ЦИФРА) , (; , СИМ_ТЧКЗПТ) , (=, СИМ_РАВНО)	(Dz12, ИДЕНТИФИКАТОР) , (12, ЦЕЛОЕ) , (; , ТЧКЗПТ) , (=, РАВНО)
Блок идентификации ключевых слов	Dz12	НЕ КЛСЛОВО
	const	КЛСЛОВО CONST
Синтаксический блок	ИДЕНТИФИКАТОР, ЦЕЛОЕ, ТЧКЗПТ РАВНО	ОШИБКА
	КЛСЛОВО_CONST, ИДЕНТИФИКАТОР, РАВНО, ЦЕЛОЕ	ОШИБКА
	КЛСЛОВО_CONST, ИДЕНТИФИКАТОР, РАВНО, ЦЕЛОЕ, ТЧКЗПТ	ПРАВИЛЬНО
Головной модуль	const Dz=10;	АССЕРТ
	const Dz12=-10;	АССЕРТ
	const On=True;	АССЕРТ
	const end=False;	РЕЈЕСТ

Подготовленные в ходе работ на данном этапе наборы тестов позже войдут в раздел отчета "Тестирование".

Помимо тестов, необходимо разработать *тестовые программы* для тестирования модулей, входящих в модульную структуру. *Тестовая программа модуля* должна обеспечивать запуск модуля на подготовленном наборе входных данных теста, *автоматическое* сравнение полученных результатов с соответствующими выходными данными теста и выдачу соответствующих диагностических сообщений.

2.4 Кодирование

На этапе *кодирования* разработанный ранее на бумаге проект программной системы должен быть реализован в виде текстовых файлов. На данном этапе необходимо выполнить следующие работы:

1. Разработать *структуру текста программы*.
2. Разработать *алгоритмы* реализации модулей, определенных ранее на этапе проектирования.
3. Выполнить *кодирование модулей* программы.

В ходе работ на данном этапе необходимо подготовить раздел отчета "Кодирование".

2.4.1 Разработка структуры текста программы

На данном шаге этапа кодирования необходимо разработать представление модульной структуры программы в виде *файловой структуры* текста программы (в каком файле или файлах будут располагаться модули программы).

В языке программирования Turbo Pascal [4] подпрограммы и другие программные объекты могут быть объединены в модуль *unit*. *Модуль unit* представляет собой отдельно хранимую, независимо компилируемую и неисполняемую программную единицу. Как правило, подпрограммы объединяемые в модуль *unit*, имеют сходное назначение. Например, модуль *unit* может экспортировать ряд подпрограмм для обработки строк. Модули *unit* являются дополнительным средством структурирования текста программы.

На Рис. 2 приведены два примера возможной структуры текста программы для программы, модульная структура которой рассмотрена выше (см. Рис. 1).

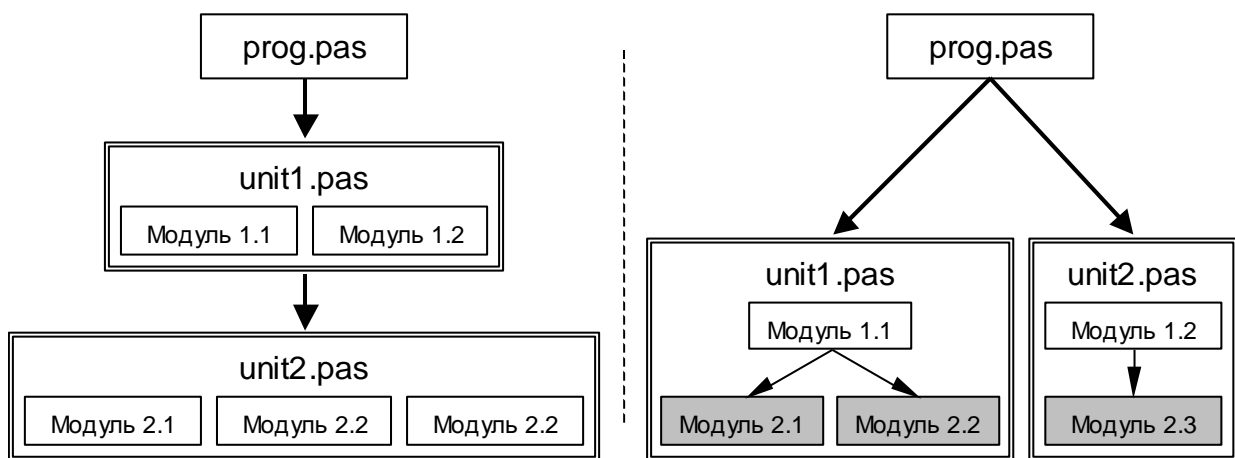


Рис. 2. Примеры структуры текста программы

На данном рисунке стрелки отражают связи *по управлению*. Название модулей *unit* заключено в двойную рамку. Если данный модуль *unit* экспортирует некоторую подпрограмму (то есть подпрограмма определена в интерфейсной секции данного модуля *unit*), то ее название не заштриховано. Если данный модуль *unit* не экспортирует некоторую подпрограмму (данная подпрограмма определена только в секции реализации данного модуля *unit*), то ее название заштриховано.

Как и в случае модульной структуры программы, структура текста программы должна представлять собой *иерархию*. Связи вида "через уровень", "снизу вверх", "горизонтальные" недопустимы.

После того, как разработана структура текста программы, необходимо *подготовить файлы для исходных текстов* головного модуля программы, всех модулей *unit* и тес-

товых программ. При этом все экспортируемые модулями `unit` подпрограммы должны иметь *пустые тела* (нулевое количество операторов между `begin` и `end` данной подпрограммы). Далее необходимо выполнить *компиляцию* головного модуля и тестовых программ и убедиться, что все подготовленные файлы с исходными текстами являются синтаксически правильными.

2.4.2 Разработка алгоритмов

На данном шаге этапа кодирования необходимо разработать *алгоритмы* реализации модулей, определенных ранее на этапе проектирования (см. 2.3.1).

2.4.2.1 Блок транслитерации

Блок транслитерации необходимо реализовать в виде детерминированного конечного автомата [3], который выполняет обработку и распознавание входной символьной цепочки. *Обработка* входной цепочки заключается в собственно выполнении транслитерации – формировании цепочки лексем вида ("символ цепочки", "класс символа цепочки"). *Распознавание* входной цепочки означает, что данный автомат должен отвергать все символьные цепочки, содержащие символы, которые заведомо не удовлетворяют формулам Бэкуса-Наура из условия задачи (например, буквы кириллицы, символы @, # и др.).

В Табл. 2 приведена спецификация транслитератора для распознавателя символьной цепочки, определенной в 1.

Табл. 2. Транслитерация символьной цепочки

Символы	Класс лексемы
a..z, A..Z	буква
0..9	цифра
=	равно
;	тчкзпт
+, -	знак
пробел	пробел
другие символы	ошибка

Отметим, что данный транслитератор *не* может быть использован для реализации распознавателя других цепочек без соответствующих изменений. Например, для данного транслитератора символ '\$' является ошибочным, а в случае распознавателя шестнадцатеричной константы (то есть `const Hex=$10;` и др.) символу '\$' должен соответствовать класс лексемы *зндоллар*.

