МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

**В.Г. Кошкидько, А.И. Панычев,
П.А. Дятлов**

# ПРАКТИКУМ ПО ИНФОРМАТИКЕ

**Часть 1**

**Учебное пособие**

**Ростов – на – Дону
Издательство Южного федерального университета
2014**

УДК 681.3х5(075.8)

**Рецензенты:**

кафедра социальной педагогики и психологии Таганрогского государственного педагогического института; заведующий кафедрой кандидат педагогических наук, профессор В.В. Гура;

кандидат химических наук, доцент Таганрогского политехнического института (филиал Донского государственного технического университета) А.Д. Телеш.

Кошкидько В.Г., Панычев А.И., Дятлов П.А. Практикум по информатике. Часть 1: учебное пособие. – Ростов – на - Дону: Изд-во ЮФУ, 2014. – 69 с.

В учебном пособии описывается современный практикум, состоящий из шести практических работ, направленный на освоение приемов математического и компьютерного моделирования, основ алгоритмизации и программирования на примере среды программирования Delphi.

Ил. 10. Табл. 3. Библиогр.: 15 назв.

УДК 681.3х5(075.8)

# ВВЕДЕНИЕ

Учебное пособие предназначено для студентов, обучающихся по направлениям «Телекоммуникации» и «Радиотехника» и изучающих дисциплины «Информатика» и «Прикладная информатика».

Практикум включает работы, позволяющие освоить основные этапы решения технических научных и инженерных задач с использованием компьютерной техники. Отличительной особенностью учебного пособия является его направленность на решение задач, относящихся к областям радиотехники и телекоммуникаций. В качестве объектов исследования предложены электрические цепи, в качестве конкретных задач – расчеты электротехнических параметров, таких как коэффициент передачи, временные диаграммы, нагрузочные характеристики и другие. В частности, задачи, направленные на изучение линейных, ветвящихся и циклических алгоритмов, демонстрируют возможность постановки различных исследовательских задач на одном объекте исследования – одной и той же электрической цепи.

Помимо теоретических сведений по каждому разделу практикум содержит руководства к выполнению практических работ, методические рекомендации по выполнению домашнего и лабораторного заданий, контрольные вопросы для проверки уровня знаний и самоконтроля студентов.

Основная цель практикума – закрепление и углубление теоретических знаний, полученных студентами при изучении дисциплины, приобретение практических навыков алгоритмизации и программирования.

Практикум обеспечивает возможность индивидуализации обучения студентов. Каждый студент имеет возможность самостоятельно проработать изучаемый материал, закрепить свои знания с помощью контрольных вопросов по каждой теме.

Многие вопросы, лишь кратко упомянутые в теоретических разделах, представляют существенный интерес для заинтересованного читателя. Помочь в самостоятельном углубленном изучении материала призван библиографический список с аннотациями.

# 1. ИСПОЛЬЗОВАНИЕ КОМПЬЮТЕРНОЙ ТЕХНИКИ ДЛЯ РЕШЕНИЯ ИНЖЕНЕРНЫХ И НАУЧНЫХ ЗАДАЧ

Решение научных и инженерных технических задач немыслимо без использования компьютерной техники. Применение вычислительной техники позволяет значительно ускорить обработку больших информационных массивов. Компьютеры способны быстро выполнять действия с числовыми данными и являются эффективным помощником исследователя, но никак не могут заменить человека. Компьютер работает по программе, созданной человеком, и, следовательно, именно от человека зависит эффективность использования компьютерной техники.

В общем случае решение любой технической задачи состоит из нескольких этапов, каждый из которых важен и не может быть пропущен. Основные этапы решения задач с помощью компьютерной техники следующие:

1. Выбор объекта исследования.
2. Постановка задачи.
3. Составление или выбор модели задачи.
4. Разработка алгоритма решения задачи.
5. Составление вычислительной программы.
6. Компьютерный эксперимент.

Рассмотрим подробно содержание каждого из перечисленных этапов.

**1. Выбор объекта исследования.** Выбор объекта исследования, как правило, диктуется интересами дальнейшего развития науки и техники и производится руководителем научно-исследовательского подразделения или, при достаточном опыте, может выполняться исследователем самостоятельно.

**2. Постановка задачи.** На одном объекте исследования могут быть сформулированы различные *физические задачи*. Например, если объектом исследования является некоторая электрическая цепь, то различными физическими задачами исследования для нее могут быть расчет коэффициента передачи этой цепи, расчет зависимостей напряжения на выходе от параметров схемы, исследование частотных характеристик, анализ переходных процессов и так далее.

Задача может быть корректной (хорошо поставленной) или некорректной (плохо поставленной). В хорошо поставленной задаче не должно быть лишних данных, затрудняющих понимание сути проблемы. С другой стороны, исходных данных не должно быть недостаточно – иначе проблема не может быть решена. Особо важно четко определить, что требуется получить, в какой форме и в каком количестве. Результат может быть представлен в виде набора чисел, таблиц или графиков, текстовой информации, графического изображения и так далее.

Умение ставить задачу – умение, приходящее с опытом. Хорошо поставленная задача – это уже половина ее решения.

**3. Составление или выбор модели объекта.** *Моделью* называется образ реального объекта (процесса), выраженный в материальной или идеальной форме и отражающий существенные свойства исследуемого объекта (процесса). Важнейшим условием использования модели является ее адекватность, т.е. соответствие реальному объекту по тем свойствам, которые наиболее существенны для данной задачи исследования. Следовательно, в зависимости от конкретной задачи исследования для одного и того же объекта могут использоваться различные модели.

Если реальный объект заменяется своим образом в материальной форме, то полученная модель называется *натурной*. Обычно она представляет собой масштабированный (геометрически подобный) макет исходного объекта.

Если реальный объект заменяется своим образом в идеальной форме, то полученная модель называется *мысленной*. Обычно она представляет собой описание объекта с помощью лингвистического языка, рисунков, графиков, формул, уравнений, неравенств и тому подобное. При использовании компьютеров чаще всего применяются именно мысленные модели.

Таким образом, на этапе составления или выбора модели исходная физическая задача «упрощается», при этом во внимание принимаются не все свойства объекта, а лишь некоторые его свойства, являющиеся наиболее существенными в данном случае. Для составления модели необходимо высказать предположения, которые позволяют из всего разнообразия информации об изучаемом объекте выделить *исходные данные*, определить, что будет служить *ре-*

*зультатом* и какова *связь* между исходными данными и результатом.

Любая задача решается определенным исполнителем, поэтому модель задачи должна составляться в расчете на этого исполнителя – исполнитель должен *понять* и *уметь выполнить* необходимые действия. Если исполнителем решения является компьютер, то используемую модель называют *компьютерной моделью*. Если исполнитель умеет только вычислять, то для него необходимо составить модель, в которой исходные данные и результат представляются в виде чисел, а связь между ними – в виде математических соотношений. Такая модель называется *математической моделью*. Практически всегда компьютерные модели составляются на основе математических моделей.

Поскольку в математических моделях используются числа, формулы, уравнения и другие математические соотношения, то одна и та же математическая модель может соответствовать различным физическим задачам. По этой причине составление математической модели называется *формализацией* задачи, а сама модель – *формальной постановкой* задачи.

Пояснить смысл формальной постановки задачи можно следующим примером. Пусть математическая модель сформулирована так: исходные данные – два числа $a$ и $b$; предполагаемый результат – число $c$; связь между данными и результатом задана соотношением $c = a / b$. Если под числом $a$ понимать напряжение на участке электрической цепи, а под числом $b$ – сопротивление этого участка, то результатом $c$ будет ток на этом участке цепи. Если же под числом $a$ понимать, например, массу собранного с поля урожая пшеницы, а под числом $b$ – площадь этого поля, то результатом $c$ будет урожайность пшеницы. Нетрудно предложить и другие физические задачи, имеющие такую же математическую модель.

Очевидно, что при составлении модели можно ошибиться, поэтому выбор модели – это всегда гипотеза, которая проверяется в процессе исследования. Очень важно уяснить: поскольку исходная задача заменяется моделью, то полученный результат будет относиться именно к модели. А поскольку модель лишь приближенно описывает исходный объект, то и полученный результат всегда будет *приближенным*.

Для одной задачи можно составить разные модели и, соответственно, получить различные решения. Эти решения будут несовпа-

дающими, но, тем не менее, справедливыми в рамках своей модели. Точность решения определяется точностью использованной модели.

**4. Разработка алгоритма решения задачи.** *Алгоритм* – это система точных и понятных предписаний о содержании и последовательности выполнения конечного числа действий, необходимых для получения результата.

Составленный для решения задачи алгоритм должен быть ориентирован на конкретного исполнителя (компьютер, робот, человек) и обладать следующими свойствами.

- *Дискретность*. Процесс получения результата разбивается на последовательность отдельных шагов, и только выполнив один из них, можно переходить к следующему.
- *Понятность*. Каждый исполнитель алгоритма имеет свою систему команд, которые он способен понять и исполнить, поэтому предписания алгоритма должны быть из этой системы команд.
- *Детерминированность (определенность)*. Ни одна команда не должна содержать указаний, смысл которых может быть воспринят исполнителем неоднозначно, то есть алгоритм не должен оставлять исполнителю место для произвола.
- *Результативность*. Процесс выполнения алгоритма должен прекратиться за конечное число шагов, и при этом должен быть получен какой-либо определенный результат.

Для получения результата используются следующие виды алгоритмов.

*Линейный алгоритм*. В этом случае последовательность операций при исполнении совпадает с порядком их следования в записи алгоритма и не зависит ни от чего.

*Ветвящийся алгоритм*. В этом случае на некотором шаге процесса исполнения дальнейшее выполнение алгоритма может продолжиться по одному из двух или более путей в зависимости от истинности или ложности некоторого условия.

*Циклический алгоритм*. Используется для многократного выполнения одного и того же набора действий, называемого *тело цикла*, как правило, с несколько измененными параметрами.

Алгоритм можно записать различными способами. Один из вариантов – словесное описание отдельных шагов, обычно пронумерованных в порядке их выполнения. Однако наиболее наглядным

является графическое представление алгоритма, называемое блок-схемой. В этом случае каждая команда представляется в виде некоторой геометрической фигуры, внутри которой записывается содержание выполняемого действия. Переходы от команды к команде изображаются с помощью линий связи, а направление перехода указывается стрелкой.

Овалом обозначаются начало и конец алгоритма (рис. 1.1).

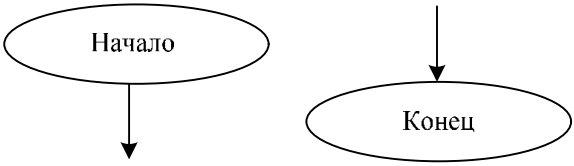



Рис. 1.1. Элементы блок-схемы алгоритма «Начало» и «Конец»

Команды ввода данных с клавиатуры или из файла на носителе информации и вывода данных на дисплей монитора обозначаются параллелепипедом (рис. 1.2).

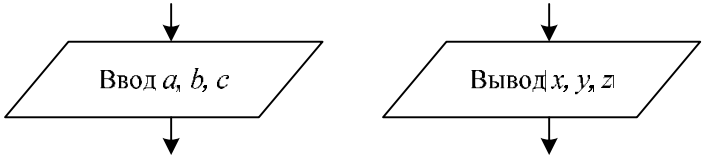



Рис. 1.2. Элементы блок-схемы алгоритма «Ввод» и «Вывод»

Печать значений переменных обозначается фигурой, представленной на рис. 1.3..

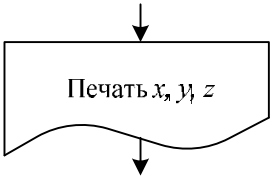



Рис. 1.3. Элемент блок-схемы алгоритма «Печать»

Основным элементом линейного алгоритма является элемент общей обработки (арифметическое предписание), изображаемый в виде прямоугольника. Он имеет один вход и один выход (рис. 1.4).

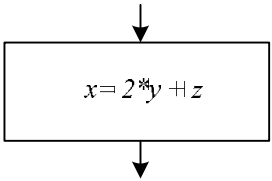



Рис. 1.4. Элемент общей обработки
блок-схемы алгоритма

Условие, входящее в состав ветвящегося алгоритма, называется элементом принятия решения (логическим предписанием, развилкой) и представляется в виде ромба, имеющего один вход и два выхода. Выход, соответствующий истинному значению условия, обозначается «да». Выход, соответствующий ложному значению условия, обозначается «нет». Если по каждому из этих выходов предусмотрены некоторые действия, то элемент алгоритма называется *ветвлением в полной форме* (рис. 1.5,а). Если же выполнение некоторых команд предусмотрено только по одному из выходов, то алгоритм называется *ветвлением в неполной форме* (рис. 1.5,б).

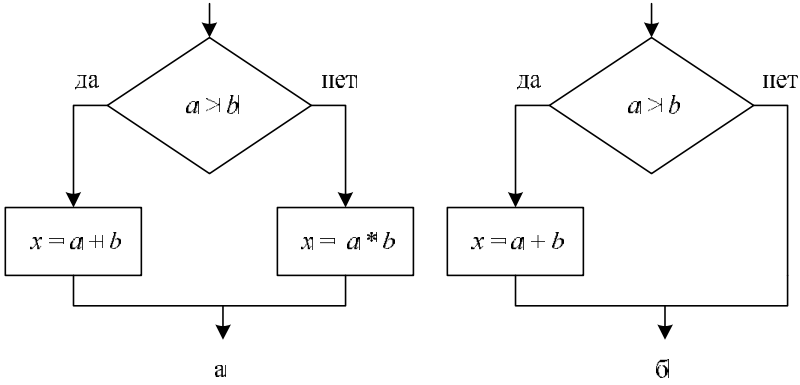



Рис. 1.5. Блок-схемы ветвящихся алгоритмов:
а – ветвление в полной форме;
б – ветвление в неполной форме

Циклические алгоритмы могут быть двух видов. Первый вид называется *условным циклом* и используется, когда заранее неизвестно число повторов выполнения тела цикла. Условие, разрешающее или запрещающее очередное повторение цикла, может находиться как перед телом цикла, так и после него. В первом случае цикл на-

зывается циклом *с управляемым началом* (рис. 1.6,а), во втором – *с управляемым окончанием* (рис. 1.6,б).

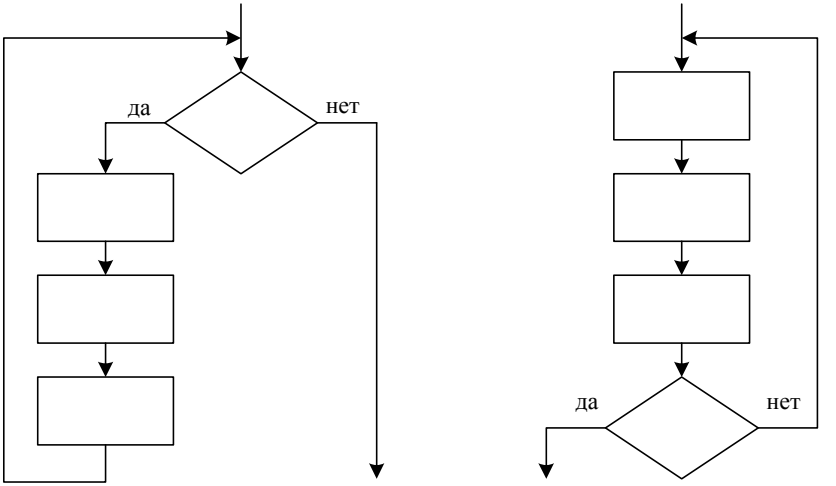



Рис. 1.6. Блок-схемы условных циклических алгоритмов:
а – с управляемым началом; б – с управляемым окончанием

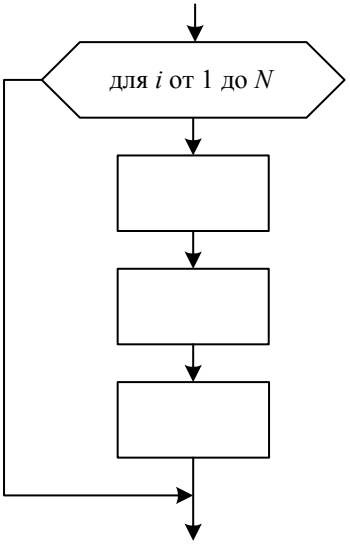



Рис. 1.7. Блок-схема безусловного циклического алгоритма

Второй вид циклического алгоритма называется *безусловным циклом* (циклом с параметром) и используется, когда число повторов известно заранее. Блок-схема такого цикла представлена на рис. 1.7.

В общем случае блок-схема алгоритма представляет собой комбинацию всех перечисленных элементов.

Составление алгоритма решения задачи является важнейшим этапом. Как правило, для решения одной и той же задачи можно разработать несколько правильных алгоритмов. Основная сложность со-

стоит в разработке наиболее эффективного алгоритма, позволяющего с меньшими затратами ресурсов компьютера получить результат за минимальное время.

**5. Составление вычислительной программы.** Вычислительная программа является ни чем иным, как записью алгоритма с помощью средств некоторой системы программирования. Для технических расчетов применяются такие языки и системы программирования, как Delphi, Pascal, C, Fortran, MathCAD и другие. Программа позволяет записать алгоритм в виде, понятном оборудованию компьютера.

Практически всегда по одному и тому же алгоритму можно составить несколько разных программ, дающих правильный результат. Основная задача этого этапа состоит в наиболее эффективном использовании возможностей языка программирования для реализации разработанного алгоритма.

**6. Компьютерный эксперимент.** Этот этап включает в себя несколько важных моментов. Во-первых, необходимо добиться работоспособности составленной ранее программы и получить *результат*. Во-вторых, обязательно надо выполнить *тестирование* результатов. Тестирование подразумевает сопоставление полученных результатов с экспериментальными данными, теоретическими воззрениями и другой информацией. Только в случае совпадения рассчитанного по программе результата с соображениями, полученными из внешних дополнительных источников, можно считать работу программы правильной.