2.4.2.2 Лексический блок

Лексический блок необходимо реализовать в виде детерминированного конечного автомата, который выполняет обработку и распознавание входной цепочки лексем (полученной как результат работы блока транслитерации). *Обработка* входной цепочки заключается в формировании цепочки лексем вида ("символ входного языка", "класс символа входного языка"). *Распознавание* входной цепочки означает, что данный автомат должен отвергать все цепочки лексем, содержащие лексемы, которые заведомо не удовлетворяют формулам Бэкуса-Наура из условия задачи (например, два знака числа подряд, знак числа после десятичной точки и др.).

Построение конечного автомата лексического блока необходимо выполнить следующим образом. Сначала, используя метод разметки символов [3], нужно построить *конечный распознаватель* входной цепочки лексем: определить список состояний (назва-

ние/номер и семантика состояния), выделить в данном списке начальное состояние и допускающие состояния и указать функцию переходов.

Для разметки символов цепочки, определенной в 1, рассмотрим следующие типичные символьные цепочки (для удобства чтения пробелы заменены символами □):

□□□const□□□Min□□□=□□□-□□□123□□□;□□□

□□□const□□□Yes□□□=□□□TRUE□□□;□□□

В Табл. 3 определены состояния конечного распознавателя лексического блока символьной цепочки, определенной в 1. Начальное состояние – НАЧ. Допустимые состояния – ТЧКЗПТ и ПРОБЕЛ6.

Табл. 3. Состояния конечного распознавателя лексического блока

№ п/п	Состояние	Семантика
1.	НАЧ	Момент до начала обработки цепочки либо чтение пробелов в начале строки.
2.	КЛСЛОВО	Чтение ключевого слова const.
3.	ПРОБЕЛ1	Чтение пробелов, находящихся между ключевым словом const и именем константы.
4.	ИМЯ	Чтение имени константы.
5.	ПРОБЕЛ2	Чтение пробелов, находящихся между именем константы и знаком =.
6.	РАВНО	Прочитан знак =.
7.	ПРОБЕЛ3	Чтение пробелов, находящихся между знаком = и значением константы.
8.	ЗНАК	Прочитан знак целой константы.
9.	ПРОБЕЛ4	Чтение пробелов, находящихся между знаком целой константы и значением целой константы.
10.	ЦЕЛОЕ	Чтение значения целой константы.
11.	ПРОБЕЛ51	Чтение пробелов, находящихся между значением целой константы и знаком ;.
12.	ЛОГКОНСТ	Чтение ключевого слова True либо ключевого слова False.
13.	ПРОБЕЛ52	Чтение пробелов, находящихся между значением логической константы и знаком ;.
14.	ТЧКЗПТ	Прочитан знак ;.
15.	ПРОБЕЛ6	Чтение пробелов, находящихся после знака ;.
16.	Е	Ошибка.

В Табл. 4 приведена спецификация конечного распознавателя лексического блока.

Табл. 4. Конечный распознаватель лексического блока

	<i>буква</i>	<i>цифра</i>	<i>пробел</i>	<i>равно</i>	<i>знак</i>	<i>тчкзпт</i>	
НАЧ	КЛСЛОВО	Е	НАЧ	Е	Е	Е	0
КЛСЛОВО	КЛСЛОВО	КЛСЛОВО	ПРОБЕЛ1	Е	Е	Е	0
ПРОБЕЛ1	ИМЯ	Е	ПРОБЕЛ1	Е	Е	Е	0
ИМЯ	ИМЯ	ИМЯ	ПРОБЕЛ2	РАВНО	Е	Е	0
ПРОБЕЛ2	Е	Е	ПРОБЕЛ2	РАВНО	Е	Е	0
РАВНО	ЛОГКОНСТ	ЦЕЛОЕ	ПРОБЕЛ3	Е	ЗНАК	Е	0
ПРОБЕЛ3	ЛОГКОНСТ	ЦЕЛОЕ	ПРОБЕЛ3	Е	ЗНАК	Е	0
ЗНАК	Е	ЦЕЛОЕ	ПРОБЕЛ4	Е	Е	Е	0
ПРОБЕЛ4	Е	ЦЕЛОЕ	ПРОБЕЛ4	Е	Е	Е	0

	буква	цифра	пробел	равно	знак	тчкзпт	
ЦЕЛОЕ	Е	ЦЕЛОЕ	ПРОБЕЛ51	Е	Е	ТЧКЗПТ	0
ПРОБЕЛ51	Е	Е	ПРОБЕЛ51	Е	Е	ТЧКЗПТ	0
ЛОГКОНСТ	ЛОГКОНСТ	Е	ПРОБЕЛ52	Е	Е	ТЧКЗПТ	0
ПРОБЕЛ52	Е	Е	ПРОБЕЛ52	Е	Е	ТЧКЗПТ	0
ТЧКЗПТ	Е	Е	ПРОБЕЛ6	Е	Е	Е	1
ПРОБЕЛ6	Е	Е	ПРОБЕЛ6	Е	Е	Е	1
Е	Е	Е	Е	Е	Е	Е	0

Далее необходимо выполнить *редукцию* построенного автомата (или доказать, что построенный автомат является минимальным). Редукция автомата осуществляется следующим образом [3]. Сначала необходимо найти и отбросить недостижимые состояния построенного автомата. Затем необходимо найти группы эквивалентных состояний и для всех групп заменить каждое состояние в найденной группе на одно состояние. Поиск эквивалентных состояний наиболее эффективно проводить методом разбиения [3].

В рассматриваемом нами примере у конечного распознавателя лексического блока нет недостижимых состояний. Состояния РАВНО и ПРОБЕЛ3, ЗНАК и ПРОБЕЛ4, ТЧКЗПТ и ПРОБЕЛ6 попарно идентичны, а следовательно, эквиваленты. Дальнейший процесс редукции конечного распознавателя лексического блока приведен в Табл. 5.

Табл. 5. Редукция конечного распознавателя лексического блока

Шаг	Результат (блоки состояний)	Действия
0.	$P_0 = \{ НАЧ, КЛСЛОВО, ПРОБЕЛ1, ИМЯ, ПРОБЕЛ2, РАВНО, ЗНАК, ЦЕЛОЕ, ПРОБЕЛ51, ПРОБЕЛ52, ЛОГКОНСТ, ТЧКЗПТ, Е \}$	Разбиваем P_0 на два блока: допустимые и отвергающие состояния.
1.	$P_{1_1} = \{ НАЧ, КЛСЛОВО, ПРОБЕЛ1, ИМЯ, ПРОБЕЛ2, РАВНО, ЗНАК, ЦЕЛОЕ, ПРОБЕЛ51, ПРОБЕЛ52, ЛОГКОНСТ, Е \},$ $P_{1_2} = \{ ТЧКЗПТ \}$	Разбиваем P_{1_1} по входу <i>буква</i> .
2.	$P_{2_1} = \{ НАЧ, КЛСЛОВО \},$ $P_{2_2} = \{ ПРОБЕЛ1, ИМЯ \},$ $P_{2_3} = \{ РАВНО, ЛОГКОНСТ \},$ $P_{2_4} = \{ ЗНАК, ЦЕЛОЕ, ПРОБЕЛ2, ПРОБЕЛ51, ПРОБЕЛ52, Е \},$ $P_{2_5} = \{ ТЧКЗПТ \}$	1. Разбиваем P_{2_1} по входу <i>цифра</i> . 2. Разбиваем P_{2_2} по входу <i>цифра</i> . 3. Разбиваем P_{2_3} по входу <i>тчкзпт</i> . 4. Разбиваем P_{2_4} по входу <i>равно</i> .
3.	$P_{3_1} = \{ НАЧ \},$ $P_{3_2} = \{ КЛСЛОВО \},$ $P_{3_3} = \{ ПРОБЕЛ1 \},$ $P_{3_4} = \{ ИМЯ \},$ $P_{3_5} = \{ РАВНО \},$ $P_{3_6} = \{ ЛОГКОНСТ \},$ $P_{3_7} = \{ ПРОБЕЛ2 \},$ $P_{3_8} = \{ ЗНАК, ЦЕЛОЕ, ПРОБЕЛ51, ПРОБЕЛ52, Е \},$ $P_{3_9} = \{ ТЧКЗПТ \}$	Разбиваем P_{3_8} по входу <i>тчкзпт</i> .

Шаг	Результат (блоки состояний)	Действия
4.	$P_{4_1} = \{ \text{НАЧ} \},$ $P_{4_2} = \{ \text{КЛСЛОВО} \},$ $P_{4_3} = \{ \text{ПРОБЕЛ1} \},$ $P_{4_4} = \{ \text{ИМЯ} \},$ $P_{4_5} = \{ \text{РАВНО} \},$ $P_{4_6} = \{ \text{ЛОГКОНСТ} \},$ $P_{4_7} = \{ \text{ПРОБЕЛ2} \},$ $P_{4_8} = \{ \text{ЗНАК, Е} \},$ $P_{4_9} = \{ \text{ЦЕЛОЕ, ПРОБЕЛ51, ПРОБЕЛ52} \},$ $P_{4_{10}} = \{ \text{ТЧКЗПТ} \}$	1. Разбиваем P_{4_8} по входу <i>цифра</i> . 2. Разбиваем P_{4_9} по входу <i>цифра</i> .
5.	$P_{5_1} = \{ \text{НАЧ} \},$ $P_{5_2} = \{ \text{КЛСЛОВО} \},$ $P_{5_3} = \{ \text{ПРОБЕЛ1} \},$ $P_{5_4} = \{ \text{ИМЯ} \},$ $P_{5_5} = \{ \text{РАВНО} \},$ $P_{5_6} = \{ \text{ЛОГКОНСТ} \},$ $P_{5_7} = \{ \text{ПРОБЕЛ2} \},$ $P_{5_8} = \{ \text{ЗНАК} \},$ $P_{5_9} = \{ \text{ЦЕЛОЕ} \},$ $P_{5_{10}} = \{ \text{ПРОБЕЛ51, ПРОБЕЛ52} \},$ $P_{5_{11}} = \{ \text{ТЧКЗПТ} \}$ $P_{5_{12}} = \{ \text{Е} \}.$	Блок $P_{5_{10}}$ не допускает дальнейшего разбиения, так как относительно любого входного символа из состояний ПРОБЕЛ51 и ПРОБЕЛ52 осуществляется переход в один и тот же блок состояний. Таким образом, состояния ПРОБЕЛ51 и ПРОБЕЛ52 эквивалентны и могут быть заменены одним состоянием ПРОБЕЛ5.

В Табл. 6 приведен результат редукции конечного распознавателя лексического блока.

Табл. 6. Минимальный конечный распознаватель лексического блока

	<i>буква</i>	<i>цифра</i>	<i>пробел</i>	<i>равно</i>	<i>знак</i>	<i>тчкзпт</i>	
НАЧ	КЛСЛОВО	Е	НАЧ	Е	Е	Е	0
КЛСЛОВО	КЛСЛОВО	КЛСЛОВО	ПРОБЕЛ1	Е	Е	Е	0
ПРОБЕЛ1	ИМЯ	Е	ПРОБЕЛ1	Е	Е	Е	0
ИМЯ	ИМЯ	ИМЯ	ПРОБЕЛ2	РАВНО	Е	Е	0
ПРОБЕЛ2	Е	Е	ПРОБЕЛ2	РАВНО	Е	Е	0
РАВНО	ЛОГКОНСТ	ЦЕЛОЕ	РАВНО	Е	ЗНАК	Е	0
ЗНАК	Е	ЦЕЛОЕ	ЗНАК	Е	Е	Е	0
ЦЕЛОЕ	Е	ЦЕЛОЕ	ПРОБЕЛ5	Е	Е	ТЧКЗПТ	0
ПРОБЕЛ5	Е	Е	ПРОБЕЛ5	Е	Е	ТЧКЗПТ	0
ЛОГКОНСТ	ЛОГКОНСТ	Е	ПРОБЕЛ5	Е	Е	ТЧКЗПТ	0
ТЧКЗПТ	Е	Е	ТЧКЗПТ	Е	Е	Е	1
Е	Е	Е	Е	Е	Е	Е	0

Затем полученный минимальный конечный распознаватель необходимо преобразовать в *обрабатывающий автомат*. Алфавит распознавателя следует дополнить символом "концевой маркер", а переходы в состояние "ошибка" заменить на вызов некоторой примитивной процедуры "отвергнуть", которая останавливает обработку и отвергает входную цепочку. Остальные переходы также следует дополнить вызовами некоторых примитивных процедур, которые обеспечивают формирование выходной цепочки лексем (например, "допустить", "сформировать лексему" и др.).

В рассматриваемом нами примере обрабатывающему автомату лексического блока потребуются примитивные процедуры, указанные в Табл. 7.

Табл. 7. Прimitives процедуры обрабатывающего автомата лексического блока

№ п/п	Процедура	Семантика
1.	<i>ДА</i>	Остановить обработку и допустить цепочку.
2.	<i>НЕТ</i>	Остановить обработку и отвергнуть цепочку.
3.	<i>ОБРАБОТАТЬ</i>	Добавить входной символ к значению текущей лексемы.
4.	<i>ЛЕКСЕМА (класс)</i>	Увеличить счетчик лексем на 1, установить заданный класс текущей лексемы.

Обрабатывающий автомат лексического блока для рассматриваемого нами примера приведен в Табл. 8.