Эффективным приемом, позволяющим провести тестирование, является задание некоторых характерных входных данных, при которых результат можно предсказать только на основе физических соображений, без вычислений. Обычно при этом численное значение результата предсказывается качественно, например: он должен быть положительным (или, напротив, отрицательным), или он должен быть очень маленьким ($\approx 0$), или он должен быть очень большим ($\approx \infty$) и так далее. Проверка правильности полученного результата с помощью расчетов другими средствами (вручную, с помощью калькулятора или другого языка программирования) малоэффективна, так как заложенная, например, в математической модели ошибка в этом случае не может быть выявлена.

Выполнение всех перечисленных этапов при решении задач с помощью ЭМВ, как правило, носит итерационный циклический характер, который можно изобразить в виде логической схемы решения технических задач с помощью компьютера (рис. 1.8).

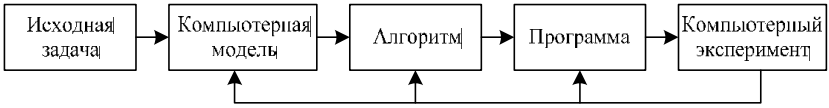



Рис. 1.8. Логическая схема решения задач с помощью компьютера

В случае неудовлетворительных результатов тестирования следует возвратиться к предыдущим этапам и устранить ошибки.

Прежде всего проверяется правильность программы. С ней могут быть связаны ошибки двух видов. Первые называются синтаксическими ошибками и являются следствием неправильного использования операторов языка программирования. Они, как правило, легко устраняются. Ошибки второго вида возникают на этапе проведения расчетов (исполнения программы) и чаще всего вызваны попыткой выполнения запрещенных операций, таких как деление на ноль, выход за пределы значений индекса массива, переполнение разрядной сетки и так далее. Ошибки такого типа обычно устраняют, вводя в текст программы проверку всех величин, участвующих в потенциально опасных операциях. Можно также попытаться изменить входные данные.

Когда правильность программы не вызывает сомнений, а результаты тестирования по-прежнему остаются неудовлетворительными, следует проверить алгоритм. В случае его корректировки, разумеется, надо внести соответствующие изменения в расчетную программу.

Ошибки также могут содержаться в математической и компьютерной моделях. Найти и устранить такие ошибки наиболее трудно. Главными помощниками в этом могут стать теория и физический эксперимент. В случае изменения модели необходимо внести соответствующие корректировки в алгоритм и программу.

Этапы от составления модели до компьютерного эксперимента могут повторяться несколько раз до тех пор, пока эксперимент и теория не совпадут с достаточной для практики точностью. Напомним, решение всегда является приближенным, справедливым в рамках выбранной модели.

## ПРАКТИЧЕСКАЯ РАБОТА № 1

### 1.1. Цель работы

Приобретение навыков работы с компьютером, освоение основных команд операционной системы. Ознакомление с организацией среды программирования Delphi, создание и исполнение программ средствами Delphi. Составление краткого конспекта основных команд.

### 1.2. Задача работы

Разработать программу для расчета времени подъема на высоту $h$ тела, брошенного вертикально вверх с начальной скоростью $v_0$.

### 1.3. Решение задачи

В настоящей работе студенты не выполняют формализацию задачи, вывод основных расчетных соотношений, разработку алгоритма, программы и тестов. Результаты этих этапов решения задачи приведены ниже.

**Объект исследования.** Объектом исследования является брошенное вертикально вверх тело.

**Постановка задачи.** Необходимо рассчитать время подъема тела на заданную высоту.

**Формулировка математической модели.** Как известно из механики, в качестве исходных данных в поставленной задаче могут быть использованы следующие величины: начальная скорость $v_0$ и высота подъема $h$. Необходимо рассчитать и вывести на экран монитора время, за которое тело поднимется на эту высоту.

Вывод математических соотношений произведем при следующих допущениях: тело является материальной точкой, сопротивлением воздуха пренебрегаем. При указанных допущениях высота подъема $h$ может быть найдена по формуле $h = v_0 t - \frac{gt^2}{2}$, где $g$ = 9,80665 м/с – ускорение свободного падения; $t$ – время, равное нулю в момент начала движения.

Для нахождения времени подъема $t$ получаем квадратное уравнение $\frac{g}{2}t^2 - v_0 t + h = 0$, откуда и найдем искомое время $t$:

$$t_1 = \frac{v_0 - \sqrt{v_0^2 - 2gh}}{g};\ t_2 = \frac{v_0 + \sqrt{v_0^2 - 2gh}}{g}.$$

Таким образом, на заданной высоте *h* тело в общем случае находится дважды: когда оно поднимается вверх после броска и когда падает вниз.

**Алгоритм решения задачи**. Алгоритм может быть представлен в виде следующей блок-схемы (рис. 1.9).

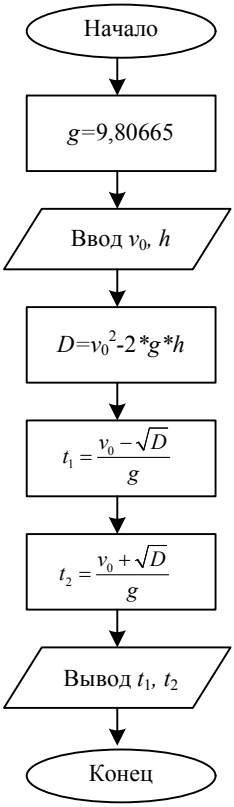



Рис. 1.9. Блок-схема алгоритма решения задачи

**Программа для решения задачи**. Программа, подготовленная средствами Delphi, может быть записана следующим образом.

```
PROGRAM TIME;
Const g = 9.80665;
Var h, v0, D, t1, t2 : real;
Begin
    Writeln('Расчет времени подъема тела');
    Writeln;
    Write('Начальная скорость (м/с):'); Readln(v0);
    Write('Высота подъема (м):'); Readln(h);
    D:=Sqr(v0) – 2.0*g*h;
    t1:=(v0 – Sqrt(D))/g;
    t2:=(v0 + Sqrt(D))/g;
    Writeln;
    Write('Время подъема t1= ', t1, ' t2= ', t2);
    Readln
End.
```

**Тесты.** Тесты должны помочь убедиться в правильности работы программы. Оформим их в виде табл. 1.1. Тесты необходимо подбирать так, чтобы результат можно было предсказать из физических соображений без расчетов или легко рассчитать вручную. Для этого надо включить в рассмотрение частные случаи: тело находится на земле, высота стремится к бесконечности, и так далее.

Таблица 1.1

Тесты

| Исходные данные | | Результаты | |
|---|---|---|---|
| $v_0$ | $h$ | Ожидается | Получено |
| 20 | 0 | $t_1$=0 $t_2$=? | |
| 0 | 0 | $t_1$ и $t_2$ – любые | |
| 20 | 10 | $t_1$=? $t_2$=? | |

В графу «Ожидается» записываются результаты, которые предсказаны из физических соображений, а в графу «Получено» заносятся результаты, полученные путем компьютерных расчетов.

### 1.4. Домашнее задание

Оформить протокол выполнения лабораторной работы (см. приложение), содержащий конспект основных команд среды программирования Delphi, производящих редактирование текста програм-

мы и запуск программы на исполнение.

### 1.5. Лабораторное задание

1. Активизировать среду программирования Delphi и создать текст программы, приведенной в подразд. 1.3. Сохранить текст программы в файле на носителе информации.

2. Вызвать программу для исполнения. Выполнить тестирование программы и записать результаты в табл.1.1. Последовательно рассчитать время подъема тела на заданную высоту при начальной скорости $v_0 = 15$ м/с и нескольких значениях высоты $h = 2, 5, 10, 15$ м.

Убедиться, что при $h = 15$ м работа программы завершается аварийно с сообщением об ошибке. Разобраться в причине ошибки.

3. Выполнить корректировку исходного текста программы с целью улучшения ее качества:

- изменить строку
  *Writeln('Расчет времени подъема тела');*

  на строку
  *Writeln('Модифицированная программа');*
- после строки
  *D:=Sqr(v0)-2.0*g*h;*

  вставить строку
  *if D>0 then begin*
- перед строкой
  *Readln*

  вставить строки
  *end*
  *else*
  *Writeln('Высота h недостижима');*
- переставить строки
  *t1=(v0-Sqrt(D))/g;*
  *t2:=(v0+Sqrt(D))/g;*

  вставив их между строками
  *Writeln;* и *Write('Время подъема t1= ', t1, ' t2= ', t2);*
- сохранить этот вариант текста программы с новым именем.

4. Повторить расчеты в соответствии с п. 2 с помощью преобразованной программы, проверить ее работоспособность для случая $h = 15$ м.

5. Распечатать текст модифицированной программы для отчета.

### 1.6. Содержание отчета

1. Название, цель и задача работы.
2. Краткий конспект основных команд среды программирования Delphi, производящих редактирование тектса программы и ее исполнение.
3. Листинг текста модифицированной программы.

### Контрольные вопросы

1. Перечислите основные этапы решения технических задач с помощью компьютера и дайте их краткую характеристику.
2. Поясните содержание этапов «Выбор объекта исследования» и «Постановка задачи» при решении задачи с помощью компьютера.
3. Дайте определение модели реального объекта. Перечислите виды моделей.
4. Дайте определение алгоритма. Перечислите свойства алгоритма.
5. Дайте определение алгоритма. Перечислите виды алгоритмов.
6. Дайте определение линейного и ветвящегося алгоритма. Перечислите их виды.
7. Дайте определение циклического алгоритма. Перечислите его виды.
8. Перечислите основные элементы блок-схемы алгоритма.
9. Приведите варианты блок-схем для ветвящихся алгоритмов.
10. Приведите варианты блок-схем для циклических алгоритмов.
11. Поясните смысл и содержание тестирования расчетных программ.
12. Поясните логическую схему решения задач с помощью компьютера.

# 2. ЛИНЕЙНЫЕ АЛГОРИТМЫ

*Линейный алгоритм* представляет собой последовательность операций, порядок выполнения которых не зависит ни от каких условий. Реализация линейного алгоритма средствами среды программирования Delphi заключается в составлении программы, в которой операторы, записанные в соответствии с требованиями языка, размещаются один за другим в порядке их выполнения.

Рассмотрим структуру программы на языке Delphi. В начале программы находится *заголовок*, состоящий из зарезервированного слова *Program* и имени программы (например, *Program Test;)*. Он несет смысловую нагрузку, и рекомендуется его выбирать таким, чтобы быстро распознавать нужную программу среди других.

Далее следует *программный блок*, содержащий в общем случае:

[раздел описания констант];

[раздел описания типов данных];

[раздел описания переменных];

[раздел описания процедур и функций];

раздел операторов.

В частном случае некоторые разделы, кроме раздела операторов, могут отсутствовать. Необязательные разделы указаны в квадратных скобках.

В разделе *описания констант* производится отождествление идентификаторов констант и постоянных значений, которые они обозначают. Раздел начинается зарезервированным словом *Const* (константа), за которым следует ряд пар, состоящих из идентификаторов и констант, соединенных знаком = (*равно*). Пары отделяются друг от друга точкой с запятой.

**Пример.**

```
Const
  pi = 3.141593;
  MAX = 100;
  Beg = 'Начало';
  code = #72;
  c = 'a';
```

После того как константа определена, ей нельзя присвоить другое значение.

*Раздел описания типов* данных начинается зарезервированным

словом *Type* (тип), за которым следует одно или несколько определений типов, разделенных точкой с запятой.

**Пример.**

```
Type
  Matric = array[1..20] of real;
  LatBukva = ('a'..'z');
  Days = 1..31;
```

Типы подразделяются на стандартные и определяемые пользователем. Стандартные типы *(integer, real, boolean, char* и другие) не требуют описания, в отличие от типов, определяемых пользователем.

*Раздел описания переменных* начинается зарезервированным словом *Var* (переменная), затем через запятую перечисляются имена переменных и через двоеточие указывается их тип. Каждая строка заканчивается точкой с запятой.

**Пример.**

```
Var
   x, y, z          : integer;
   Sum, Fq          : real;
   Progend          : boolean;
   C                : char;
```

В *разделе описания процедур и функций* помещаются подпрограммы пользователя. Стандартные процедуры и функции являются частью языка и могут вызываться без предварительного описания. Описание процедур и функций пользователя рассмотрено в разд. 5.

*Раздел операторов* в программе является основным, так как именно в нем с предварительно описанными переменными, константами, значениями функций выполняются действия, позволяющие получить результат, ради которого создавалась программа.

Раздел операторов начинается зарезервированным словом *Begin* (начало), далее следуют операторы, отделенные друг от друга точкой с запятой. Завершает раздел зарезервированное слово *End* (конец) и точка. Таким образом, структура раздела операторов имеет вид

```
Begin
   <оператор> ;
   <оператор> ;
```

……
<оператор>
*End.*

В программе, реализующей линейный алгоритм, операторы выполняются строго последовательно в том порядке, в котором они записаны в тексте программы в соответствии с синтаксисом и правилами пунктуации. Слова *Begin* и *End* являются аналогом открывающей и закрывающей скобки в обычных арифметических выражениях и иногда называются операторными скобками.

Таким образом, синтаксически программа состоит из заголовка и блока. Блок состоит из двух частей: описательной и исполнительной. Первая часть может отсутствовать, без второй части блок не имеет смысла. Описательная часть блока включает в себя разделы описания констант, типов, переменных, процедур и функций; исполнительная часть – раздел операторов. Наиболее часто в разделе операторов используются операторы присваивания и операторы ввода-вывода.

*Оператор присваивания* имеет вид

*A := B;*

где *A* – имя переменной, *B* – выражение. Оператор присваивания распознается по символу **:=** (*присвоить*) и предписывает вычислить значение выражения, заданного в его правой части, и присвоить результат вычисления переменной, указанной в левой части. Переменная и значение выражения должны иметь одинаковый тип.

**Пример.**

*Sort := 1;*
*Cena := 4.99*;
*Result := sin(A)+B*cos(C);*
*Loss := 'Потери';*
*c := 'x';*
*Find := x>y;*

*Операторы ввода-вывода* данных используются для передачи информации между оперативной памятью и внешними носителями информации, входящими в состав компьютерного оборудования (монитор, клавиатура, магнитные, оптические и электронные устройства долговременной памяти, принтер и другие устройства). Для выполнения операций ввода-вывода служат четыре оператора: *Read, Readln, Write* и *Writeln.*

Оператор чтения *Read* обеспечивает передачу в программу с системного устройства ввода числовых данных, символов, строк и так далее для текущей их обработки программой. Обычно системным устройством ввода служит клавиатура компьютера.

**Пример.**

```
Var I : real;
    J : integer;
    K : char;
Begin
  Writeln('Введите I, J ,K');
  Read(I ,J, K);
End.
```

В ходе выполнения этой программы на экран монитора выводится сообщение ”*Введите I, J, K”.* Дойдя до оператора *Read,* программа останавливается и ждет ввода значений для переменных *I*, *J* и *K*. Пользователь набирает нужные значения через один или несколько пробелов и нажимает клавишу <ENTER>.

Значения переменных должны задаваться в том порядке, в каком они указаны в операторе ввода (в нашем примере: *I, J, K).* Значения вводятся в соответствии с правилами записи констант языка Delphi. Если соответствие нарушено (например, *J* имеет целочисленный тип *integer*, а при вводе набирается значение символьного типа *char*), то возникает ошибка ввода-вывода.

Оператор чтения *Readln* аналогичен оператору *Read,* единственное отличие заключается в том, что после считывания последнего в списке значения для одного оператора *Readln* данные для следующего оператора ввода (*Read* или *Readln*) будут считываться с начала новой строки.

Оператор записи *Write* производит вывод числовых данных, строк и булевских значений на системное устройство вывода, которым обычно является дисплей.

**Пример.**

```
Var
  A, AB : real;
  B : integer;
Begin
  A := 5.43; B := 7; AB := A*B;
  write('A= ', A, ' B= ', B,' Произведение = ', AB)
End .
```

Оператор записи *Writeln* аналогичен оператору *Write,* но после вывода последнего в списке значения происходит перевод курсора к началу следующей строки. Оператор *Writeln,* записанный без параметров, вызывает перевод строки. Как указывалось выше, чрезвычайно важное значение имеет очередность ввода исходных данных. Для того чтобы не перепутать порядок ввода, целесообразно перед вводом данных вывести на дисплей пояснение, что же надо вводить. Оправданы смысловые пояснения, например «высота», «напряжение U1» и т.п., а не идентификаторы переменных, смысла которых не знает никто, кроме автора программы.

**Пример.**

```
Program Power;
Var
  U, I, P : real;
Begin
  Writeln('Напряжение ?'); Readln(U);
  Writeln('Ток ?'); Readln(I);
  P := U*I;
  Writeln('Мощность равна ', P,' Вт')
End.
```

## ПРАКТИЧЕСКАЯ РАБОТА № 2

### 2.1. Цель работы

Изучение структуры программы в среде Delphi, получение навыков ввода-вывода информации и выполнения простейших вычислений.