Табл. 8. Обрабатывающий автомат лексического блока

	<i>буква</i>	<i>цифра</i>	<i>пробел</i>	<i>равно</i>	<i>знак</i>	<i>тчкзпт</i>	<i>- </i>
НАЧ	① КЛСЛОВО		НАЧ				
КЛСЛОВО	② КЛСЛОВО	② КЛСЛОВО	ПРОБЕЛ1				
ПРОБЕЛ1	① ИМЯ		ПРОБЕЛ1				
ИМЯ	② ИМЯ	② ИМЯ	ПРОБЕЛ2	③ РАВНО			
ПРОБЕЛ2			ПРОБЕЛ2	③ РАВНО			
РАВНО	① ЛОГКОНСТ	④ ЦЕЛОЕ	РАВНО		⑤ ЗНАК		
ЗНАК		④ ЦЕЛОЕ	ЗНАК				
ЦЕЛОЕ		② ЦЕЛОЕ	ПРОБЕЛ5			⑥ ТЧКЗПТ	
ПРОБЕЛ5			ПРОБЕЛ5			⑥ ТЧКЗПТ	
ЛОГКОНСТ	② ЛОГКОНСТ		ПРОБЕЛ5			⑥ ТЧКЗПТ	
ТЧКЗПТ			ТЧКЗПТ				<i>ДА</i>

Пустым клеткам соответствует вызов примитивной процедуры *НЕТ*. Для удобства чтения действия обрабатывающего автомата, выполняемые перед переходом в новое состояние, обозначены цифрами. Семантика действий дана в Табл. 9.

Табл. 9. Процедуры переходов обрабатывающего автомата лексического блока

Действие	Семантика
①	ЛЕКСЕМА (ИДЕНТ) ; <i>ОБРАБОТАТЬ</i> ;
②	<i>ОБРАБОТАТЬ</i> ;
③	ЛЕКСЕМА (РАВНО) ; <i>ОБРАБОТАТЬ</i> ;
④	ЛЕКСЕМА (ЦЕЛОЕ) ; <i>ОБРАБОТАТЬ</i> ;
⑤	ЛЕКСЕМА (ЗНАК) ; <i>ОБРАБОТАТЬ</i> ;

6	ЛЕКСЕМА (ТЧКЗПТ) ; ОБРАБОТАТЬ ;
----------	------------------------------------

2.4.2.3 Блок идентификации ключевых слов

Для реализации *блока идентификации слов* может быть использован один из следующих алгоритмов [3]: линейный поиск, бинарный поиск, конечный распознаватель, индексация, хеширование. Выбор алгоритма реализации данного модуля санкционируется преподавателем, ведущим практику в данной группе.

2.4.2.4 Синтаксический блок

Синтаксический блок необходимо реализовать в виде детерминированного конечного распознавателя. Построенный автомат необходимо подвергнуть редукции способом, аналогичным приведенному выше в указаниях к разработке алгоритмов для лексического блока.

Конечный распознаватель синтаксического блока рассматриваемого нами примера приведен в Табл. 10. Пустым клеткам соответствует вызов примитивной процедуры *НЕТ*, которая отвергает цепочку.

Табл. 10. Конечный распознаватель синтаксического блока

	КЛСЛОВО_ CONST	ИДЕНТ	РАВНО	ЗНАК	ЦЕЛОЕ	КЛСЛОВО_ TRUE_FALSE	КЛСЛОВО_ ДРУГОЕ	ТЧКЗПТ	
НАЧ	CONST								0
CONST		ИМЯ							0
ИМЯ			РАВНО						0
РАВНО				ЗНАК	ЦЕЛОЕ	ЛОГКОНСТ			0
ЗНАК					ЦЕЛОЕ				0
ЦЕЛОЕ								ТЧКЗПТ	0
ЛОГКОНСТ								ТЧКЗПТ	0
ТЧКЗПТ									1

Заметим, что состояния ЦЕЛОЕ и ЛОГКОНСТ идентичны, а следовательно, эквивалентны. Заменяем эти состояния одним состоянием ЗНАЧКОНСТ. Результат приведен в Табл. 11. Очевидно, что полученный автомат не подлежит дальнейшей редукции и является минимальным.

Табл. 11. Минимальный конечный распознаватель синтаксического блока

	КЛСЛОВО_ CONST	ИДЕНТ	РАВНО	ЗНАК	ЦЕЛОЕ	КЛСЛОВО_ TRUE_FALSE	КЛСЛОВО_ ДРУГОЕ	ТЧКЗПТ	
НАЧ	CONST								0
CONST		ИМЯ							0
ИМЯ			РАВНО						0
РАВНО				ЗНАК	ЗНАЧКОНСТ	ЗНАЧКОНСТ			0
ЗНАК					ЗНАЧКОНСТ				0
ЗНАЧКОНСТ								ТЧКЗПТ	0
ТЧКЗПТ									1

2.4.3 Кодирование модулей

На данном шаге этапа кодирования необходимо выполнить *кодирование модулей*, разработанных ранее. Кодирование модулей необходимо производить, двигаясь *снизу*

вверх по схеме модульной структуры программы (все подпрограммы одного уровня иерархии, входящие в один модуль unit).

По окончании кодирования модуля необходимо перейти к *тестированию* данного модуля (см. 2.5). Приступать к кодированию следующего модуля разрешается только *после* устранения всех выявленных ошибок.

2.5 Тестирование

На этапе *тестирования* необходимо выполнить тестовые программы модулей, используя в качестве входных данных входные данные подготовленных ранее тестов, и сравнить полученные выходные данные с выходными данными тестов. Тестирование модуля в отдельности называется *автономным*. Тестирование программной системы в целом (то есть головного модуля) называется *комплексным*.

Тестирование модуля выполняется непосредственно *после кодирования* данного модуля. Таким образом, тестирование всех модулей выполняется в порядке *снизу вверх* по схеме модульной структуры программы.

При тестировании модуля несовпадение ожидаемых и действительных результатов означает, что данный тест позволил показать *наличие ошибки* в соответствующем модуле. Совпадение всех ожидаемых и действительных результатов *не* означает, что в данном модуле отсутствуют ошибки, поскольку тестовый набор, как правило, не совпадает с множеством всех возможных входных данных модуля.

В случае, если установлен факт наличия ошибки, необходимо провести отладку данного модуля. *Отладка* модуля подразумевает поиск и устранение ошибки. При выполнении отладки полезно сочетать использование *отладчика* системы программирования [4] с "*сухой*" отладкой (работа без компьютера, с карандашом и листингом модуля).

После устранения ошибки необходимо *вновь* выполнить тестирование данного модуля на *всех* тестах. Тестирование и последующая отладка модуля продолжают до тех пор, пока тестирование не приведет к совпадению всех ожидаемых и действительных результатов.

Проведение тестов необходимо *документировать*. Пример оформления протокола тестирования модуля приведен в Табл. 12.

Табл. 12. Пример протокола тестирования головного модуля

№ п/п	Входные данные	Выходные данные	Действительный результат	Тест пройден?
1.	const N=10;	ACCEPT	ACCEPT	Да
2.	const N=-10;	ACCEPT	ACCEPT	Да
3.	const N=+10;	ACCEPT	ACCEPT	Да
4.	const N=-+10;	REJECT	ACCEPT	НЕТ
5.	const On=True;	ACCEPT	ACCEPT	Да
6.	const MaxN = 123 ;	ACCEPT	ACCEPT	Да
7.	const N=10	REJECT	REJECT	Да
8.	const Min=-True;	REJECT	REJECT	Да
9.	const Off=False;	REJECT	REJECT	Да
10.	const Min=123.456;	REJECT	REJECT	Да
11.	const BEGIN=123;	REJECT	REJECT	Да

Табл. 12 отражает следующую типичную ситуацию, возникающую при проведении комплексного тестирования: в ходе автономного тестирования ошибки не обнаружены, а тестирование головного модуля выявило ошибку (тест № 4).

В данной ситуации можно использовать следующий достаточно эффективный способ поиска ошибки. По тесту, который выявил ошибку в головном модуле, необходимо подготовить соответствующие *дополнительные тесты* для подчиненных ему модулей. В Табл. 13 приведен пример дополнительных тестов для модулей распознавателя цепочки, подготовленных по тесту № 4 головного модуля². Далее необходимо выполнить тестирование модулей на дополнительных тестах. Очевидно, что будет установлен факт наличия ошибки как минимум в одном из модулей. Данный модуль необходимо отладить в соответствии с описанной выше дисциплиной.

Табл. 13. Пример дополнительных тестов для модулей распознавателя цепочки

Модуль	Тесты	
	Входные данные	Выходные данные
Блок транслитерации	const N=--+10;	(с, буква) , (о, буква) , (n, буква) , (s, буква) , (t, буква) , (, пробел) , (=, равно) , (N, равно) , (-, знак) , (+, знак) , (1, цифра) , (0, цифра) , (; , тчкзпт)
Лексический блок	(с, буква) , (о, буква) , (n, буква) , (s, буква) , (t, буква) , (, пробел) , (=, равно) , (N, равно) , (-, знак) , (+, знак) , (1, цифра) , (0, цифра) , (; , тчкзпт)	(const, ИДЕНТИФИКАТОР) , (N, ИДЕНТИФИКАТОР) , (=, РАВНО) , (-, ЗНАК) , (+, ОШИБКА)
Блок идентификации ключевых слов	N	НЕ_КЛСЛОВО
	const	КЛСЛОВО CONST
Синтаксический блок	КЛСЛОВО_CONST, ИДЕНТИФИКАТОР, ЦЕЛОЕ, РАВНО, ТЧКЗПТ	ПРАВИЛЬНО

² Тесты изображены *символически*, без указания точного формата входных и выходных данных.

2.6 Сопровождение

Сопровождение предполагает внедрение программной системы (передача системы и документации на нее заказчику, обучение конечных пользователей и т.п.) и последующее устранение ошибок, выявляемых во время эксплуатации системы.

В контексте учебной практики данный этап подразумевает *защиту отчета* о прохождении практики. К защите отчета студент обязан подготовить:

1. Полностью готовый *текст отчета*.
Указания по оформлению отчета приведены в разделе 3.
2. *Исходные тексты* программной системы.
Обязательно наличие спецификаций файлов и подпрограмм. *Спецификация файла* представляет собой комментарий, в котором указано имя файла, фамилия и группа автора, дата написания и назначение файла. *Спецификация подпрограммы* представляет собой комментарий, в котором указано назначение подпрограммы и семантика параметров подпрограммы. В исходных текстах должна соблюдаться *лесенка* – отступы от левого края, показывающие структурную вложенность операторов.
3. *Исполняемые файлы* программы и тестовых программ модулей, а также *файлы тестовых наборов*.

При проведении зачета по учебной практике преподаватель использует следующие *критерии итоговой оценки* за вычислительную практику:

1. Оценка "*отлично*" выставляется в случае, если:
 - студент подготовил полный и аккуратно оформленный в соответствии с требованиями отчет;
 - разработал и успешно протестировал программную систему;
 - успешно защитил подготовленный отчет (защита отчета предполагает быстрые и точные ответы студента на вопросы преподавателя, касающиеся программной системы).В случае отсутствия отчета студент получает оценку "*неудовлетворительно*" – даже если имеются исходные тексты программы, и программа была успешно протестирована.
2. Оценка "*хорошо*" выставляется в случае, если:
 - студент подготовил полный и аккуратно оформленный в соответствии с требованиями отчет;
 - разработал программную систему, однако тестирование системы преподавателем во время защиты отчета показало наличие ошибки либо
 - студент не смог достаточно успешно защитить подготовленный им отчет.
3. Оценка "*удовлетворительно*" выставляется в случае, если:
 - студент подготовил аккуратно оформленный в соответствии с требованиями отчет, однако
 - разработка программной системы не доведена до конца (в данном случае должны быть выполнены работы, как минимум, по следующим этапам технологического цикла разработки: анализ, спецификация и проектирование).
4. Оценка "*неудовлетворительно*" выставляется в случае, если:
 - студент не подготовил отчет о прохождении практики либо
 - в ходе разработки программной системы не выполнил хотя бы одну из работ, предусмотренных следующими этапами технологического цикла: анализ, спецификация и проектирование.
5. Оценка может быть *снижена* в одном из следующих случаев:
 - подготовленный студентом отчет оформлен небрежно или не в соответствии с требованиями;

- исходные тексты программной системы оформлены не в соответствии с требованиями (отсутствие спецификаций, лесенки и др.);
- на защите отчета студент не может дать быстрый и точный ответ на вопрос относительно разработанной им программной системы

2.7 Распределение времени

Учебная практика длится *четыре учебные недели* (по 5 дней). Распределение времени и текущий контроль выполняемых работ осуществляется в соответствии с *графиком* прохождения учебной практики, который размещен на странице преподавателя.

3. Указания по подготовке отчета о практике

Подготовка отчета о прохождении учебной практики должна осуществляться *во время* практики, а *не по окончании* выполнения всех работ. *Одновременно* с прохождением очередного этапа технологического цикла разработки необходимо готовить соответствующий раздел отчета.

Основные *требования к оформлению* отчета следующие:

1. Отчет должен быть сброшюрован из листов формата А4.
2. Титульный лист должен содержать следующие данные:
 - вверху листа – полностью наименования министерства образования, университета и кафедры;
 - в середине листа – заголовок "Отчет по учебной практике", инициалы, фамилия и группа студента, подготовившего отчет, инициалы и фамилия преподавателя, проверяющего отчет, дату защиты отчета;
 - внизу листа – город и год.
3. Текст должен располагаться только на одной стороне листа. Ориентация листа книжная. Размер полей не менее 2 см. Размер шрифта не менее 12 пунктов. Листы отчета (кроме титульного) должны быть пронумерованы.
4. Отчет должен быть подготовлен аккуратно, без многочисленных помарок и исправлений. Рукописное исполнение отчета нежелательно (но возможно).