### 2.2. Задача работы

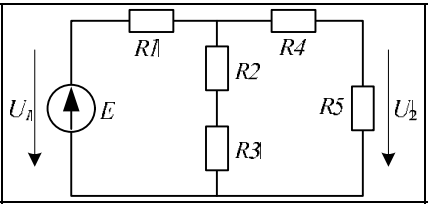
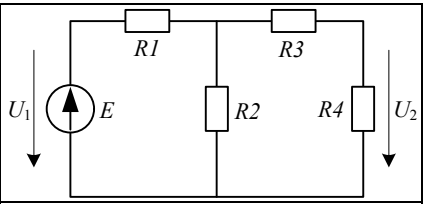
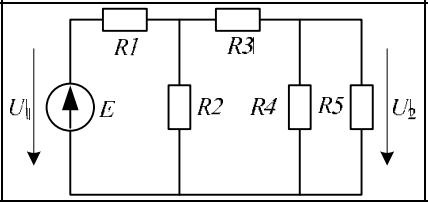
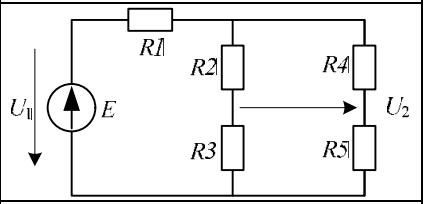
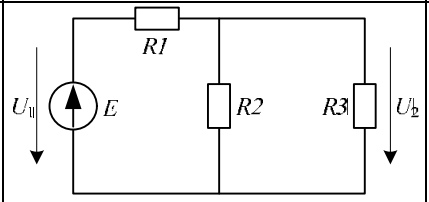
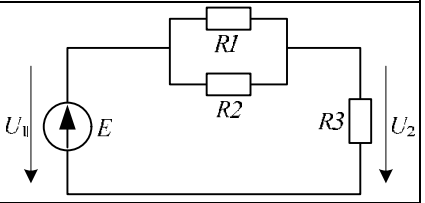
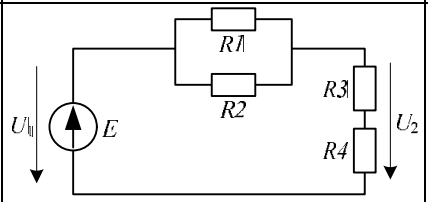
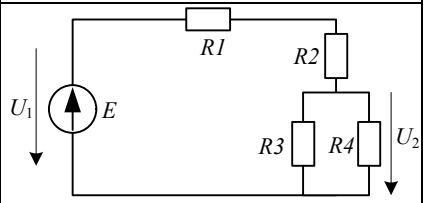
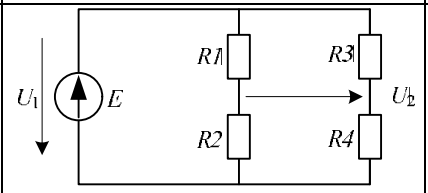
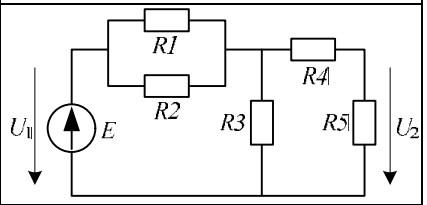
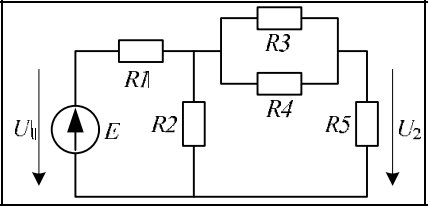
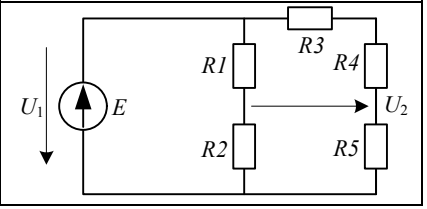
Рассчитать коэффициент передачи электрической цепи. Варианты схем цепей приведены в табл. 2.1. Значения сопротивлений и величину ЭДС *E* источника постоянного напряжения выбрать самостоятельно.

### 2.3. Домашнее задание

1. Изучить элементы среды программирования Delphi: алфавит и словарь языка, структуру программы, зарезервированные слова, стандартные идентификаторы, константы и переменные, типы данных, выражения, операнды, операторы, комментарии.

Таблица 2.1

Варианты заданий

| | | | |
|---|---|---|---|
| 1 |  | 2 |  |
| 3 |  | 4 |  |
| 5 |  | 6 |  |
| 7 |  | 8 |  |
| 9 |  | 10 |  |
| 11 |  | 12 |  |

2. Изучить основные операторы Delphi: оператор присваивания **:=**, операторы ввода *Read* и *Readln,* операторы вывода *Write* и *Writeln.*

3. Выполнить формализацию поставленной задачи. Выделить исходные данные и результаты расчета (обратить внимание на единицы измерения). Произвести математическую подготовку задачи.

Для выполнения расчетов необходимо получить выражение коэффициента передачи цепи *K*. Коэффициентом передачи цепи *K* называется отношение выходного напряжения цепи $U_2$ к входному напряжению $U_1$: $K=U_2 / U_1$. Величина входного напряжения во всех вариантах схем равна ЭДС источника постоянного напряжения: $U_1=E$. Для расчета $U_2$ можно использовать любые известные методы и законы (закон Ома, законы Кирхгофа, методы узловых напряжений и контурных токов).

Следует помнить, что при последовательном соединении сопротивлений *Ra* и *Rb* результирующее сопротивление равно их сумме *Ra+Rb*, а при параллельном соединении сопротивлений – величине *(Ra·Rb)/(Ra+Rb)*. Необязательно стремиться получить одну конечную (зачастую сложную) формулу для расчета искомой величины. Последовательность из нескольких более простых формул также может привести к нужному результату.

4. Разработать алгоритм и программу.

5. Составить тест-таблицу.

6. Подготовиться к ответам на контрольные вопросы.

### 2.4. Лабораторное задание

1. Создать и сохранить в файле текст программы.
2. Выполнить отладку и тестирование программы.
3. Получить и проанализировать результаты.

### 2.5. Содержание отчета

1. Название, цель и задача работы.
2. Формальная постановка задачи, расчетные выражения.
3. Алгоритм и программа для решения поставленной задачи.
4. Тест-таблица.
5. Текст отлаженной программы, анализ ошибок.

**Контрольные вопросы**

1. Напишите программу для вычисления дробной части среднего арифметического и среднего геометрического двух заданных положительных чисел $A$ и $B$.

2. Напишите программу для вычисления остатка от деления двух заданных целых положительных чисел $A$ и $B$.

3. Напишите программу для нахождения произведения цифр заданного четырехзначного целого числа.

4. Составьте программу для определения числа, полученного выписыванием в обратном порядке цифр заданного трехзначного целого числа.

5. Напишите программу, определяющую сумму двух первых и разность двух последних цифр заданного четырехзначного целого числа.

6. Напишите программу, определяющую куб суммы цифр заданного трехзначного целого числа.

7. Составьте программу, определяющую среднее арифметическое цифр заданного четырехзначного целого числа.

8. Напишите программу, вычисляющую периметр и площадь квадрата, вписанного в окружность с заданным радиусом $R$.

9. Составьте программу, вычисляющую периметр и площадь прямоугольного треугольника по длинам его катетов $a$ и $b$.

10. Дана сторона L равностороннего треугольника. Напишите программу, вычисляющую площадь этого треугольника.

11. Плацкартный билет на поезд до города N сейчас стоит $R$ рублей. За предыдущий год цена повышалась дважды – первый раз на 30%, а второй на 40%. Разработайте программу, вычисляющую стоимость билета: а) перед первым повышением цены; б) перед вторым повышением цены.

12. Напишите программу, вычисляющую площадь поверхности шара с заданным радиусом $R$.

13. Разработайте программу для вычисления длины окружности, площади круга и объема шара с одним и тем же радиусом $R$.

14. Составьте программу, предусматривающую ввод угла в градусах и вывод значения этого угла в радианах.

15. Составьте программу для решения квадратного уравнения $ax^2 + bx + c = 0$.

16. Сахарная свекла содержит 14% сахара. С 1 га собирают $X$ тонн сахарной свеклы. Составить программу, вычисляющую,

сколько гектаров надо засеять сахарной свеклой, чтобы получить $Z$ тонн сахара?

17. Напишите программу, которая вводит целые значения для $i$ и $j$ и выводит на экран значения $i^2, j^2, i^2 + j^2, i^2 - j^2$.

18. Напишите программу, которая для заданного целого числа $a$ выводит на экран следующую таблицу:

$$\begin{array}{ccc} a & & \\ a^3 & a^6 & \\ a^6 & a^3 & a \end{array}$$

19. Идет $k$-я секунда суток. Напишите программу, определяющую, сколько полных часов $h$ и полных минут $m$ прошло к этому времени (например, $h$=3 *и* $m$=40, если $k$=13 257=3·3600+40·60+57).

20. Разработайте программу работы кассового автомата, производящего выдачу сдачи при стоимости покупки $P$ рублей, если в автомат помещена сумма $S$ рублей. Сдача должна быть выдана минимальным количеством монет и купюр достоинством 1, 2, 5, 10, 50 и 100 рублей.

21. Для введенной суммы $S$ разработайте способ размена наименьшим числом монет и купюр достоинством 1, 2, 5, 10 и 50 рублей.

22. Разработайте программу для кассового автомата, которая бы вычисляла минимальное количество монет и купюр достоинством 1, 2, 5, 10, 50, 100 и 500 рублей, чтобы дать сдачу $m$ рублей.

23. Разработайте программу, вычисляющую необходимое целое количество банок краски для покраски цистерны цилиндрической формы. Исходные данные: диаметр $d$ и длина $h$ цистерны, расход краски (площадь поверхности, которую можно покрасить одной банкой краски) $w$.

24. Разработайте программу для вычисления производной функции $x^k$ в заданной точке $x=a$ ($a>0$). Значение $k$ задайте самостоятельно.

25. Разработайте программу для вычисления второй производной функции $1 - e^{4\cos(x)}$ в заданной точке $x=a$.