Основные *требования к содержанию* отчета следующие (далее приведено оглавление отчета и содержание каждого пункта):

1. Спецификация

В данном разделе следует поместить *формулировку задания*: определение символической цепочки для распознавания, описание формата входных и выходных данных, примеры входных и соответствующих им выходных данных.

2. Проектирование

Данный раздел следует разбить на следующие пункты:

2.1 Модульная структура

В данном пункте следует поместить *рисунок со схемой модульной структуры* с кратким описанием назначения входящих в нее модулей.

2.2 Интерфейсы модулей

В данном пункте следует поместить *заголовки и спецификации модулей*, упомянутых в пункте 2.1. Заголовок модуля должен содержать имя модуля, имена и типы формальных параметров. Спецификация модуля должна содержать назначение модуля, описание формальных параметров модуля и их семантики (для модулей-функций также описание возвращаемого результата и его семантики).

В данном пункте также следует поместить *определение используемых типов данных* с комментариями, поясняющими семантику этих типов.

3. Кодирование

Данный раздел следует разбить на следующие пункты:

3.1 Структура текста программы

В данном пункте следует поместить *рисунок со схемой структуры текста программы* с кратким описанием назначения входящих в нее модулей unit и экспортируемых данными модулями unit подпрограмм.

3.2 Алгоритмы реализации модулей

Данный пункт следует разбить на следующие подпункты:

3.2.1 Блок транслитерации

В данном пункте необходимо поместить *таблицу транслитерации* символьной цепочки.

3.2.2 Лексический блок

В данном пункте необходимо описать построение *обрабатывающего автомата лексического блока*: построение и редукция конечного распознавателя лексического блока, затем примитивные процедуры и преобразование распознавателя в обрабатывающий автомат.

При использовании стандартных методов поиска недостижимых и эквивалентных состояний (например, методом разбиения) следует поместить соответствующую библиографическую ссылку (например, [3]).

3.2.3 Синтаксический блок

Данный пункт должен иметь содержание, аналогичное пункту 3.2.1 – применительно к *конечному автомату синтаксического блока*.

3.2.4 Блок идентификации ключевых слов

В данном пункте необходимо поместить *описание используемого метода идентификации ключевых слов* и поместить соответствующую библиографическую ссылку (например, [3]).

3.4 Размер текста программы (в строках)

Данный пункт отчета должен присутствовать, если выполнены работы по кодированию программной системы. В данном пункте следует поместить округленный до сотен *общий размер созданных исходных текстов в строках* (включая пустые строки, комментарии, спецификации программных файлов и др.).

4. Тестирование

Данный раздел следует разбить на следующие пункты:

4.1 Автономное тестирование

Данный пункт следует разбить на подпункты в соответствии с модульной структурой программы и в каждом пункте поместить *протокол тестирования соответствующего модуля*.

4.2 Комплексное тестирование

В данном пункте следует поместить *протокол тестирования головного модуля программы*.

5. Заключение

В данном разделе следует поместить *краткую сводку всех полученных результатов*: перечислить пройденные этапы технологического цикла разработки и соответствующие выполненные работы и указать не пройденные этапы (не выполненные работы) и работы, выполненные лишь частично.

6. Литература

В данном разделе следует поместить все *использованные библиографические источники*, ссылки на которые имеются в остальных разделах отчета. Список литературы должен содержать не менее двух наименований (например, [1] и [3] данного доку-

мента). Каждый элемент списка литературы должен быть оформлен в соответствии с оформлением библиографических ссылок в списке литературы данного документа.

4. Варианты заданий

В данном разделе с помощью формул Бэкуса-Наура описаны *варианты заданий* практики. Вариант задания обозначается буквой латинского алфавита от **A** до **X**. Вариант задания содержит *определение символьной цепочки* для распознавания и *примеры допустимых цепочек*.

Общие для всех вариантов определения нетерминальных цепочек выделены в отдельный пункт данного раздела. Семантические ограничения, накладываемые на распознаваемые цепочки, также выделены в отдельный пункт данного раздела.

4.1 Общие определения

Ниже в алфавитном порядке даны формулы Бэкуса-Наура, определяющие нетерминальные цепочки, общие для всех вариантов практики. Терминальные символы формул выделяются полужирным шрифтом (например, **\$**, **CONST** и т.д.). Нетерминальные символы формул, определения которых даются в варианте задания, выделены курсивом (например, *<условие>*, *<выражение>* и т.д.).

Для унификации формул все терминальные символы, являющиеся мнемониками английского языка, даны в верхнем регистре (например, **AND**, **BYTE** и т.д.). Тем не менее, использование символов нижнего регистра в цепочке также допустимо (например, **and**, **And**, **Byte**, **byte** и т.д.).

Терминальные символы цепочки могут отделяться друг от друга одним или более пробелами. Например, следующая цепочка является допустимой (для удобства чтения пробелы заменены символами □): **var**□□**A**□, □**B**□□□ : □**Byte**□□ ; □□□

<16-ричная буква>::=**A** | **B** | **C** | **D** | **E** | **F**

<16-ричная константа>::= **\$**<список 16-ричных букв и цифр>

<арифметическая операция>::=<слагаемое1><знак арифметической операции><слагаемое2>

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

<буква порядка>::=**E**

<булевская операция>::= **AND** | **OR**

<вещественная константа>::=<целое со знаком> | <вещественное со знаком> | <вещественное без знака>

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

<вещественное без знака>::=<число с фиксированной точкой> | <число с плавающей точкой>

<вызов подпрограммы>::=<идентификатор> (<список параметров>)

<дробная часть>::=пусто | <целое без знака>

<знак>::=**+** | **-**

<знак арифметической операции>::=**+** | **-** | ***** | **div** | **mod**

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

<логическая константа>::=**TRUE** | **FALSE**

<оператор if-then>::=**IF** <условие> **THEN** <оператор1> ;

<оператор if-then-else>::=**IF** <условие> **THEN** <оператор1> **ELSE** <оператор2> ;

<оператор repeat-until>::=**REPEAT** <оператор> **UNTIL** <условие> ;

<оператор while-do>::=**WHILE** <условие> **DO** <оператор> ;