## 3. ВЕТВЯЩИЕСЯ АЛГОРИТМЫ

*Ветвящийся алгоритм* строится на основе условных операторов. Условные операторы обеспечивают выполнение или невыполнение некоторого оператора или блока в зависимости от заданных условий. Delphi допускает использование двух операторов условия: *if* и *case.*

*Оператор условия if* является одним из самых популярных средств, изменяющих естественный порядок выполнения операторов программы. Он может принимать одну из двух форм:

```
if <условие>                    if <условие>
  then                            then
     <оператор 1>                    <оператор>;
  else
     <оператор 2>;
```

Здесь ключевые слова *if, then, else* означают соответственно «*если», «то», «иначе»*. Элемент <*условие*> – это выражение булевского типа. Оно может быть простым или сложным. Сложные условия образуются с помощью логических операций *and, or, not («и», «или», «не»).* При записи условий могут использоваться допустимые операции отношения. Результат отношения всегда имеет булевский тип. Элемент <*оператор*> представляет собой простой или составной оператор (составной оператор обязательно должен быть охвачен операторными скобками *begin…end*).

Первая форма оператора *if* работает следующим образом: если условие имеет значение *TRUE*, то выполняется <*оператор 1*>, если же условие имеет значение *FALSE,* то выполняется <*оператор 2*>.

**Пример.**

```
Var
  A, B : real;
Begin
  A := 7;
  B := 10;
  if A>B
    then
      writeln ('A больше B')
    else
      writeln ('A меньше или равно B');
End.
```

В данном примере значение выражения *A>B* ложно, следовательно, на экране появится сообщение «*А меньше или равно В».*

При второй форме оператора *if* выполняется *<оператор>,* если результат условия равен *TRUE,* если же он имеет значение *FALSE* , то *<оператор>* не выполняется вообще, а выполняется оператор, следующий сразу за оператором *if.*

**Пример.**

```
Var
  A, B, C : integer;
Begin
  A := 4; B := 6; C := 1;
  if A>B then C:=A+B;
  C :=C+12; writeln('C= ', C)
End.
```

В результате на экране появится сообщение «*С= 13»*, так как условие *A>B* ложно *(FALSE)* и наращивания *C := A+B* не произойдет.

Оператор *if* может входить в состав другого оператора *if.* В таком случае говорят о *вложенности* условных операторов. Формат вложенного оператора имеет вид

```
if <условие 1>
  then
    if <условие 2>
      then
        <оператор 1>
      else
        <оператор 2>
  else <оператор 3> ;
```

С целью внесения ясности в текст программы рекомендуется каждый *then* и *else* записывать строго под тем *if,* к которому они относятся, а операторы записывать на один отступ (2-3 пробела) правее.

При вложенности оператора каждое *else* соответствует тому *then,* которое непосредственно ему предшествует.

**Пример.**

```
Var
  A, B : real;
  ZNAK : char;
```

```
Begin
  A := 70; B := 10.0;
  if A>B
    then ZNAK := '>'
    else if A = B
           then
             ZNAK := '='
           else
             ZNAK := '<';
  Write(A, ZNAK, B)
End.
```

Конструкций со степенью вложенности более двух-трех лучше избегать из-за сложности их анализа при отладке программы.

Приведенные выше формы записи оператора *if* являются общими. В частных случаях, когда используемые <*оператор 1*> и <*оператор 2*> короткие, можно записать короче и оператор *if*:

```
if <условие>
  then <оператор 1>
  else <оператор 2>;
```

или

```
if <условие> then <оператор 1>
                  else <оператор 2>;
```

или

```
if <условие> then <оператор 1> else <оператор 2>;
```

или

```
if <условие> then <оператор>;
```

Ниже приведены примеры использования оператора *if*:

• *с простым оператором*

```
if A>B then D := A+B
          else D := A–B;
```

• *с составным оператором*

```
if A<B
  then
    begin
      writeln ('А меньше В');
```

```
    D := A*B
  end
else
  begin
    writeln('A больше или равно B');
    D := A/B
  end;
```

• *с использованием сложного условия*

```
if (A=B) and (C=D)
  then
    begin
      writeln('Норма');
      F := 0
    end
  else
    begin
      writeln ('Превышение нормы');
      F: = 100
    end;
```

*Оператор выбора case* является обобщением оператора *if* и позволяет сделать выбор из произвольного числа имеющихся вариантов. Он состоит из выражения, называемого селектором, и списка параметров, каждому из которых предшествует список констант выбора (список может состоять и из одной константы). Как и в операторе *if,* здесь может присутствовать слово *else*, имеющее тот же смысл. Формат оператора *case* имеет вид

```
case <выражение-селектор> of
  <список 1>: <оператор 1>;
  <список 2>: <оператор 2> ;
  ……….
  <список N>: <оператор N>
else
  <оператор>
end;
```

Оператор *case* работает следующим образом. Сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того оператора, константа выбора которого равна текущему

значению селектора. Если ни одна из констант не равна текущему значению селектора, выполняется оператор, стоящий за словом *else.* Если слово *else* отсутствует, то выполняется оператор, находящийся за границей оператора *case*.

Селектор может иметь любой скалярный тип, кроме вещественного. Список констант выбора состоит из произвольного количества значений или диапазонов, отделенных друг от друга запятыми. Границы диапазона записываются двумя константами через разграничитель .. (две точки). Тип констант в любом случае должен совпадать с типом селектора.

Ниже приведены типичные формы записи оператора *case:*

• *селектор целочисленного типа*

```
Var
  i: integer;
……….....
Readln(i);
case  i  of
  1: z := i+ 10;
  2: z := i+100;
  3: z := i+ 1000
end;
```

• *селектор интервального типа*

```
Var
  i: integer;
……….....
Readln(i);
case  i  of
  1..10:      writeln('число ',i:4, ' в диапазоне 1–10');
  11..20:     writeln('число ', i:4,' в диапазоне 11–20');
  21..30:     writeln('число ', i:4,' в диапазоне 21–30');
else
  writeln('число ', i:4, ' вне пределов контроля')
end;
```

• *селектор перечисляемого типа*

```
Var
  season: (Winter, Spring, Summer, Autumn);
……….....
  case  season  of
```

```
    Winter:   Writeln ('Зима');
    Spring    Writeln ('Весна');
    Summer:   Writeln ('Лето');
    Autumn:   Writeln ('Осень')
  end;
```

• *селектор литерного типа*

```
  Var
    Name: Char;
  ……..
    Readln(Name):
    case  Name  of
      'A', 'a': Writeln('Вычисление абсолютной величины');
      'S', 's': Writeln('Вычисление синуса угла');
      'V', 'v': Writeln('Программа работу закончила')
  end;
```

## ПРАКТИЧЕСКАЯ РАБОТА № 3

### 3.1. Цель работы

Изучение операторов, изменяющих естественный порядок выполнения операторов программы, и приобретение навыков организации ветвящихся алгоритмов.

### 3.2. Задача работы

Найти напряжение $U_2$ на выходе цепи в произвольный момент времени $t$. Варианты схем электрических цепей приведены в табл. 3.1. На вход цепи воздействует источник гармонического напряжения $e(t) = E\,Cos(2\pi f t + \varphi)$, где $f$ – частота, $\varphi$ – начальная фаза. Значения сопротивлений, амплитуду $E$, частоту $f$ и начальную $\varphi$ гармонического напряжения источника выбрать самостоятельно.

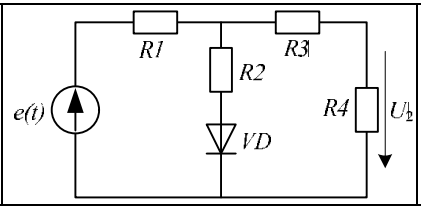
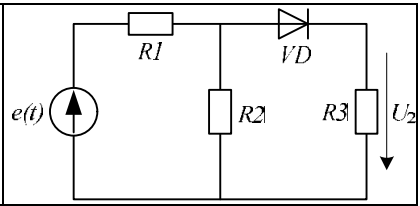
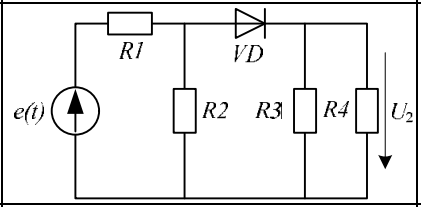
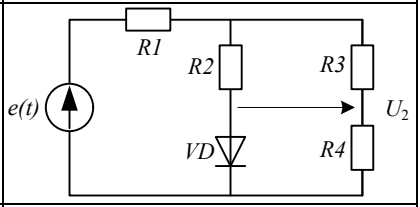
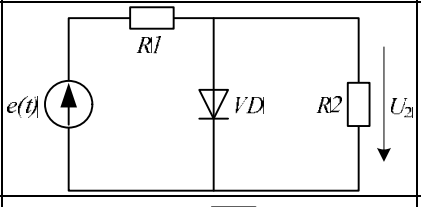
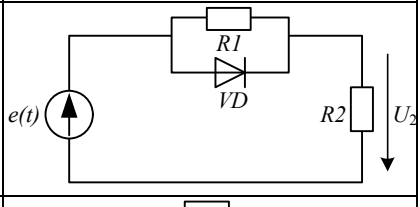
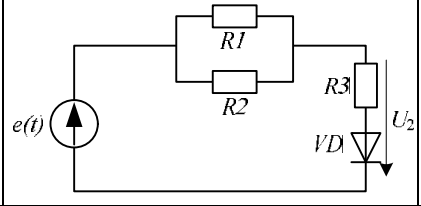
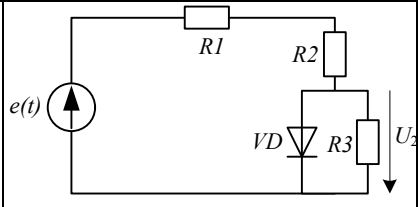
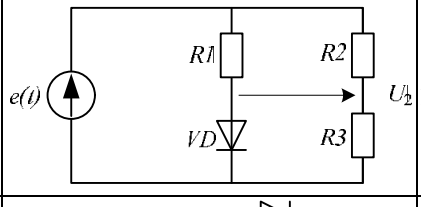
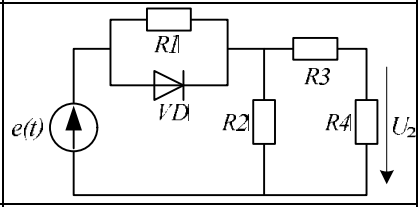
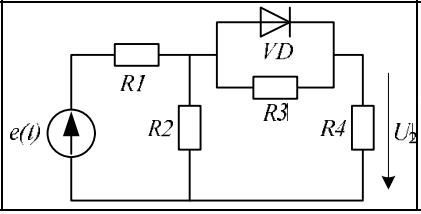
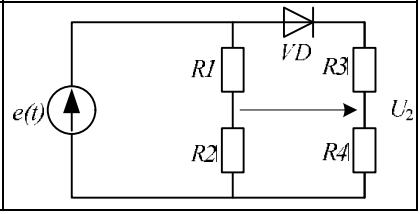
### 3.3. Домашнее задание

1. Изучить особенности булевского типа данных, операций отношения; операторов условия *if* и *case,* форматы оператора каждого вида.

2. Выполнить формализацию поставленной задачи. Выделить исходные данные и результаты расчета. Получить расчетные выражения. Вычислить значение периода $T$. Продумать использование единиц измерения.

Таблица 3.1

Варианты заданий

| | | | |
|---|---|---|---|
| **1** |  | **2** |  |
| **3** |  | **4** |  |
| **5** |  | **6** |  |
| **7** |  | **8** |  |
| **9** |  | **10** |  |
| **11** |  | **12** |  |

Особенности этой задачи следующие: 1) напряжение на входе

изменяется по гармоническому закону (по закону функции *cos)* и 2) в цепи присутствует нелинейный элемент – диод *VD*. Для упрощения расчетов рекомендуется принимать сопротивление диода равным нулю, если напряжение на аноде положительное (к диоду приложено прямое смещение), и бесконечности, если напряжение на аноде отрицательное (к диоду приложено обратное смещение). Также следует помнить, что при последовательном соединении сопротивлений *Ra* и *Rb* результирующее сопротивление равно их сумме *Ra+Rb*, а при параллельном соединении – величине *(Ra·Rb)/(Ra+Rb)*. Для расчета $U_2$ можно использовать любые известные законы и методы (закон Ома, законы Кирхгофа, методы узловых напряжений и контурных токов). Необязательно стремиться получить одну конечную (зачастую сложную) формулу для расчета искомой величины. Последовательность из нескольких более простых формул также может привести к нужному результату.

3. Разработать алгоритм и программу для решения поставленной задачи.

4. Составить тест-таблицу для проверки алгоритма и программы.

5. Подготовиться к ответам на контрольные вопросы.

### 3.4. Лабораторное задание

1. Создать текст программы, разработанной при подготовке домашнего задания, и сохранить его в файле.

2. Выполнить отладку и тестирование программы.

3. Рассчитать зависимость напряжения на выходе от времени путем многократного запуска программы на выполнение, изменяя время $t$ в интервале $[0\ldots T]$ ($T=1/f$ – период функции).

4. Построить график рассчитанной функции, проанализировать результаты и сделать выводы.

### 3.5. Содержание отчета

1. Название, цель и задача работы.

2. Формальная постановка задачи, расчетные выражения.

3. Алгоритм и программа для решения поставленной задачи.

4. Тест-таблица.

5. Текст программы и график зависимости от времени напряжения на выходе цепи.

**Контрольные вопросы**

1. Напишите программу, определяющую четность введенного с клавиатуры целого числа.

2. Напишите программу, определяющую, делится ли заданное целое число *n* на 2 и 3 без остатка.

3. Напишите программу, определяющую, дает ли заданное целое число *m* при делении на 5 в остатке 3.

4. Напишите программу, определяющую, принадлежит ли заданное число *x* одному из числовых интервалов [–2,–1] и [1,2].

5. Перераспределите значения переменных *x* и *y* так, чтобы значение *x* оказалось большим из них, а *y* – меньшим.

6. Напишите программу решения следующей задачи: значения переменных *a, b, c* поменять местами так, чтобы оказалось $a \geq b \geq c$. Исходные значения переменных *a, b, c* ввести с клавиатуры.

7. Известно, что из четырех чисел $a_1$, $a_2$, $a_3$, $a_4$ одно отлично от трех других, равных между собой. Присвойте номер этого числа переменной *n*.

8. Дано число *x*. Выведите на экран в порядке убывания числа *cos(x)*, |1+*x*| и $\sqrt{1+x^2}$ .

9. Дано число *x*. Выведите на экран в порядке возрастания числа *ch(x)*, 1+|*x*|, $1+x^2$. Функция *ch(x)* (гиперболический косинус) определяется по формуле $ch\ x = \dfrac{e^x + e^{-x}}{2}$.

10. Напишите программу, вычисляющую значение величины z, которая равна *sin*(*x*), если некоторая логическая переменная *a = true,* и *x*, если *a = false.*

11. Составьте программу решения квадратного уравнения $ax^2 + bx + c = 0$, учитывающую все возможные частные случаи.

12. Даны числа *a*, *b* и *c* ($c \neq 0$). Найдите вещественные корни уравнения $ax^4 + bx^2 + c = 0$. Если корней нет, сообщите об этом.

13. Напишите программу, вычисляющую значение величины

$$y = \begin{cases} a + bx + cx^2, & \text{если } k = 1, \\ d + cx + fx^2, & \text{если } k = 2, \\ g + hx + px^2, & \text{в остальных случаях.} \end{cases}$$

14. Переменной *k* присвойте номер четверти плоскости, в которой находится точка с заданными координатами *x* и *y* ($xy \neq 0$).

15. Напишите логическое выражение для проверки, принадлежит ли точка с координатами (*x, y*) области *S*, где область *S* пред-

ставляет собой круг радиусом $R$, из которого вырезан квадрат со стороной $a$. Центры круга и квадрата лежат в начале координат.

16. Дано: $S_1$ – площадь круга, $S_2$ – площадь квадрата. Определите, поместится ли квадрат в круг.

17. В восточном календаре принят 60-летний цикл, состоящий из 12-летних подциклов, обозначаемых названиями цвета: зеленый, красный, желтый, белый и черный. В каждом подцикле годы носят названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, курицы, собаки и свиньи. Разработайте программу, которая по номеру года, введенного с клавиатуры, определяет его название, если известно, что 1984 год – начало цикла: «год зеленой крысы».

18. Определите, равна ли сумма двух первых цифр заданного четырехзначного целого числа сумме двух его последних цифр.

19. Определите, равен ли квадрат заданного трехзначного целого числа кубу суммы цифр этого числа.

20. Определите, есть ли среди первых трех цифр из дробной части заданного положительного вещественного числа цифра 0.

21. Определите, есть ли среди цифр заданного трехзначного целого числа одинаковые.

22. Определите среднее по величине число из трех введенных с клавиатуры целых чисел. Если среди чисел есть равные, выведите сообщение «Ошибка».

23. С клавиатуры последовательно вводятся числа, состоящие из одной цифры. Составьте программу, которая подсчитывает, сколько раз появились числа $N$ и $M$, предварительно введенные с клавиатуры. Работа программы прекращается при вводе числа 0.

24. С клавиатуры последовательно вводятся числа. Разработайте программу, вычисляющую номера первого и последнего нулевых чисел. Работа программы должна прекратиться при вводе любого отрицательного числа. Если нулевых чисел не встретилось, выдайте сообщение об этом.

25. С клавиатуры последовательно вводятся числа. Напишите программу, которая после каждого введенного числа выдает среднее арифметическое всех введенных до этого чисел. Работа программы должна прекратиться, если более трех раз подряд вводится число 0.

# 4. ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ. ТАБУЛИРОВАНИЕ ФУНКЦИЙ

Для организации циклического алгоритма используются операторы повтора. *Цикл* – это последовательность операторов, которые могут выполняться более одного раза. Если количество повторений известно заранее, используется оператор *for,* если количество повторов неизвестно, применяются операторы *repeat* или *while.*

*Оператор повтора for* состоит из заголовка и тела цикла. Он может быть представлен в двух форматах:

1) *for <параметр цикла>:= <S1> to <S2> do <оператор>;*

2) *for <параметр цикла>:= <S1> downto <S2> do<оператор>;*

Здесь *<S1>* и *<S2>* – выражения, определяющие соответственно начальное и конечное значения параметра цикла; *for...do* – заголовок цикла; *< оператор >* – тело цикла, представляющее собой простой или составной оператор. Оператор *for* обеспечивает выполнение тела цикла до тех пор, пока не будут перебраны все значения параметров цикла от начального до конечного.