<оператор присваивания> ::= <идентификатор> := <выражение> ;
 <операция сравнения> ::= > | < | >= | <= | <>
 <описание константы> ::= **CONST** <идентификатор> = <значение> ;
 <описание переменных> ::= **VAR** <список идентификаторов> : <тип> ;
 <описание типа> ::= **TYPE** <идентификатор> = <тип> ;
 <основание> ::= пусто | <целое без знака>
 <порядок> ::= <знак> <целое без знака>
 <пусто> ::=
 <список 16-ричных букв и цифр> ::= <цифра> | <16-ричная буква> <список 16-ричных букв и цифр>
 <список идентификаторов> ::= <идентификатор> | <идентификатор> , <список идентификаторов>
 <стандартный тип> ::= **BOOLEAN** | **BYTE** | **CHAR** | **INTEGER** | **LONGINT** | **REAL** | **STRING** | **WORD**
 <строковый тип> ::= **STRING** | **STRING** [<длина>]
 <условный оператор> ::= <оператор if-then> | <оператор if-then-else>
 <целая константа> ::= <целое со знаком> | <целое без знака>
 <целая часть> ::= <целое без знака>
 <целое без знака> ::= <цифра> | <цифра> <целое без знака>
 <целое со знаком> ::= <знак> <целое без знака>
 <цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
 <число с плавающей точкой> ::= <основание> <буква порядка> <порядок>
 <число с фиксированной точкой> ::= <целая часть> . <дробная часть>
 <элемент массива> ::= <идентификатор> [<индексное выражение>]

4.2 Семантические ограничения

Для любого варианта практики значение нетерминального символа <идентификатор> не должно совпадать (без учета регистра) ни с одним ключевым словом языка программирования Pascal [1]. Таким образом, следующие символьные цепочки являются недопустимыми (ошибочные терминальные символы выделены полужирным шрифтом):

```

const Const=1;
var REPEAT : REAL;
var while : array [of..else] of Integer;
  
```

4.2.1 Варианты

В варианте задания определения нетерминальных символов даются в том порядке, в котором они появляются в определении символьной цепочки данного варианта. Отступы от левого края отражают структурную вложенность формул для данного варианта. Отсутствующие в тексте данного варианта определения нетерминальных символов приводятся в пункте 4.1.

А. <цепочка> ::= <описание константы>

<значение> ::= <целая константа> | <16-ричная константа>

Примеры допустимых цепочек

```

const one=1;
const Code=-23;
const HEX=$10A;
  
```

В. <цепочка>::=<описание переменных>
<тип>::=<стандартный тип> | <строковый тип>
<длина>::=<целое без знака>

Примеры допустимых цепочек

```
var eps: real;  
var L, M, N : Integer;  
var InputFileName, OutputFileName : String[12];
```

С. <цепочка>::=<описание переменных>
<тип>::=**array** [<нижняя граница>..**верхняя граница**] **of** <стандартный тип>
<нижняя граница>::=<идентификатор> | <целая константа>
<верхняя граница>::=<нижняя граница>

Примеры допустимых цепочек

```
var arr: array[1..N] of Real;  
var a1, a2, a3: array[-1..10] of Integer;  
var Inp, Out: array [Min..Max] of Real;
```

Д. <цепочка>::=<описание константы>
<значение>::=<вещественная константа>

Примеры допустимых цепочек

```
const one=1.00;  
const twoneg=-2.00;  
const Eps=3E-4;  
const Half=.5;  
const Thousand=1e+3;
```

Е. <цепочка>::=<оператор if-then-else>
<условие>::=<идентификатор> | <логическая константа>
<оператор1>::=<вызов подпрограммы>
<список параметров>::=<список идентификаторов>
<оператор2>::=<оператор1>

Примеры допустимых цепочек

```
if True then A(D) else A(B);  
if Good then OK(Param1, Param2) else Bad(i);  
if False then d_1(a,b,c,d) else foo(z1,z2);
```

Ф. <цепочка>::=<условный оператор>
<условие>::=<идентификатор>
<оператор1>::=<оператор присваивания>
<выражение>::=<вызов подпрограммы>
<список параметров>::=<идентификатор>
<оператор2>::=<вызов подпрограммы>
<список параметров>::=<целая константа>

Примеры допустимых цепочек

```
if Cond then A1:=Func(id);  
if Cond then myvar:=foo(id1) else proc(2);  
if Cond then samp:=func3(id1) else proc(-32);
```

Г. <цепочка>::=<оператор while-do>
<условие>::=<идентификатор> | <логическая константа>
<оператор>::=<оператор присваивания>
<выражение>::=<идентификатор> | <элемент массива>

<индексное выражение>::**=**<целая константа>

Примеры допустимых цепочек

```
while Cond do id:=myvar;  
while True do id:=arr[1];  
while Cond do id:=arr[-22];
```

Н. <цепочка>::**=**<оператор repeat-until>

<оператор>::**=**<оператор присваивания>
 <выражение>::**=**<целая константа>
<условие>::**=**<вызов подпрограммы>
 <список параметров>::**=**<идентификатор> | <элемент массива>
 <индексное выражение>::**=**<целая константа>

Примеры допустимых цепочек

```
repeat myvar:=1 until func(id);  
repeat id:=23 until func(arr[12]);  
repeat id:=-22 until cond(arr[-9]);
```

I. <цепочка>::**=**<оператор присваивания>

<выражение>::**=**<арифметическая операция>
 <слагаемое1>::**=**<элемент массива>
 <индексное выражение>::**=**<список целых констант>
 <список целых констант>::**=**<целая константа> | <целая констан-
та>, <список целых констант>
 <слагаемое2>::**=**<вызов подпрограммы>
 <список параметров>::**=**<список идентификаторов>

Примеры допустимых цепочек

```
myvar:=arr[1] + func(id);  
myvar:=arr[1,2,3] div func(id1, id2);  
myvar:=arr[-12,+23,3455] mod func(p1, p2, p3, p4, p5, p6);
```

J. <цепочка>::**=**<описание типа>

<тип>::**=****array** [<нижняя граница> ..<верхняя граница>] **of** <строковый тип>
 <нижняя граница>::**=**<целая константа>
 <верхняя граница>::**=**<нижняя граница>
 <длина>::**=**<идентификатор>

Примеры допустимых цепочек

```
type StrArr=array [1..10] of String;  
type Chains=array [-1..3] of String[Max];  
type MyArr=array [-1..+10] of String[MaxLen];
```

К. <цепочка>::**=**<оператор repeat-until>

<условие>::**=**<вызов подпрограммы>
 <список параметров>::**=**<список идентификаторов>
<оператор>::**=**<вызов подпрограммы>
 <список параметров>::**=**<целая константа>

Примеры допустимых цепочек

```
repeat myproc(1) until myfunc(a);  
repeat myproc(+16) until myfunc(param1, param2);  
repeat myproc(-123) until myfunc(p1, p2, p3, p4);
```

L. <цепочка>::**=**<оператор while-do>

<условие>::**=**<вызов подпрограммы>

<список параметров>::=<список идентификаторов>
 <оператор>::=<вызов подпрограммы>
 <список параметров>::=<элемент массива>
 <индексное выражение>::=<целая константа>

Примеры допустимых цепочек

```

while Cond(a) do proc(b[1]);
while Cond(param) do proc(arr[12]);
while Cond(p1, p2, p3, p4) do proc(arr[-15]);
  
```

М. <цепочка>::=<оператор присваивания>

<выражение>::=<арифметическая операция>
 <слагаемое1>::=<элемент массива>
 <индексное выражение>::=<идентификатор> | <целая константа>
 <слагаемое2>::=<вызов функции>
 <список параметров>::=<элемент массива>
 <индексное выражение>::=<идентификатор> | <целая константа>

Примеры допустимых цепочек

```

Myvar:=arr[i] + myfunc(arr[i]);
id:=arr[1] div myfunc(arr[2]);
id:=arr[-14] * MyFunc(arr[-8]);
  
```

Н. <цепочка>::=<описание типа>

<тип>::=**array** [<нижняя граница> ..<верхняя граница>] **of** <стандартный тип>
 <нижняя граница>::=<арифметическое выражение>
 <слагаемое1>::=<целая константа>
 <слагаемое2>::=<идентификатор>
 <верхняя граница>::=<нижняя граница>

Примеры допустимых цепочек

```

type arr = array [1+Min..1+Max] of Char;
type arr = array [100 div Rank ..-1+Top] of Word;
type Arr = array [3*Small..2000-Big] of Byte;
  
```

О. <цепочка>::=<оператор while-do>

<условие>::=<идентификатор><операция сравнения><идентификатор>
 <оператор>::=<оператор присваивания>
 <выражение>::=<элемент массива>
 <индексное выражение>::=<арифметическая операция>
 <слагаемое1>::=<16-ричная константа>
 <слагаемое2>::=<целое без знака>

Примеры допустимых цепочек

```

while Z<M do Z:=A[$A0+1];
while id>=Max do id2:=Mem[$A10 mod 2];
while id1<>id2 do id1:=Arr[$FFFF-id2]
  
```

Р. <цепочка>::=<условный оператор>

<условие>::=<арифметическая операция><операция сравнения><идентификатор>
 <слагаемое1>::=<идентификатор>
 <слагаемое2>::=<целое без знака>
 <оператор1>::=<вызов подпрограммы>
 <список параметров>::=<целая константа>
 <оператор2>::=<оператор1>

Примеры допустимых цепочек

```
if A+1>Max then Proc(1);
if id mod 2 <> Top then Proc(-2);
if id div 2 <> Bot then Proc(-22) else Proc(123);
```

Q. <цепочка>::=<оператор repeat-until>

<оператор>::=<вызов подпрограммы>
<список параметров>::=<арифметическое выражение>
<слагаемое1>::=<16-ричная константа>
<слагаемое2>::=<идентификатор>
<условие>::=<идентификатор><операция сравнения><целая константа>

Примеры допустимых цепочек

```
repeat func($F + i) until A>3;
repeat func($123d div min) until St<-1;
repeat func($ABCD * 3) until Z<>-123;
```

R. <цепочка>::=<оператор присваивания>

<выражение>::=<арифметическая операция>
<слагаемое1>::=<элемент массива>
<индексное выражение>::=<вызов подпрограммы>
<список параметров>::=<целая константа>
<слагаемое2>::=<слагаемое1>

Примеры допустимых цепочек

```
Z:=A[f(1)]+B[f(2)];
myvar:=Arr[func(12)] div Arr2[func(33)];
myvar:=Arr[func(-123)] * Arr2[func(-75)];
```

S. <цепочка>::=<оператор while-do>

<условие>::=<логическая константа>
<оператор>::=<оператор if-then>
<условие>::=<идентификатор><операция сравнения><16-ричная константа>
<оператор1>::=<вызов подпрограммы>
<список параметров>::=<список идентификаторов>

Примеры допустимых цепочек

```
while false do if A>$a then p(i);
while true do if MyVar<>$FFFF then Snooze(param);
while True do if Check<>$1F then Kill(p1, p2, p3, p4);
```

T. <цепочка>::=<оператор if-then>

<условие>::=<вызов подпрограммы>
<список параметров>::=<арифметическая операция>
<слагаемое1>::=<целая константа>
<слагаемое2>::=<идентификатор>
<оператор1>::=<оператор repeat-until>
<оператор>::=<вызов подпрограммы>
<список параметров>::=<логическая константа>
<условие>::=<вызов подпрограммы>
<список параметров>::=<логическая константа>

Примеры допустимых цепочек

```
if Cond(1+b) then repeat f(false) until g(true);
if Func(-1 mod k) then repeat Proc(True) until Stop(False);
if F(12*a) then repeat DoAnything(True) until Boring(False);
```

U. <цепочка>::=<оператор repeat-until>

<оператор>::=<вызов подпрограммы>
<список параметров>::=<целая константа>
<условие>::=<арифметическая операция><операция сравнения><идентификатор>
<слагаемое1>::=<элемент массива>
<индексное выражение>::=<арифметическая операция>
<слагаемое1>::=<идентификатор>
<слагаемое2>::=<слагаемое1>
<слагаемое2>::=<слагаемое1>

Примеры допустимых цепочек

```
repeat f(1) until A[i+j]+B[i+j]>Max;  
repeat func(12) until Arr[Min+Max] div Arr[Top+Bot]<Zero;  
repeat func(-12) until Arr[Min+Max] div Arr[Top+Bot]<>Zero;
```

V. <цепочка>::=<условный оператор>

<условие>::=<вызов подпрограммы>
<список параметров>::=<арифметическая операция>
<слагаемое1>::=<элемент массива>
<индексное выражение>::=<целая константа>
<слагаемое2>::=<слагаемое1>
<оператор1>::=<вызов подпрограммы>
<список параметров>::=<идентификатор>
<оператор2>::=<оператор1>

Примеры допустимых цепочек

```
if func(A[1] div A[2]) then proc(id);  
if func(Arr[10] * Arr[12]) then proc(id);  
if func(Arr[-21] mod Arr[-30]) then proc(id1) else proc(id2)
```

W. <цепочка>::=<описание константы>

<значение>::=<арифметическая операция>
<слагаемое1>::=<целое без знака> | <арифметическая операция>
<слагаемое2>::=<слагаемое1>

Примеры допустимых цепочек

```
const A=1+2;  
const id=123 div 2 mod 3;  
const Expr=10 * 20 div 10 mod 15 - 44 + 112;
```

X. <цепочка>::=<описание константы>

<значение>::=<целая константа> | <16-ричная константа> | <вещественная константа>
| <строковая константа>
<строковая константа>::='<цифра> | <идентификатор> | <цифра><идентификатор>'

Примеры допустимых цепочек

```
const Error=$FFFF;  
const Eps=1E-3;  
const OneHndNeg=-100;  
const Str='0123456789abcdefgh';
```

Список рекомендуемой литературы

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

2. *Ван Тассел Д.* Силь, разработка, эффективность, отладка и испытание программ. - М.: Мир, 1985.
3. *Льюис Ф., Розенкранц Д., Сирнз Р.* Теоретические основы проектирования компиляторов. - М.: Мир, 1979.
4. *Фаронов В.В.* Программирование на персональных ЭВМ в среде Turbo Pascal. - М.: Изд.-во МГТУ, 1995.
5. *Фуксман А.А.* Технологические аспекты создания программных систем. - М.: Статистика, 1979.
6. *Bell D., Morrey I., Pogh J.* Software Engineering. A programming Approach. Prentice Hall, 1992.