**Пример.**

Оператор

*for i:=1 to 20 do writeln(i, Sqrt(i));*

выводит на экран монитра 20 значений квадратного корня из величины *i*, которая последовательно принимает значения 1, 2, …, 20, причем каждый результат будет находиться в отдельной строке.

Параметр цикла, его начальное и конечное значения должны принадлежать к одному и тому же типу данных, при этом допустим любой скалярный тип, кроме вещественного. Если используются типы *integer, byte, word* или интервальный, то значение параметра цикла последовательно увеличивается (при использовании формата *for...to)* или уменьшается (при использовании формата *for...downto)* на единицу при каждом повторе.

**Пример.**

Оператор

*for i:=10 to 14 do write(i:3);*

выводит на экран монитора в одной строке последовательность чисел 10 11 12 13 14.

В операторе *for* не допускается изменение параметра цикла на

величину, отличную от единицы. Однако этот недостаток устраняется использованием операторов *repeat* и *while,* где можно задать любой шаг.

В операторе *for* после служебного слова *do* может находиться составной оператор, являющийся телом цикла, который обязательно должен быть охвачен операторными скобками *begin…end*. В теле цикла запрещены операторы, меняющие значение параметра цикла оператора повтора *for*.

**Пример.**

```
for i:=1 to 10 do
begin
  rez := i / pi;
  i := i+1; {ошибка – изменяется параметр цикла i}
end;
```

После нормального завершения оператора *for* значение параметра цикла равно конечному значению, и управление передается оператору, стоящему после тела цикла. Если оператор *for* не выполняется, значение параметра цикла не определено.

В теле цикла оператора *for* могут находиться и другие операторы *for*. Это позволяет строить циклы, содержащиеся внутри других циклов. Такие внутренние циклы называются *вложенными.*

**Пример.**

```
for i:=1 to 10 do
begin
  for j:=1 to 15 do
    A[i,j]:=0.0; {обнуление элементов матрицы}
end;
```

Оператор повтора *repeat* состоит из заголовка *repeat,* тела цикла и условия окончания *until.* Формат оператора имеет вид

```
repeat
  <оператор>;
  <оператор>;
  ……….
  <оператор>
until <условие>;
```

Здесь *<условие>* – выражение булевского типа. Операторы, заключенные между словами *repeat* и *until*, являются телом цикла. Вначале выполняется тело цикла, затем проверяется условие выхо-

да из цикла. Если результат булевского выражения *FALSE*, то тело цикла активизируется еще раз, если результат *TRUE*, то происходит выход из цикла (к оператору, стоящему после строки *until <условие>*).

Оператор *repeat* имеет три характерные особенности:

1) выполняется по крайней мере один раз;

2) тело цикла выполняется, пока условие равно *FALSE*;

3) в теле может находиться произвольное число операторов без операторных скобок *begin…end*.

Так как оператор *repeat* выполняется по крайней мере один раз, то важно четко проверить правильность вхождения в цикл и выхода из него. При неправильной организации выхода может возникнуть ошибка переполнения, ибо наращивание какой-либо величины в теле цикла может продолжаться до бесконечности.

**Пример.**

```
Program Repeat_Demo;
{вычисление суммы четных чисел в интервале от 0 до 10 включительно}
Var
  i ,sum: integer;
Begin
  i:=0; sum:=0;
  repeat
    sum:=sum+i;
    i:=i+2
  until > 10;
  writeln('Сумма четных чисел равна', sum:3)
End.
```

Оператор повтора *while* аналогичен оператору *repeat,* но проверка условия выполнения тела цикла производится в самом начале оператора. Формат оператора имеет вид

*while <условие> do <оператор>;*

Здесь *<условие>* – булевское выражение, а *<оператор>*– тело цикла, представляющее собой простой или составной оператор (составной оператор обязательно должен быть охвачен операторными скобками *begin…end*). Перед каждым выполнением тела цикла вычисляется значение выражения условия. Если результат равен *TRUE*, то тело цикла выполняется, и снова вычисляется значение

выражения условия. Если результат равен *FALSE*, то происходит выход из цикла и переход к первому оператору после оператора *while* (стоящий после тела цикла). Если перед первым выполнением цикла значение выражения равно *FALSE,* тело цикла вообще не выполняется и происходит переход на следующий оператор (стоящий после тела цикла).

**Пример.**

```
i:=30;
while(i<0) do
  i:=i+1;
write(i);
```

В данном случае наращивание *i* на единицу не происходит, так как условие с самого начала имеет значение *FALSE*, поэтому оператором вывода будет выведено значение 30.

Оператор *while,* как и другие операторы организации циклов, может быть вложенным.

**Пример.**

```
Program While_Demo;
{Вычисление суммы нечетных чисел в интервале от 0 до 10}
Var
  i, sum: integer;
Begin
  i:=1; sum:=0;
  while i<11 do
  begin
    sum:=sum+i;
    i:=i+2
  end;
  Write('Сумма нечетных чисел равна', sum:3)
End.
```

## ПРАКТИЧЕСКАЯ РАБОТА № 4

### 4.1. Цель работы

Изучение приемов организации однократных и вложенных циклов по простой и индексной переменной.

### 4.2.Задача работы

Рассчитать и построить зависимость от времени *t* напряжения

$U_2(t)$ на выходе электрической цепи. Варианты схем приведены в табл. 3.1 (см. практическую работу №3). На вход цепи воздействует источник гармонического напряжения $e(t) = E\,Cos(2\pi f t + \varphi)$, где $f$ – частота, $\varphi$ – начальная фаза. Значения сопротивлений, амплитуду $E$, частоту $f$ и начальную фазу $\varphi$ гармонического напряжения источника выбрать самостоятельно.

### 4.3. Домашнее задание

1. Изучить способы организации однократных и вложенных циклов с помощью операторов *for, repeat* и *while*, а также формат операторов, особенности применения каждого оператора.

2. Выполнить формализацию поставленной задачи. Обратить внимание, что рассчитываемая цепь и расчетные соотношения полностью соответствуют заданию практической работы №3. Однако постановка задачи другая: требуется получить таблицу значений функции $U_2(t)$ с использованием циклического алгоритма. Для отличной оценки необходимо разработать три варианта программы: с оператором *for,* с оператором *repeat* и с оператором *while.*

При выполнении формализации задачи обратить внимание на шаг изменения времени и число рассчитываемых точек.

3. Разработать алгоритм и программу для решения задачи.

4. Составить тест-таблицу для проверки программы.

5. Подготовиться к ответам на контрольные вопросы.

### 4.4. Лабораторное задание

1. Создать и сохранить в файле текст программы, разработанной при выполнении домашнего задания.

2. Выполнить отладку и тестирование программы.

3. Распечатать полученную в процессе выполнения программы таблицу значений функций.

4. Построить график рассчитанной функции, проанализировать результаты и сделать выводы.

### 4.5. Содержание отчета

1. Название, цель и задача работы.
2. Формальная постановка задачи, расчетные выражения.
3. Алгоритм и программа для решения поставленной задачи.
4. Тест-таблица.
5. Текст программы, график рассчитанной зависимости.

**Контрольные вопросы**

1. Используя оператор *for,* напишите программу вычисления функции $y=sin(x)$ в интервале значений $x=0...\pi$ с шагом $\pi/10$. Организуйте вывод на экран таблицы значений аргумента и функции.

2. Используя оператор *repeat,* напишите программу вычисления функции $y=cos(x)$ в интервале значений $x=0...\pi$ с шагом $\pi/10$. Организуйте вывод на экран таблицы значений аргумента и функции.

3. Используя оператор *while,* напишите программу вычисления функции $y=tg(x)$ в интервале значений $x=0...\pi$ с шагом $\pi/10$. Организуйте вывод на экран таблицы значений аргумента и функции

4. Табулируйте функцию $Y = |sin^2(x)+K^2 cos^2(x)|$ при изменении $x$ от 0,1 до 1,0 с шагом 0,1. Результаты вычислений оформите в виде таблицы. Значение $K$ введите с клавиатуры.

5. Дано натуральное число $n$ и действительное число $A$. Вычислите, не используя операцию возведения в степень, $Z=A^n$.

6. Вычислите 10! каждым из трех вариантов оператора цикла.

7. Дано натуральное число $n$. Найдите сумму цифр, использованных для его записи.

8. Вычислите число $e$ с помощью ряда $e=1+1/1!+1/2!+...+1/n!$.

9. Разработайте программу для вычисления функции $e^x$. Разложение функции $e^x$ в бесконечный ряд имеет вид $e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$. Вычислите $e^x$ путем суммирования ряда для $x=0; 0,1; 1,0; 5,0; 10,0$ и сравните с точными значениями, полученными с помощью стандартной функции $exp(x)$.

10. Вычислите значение переменной
$P=1/(x+1)+1/(x+2)+\ldots+1/(x+k)$ при $k=1...30$.

11. Имеется массив *MAS* длиной 100 элементов типа *integer*. Разработайте программу, которая подсчитывает и выводит на экран количество отрицательных, положительных и нулевых значений.

12. Вычислите значение $y=cos(x)+cos(x^2)+\ldots+cos(x^{30})$.

13. Числа Фибоначчи $f_n$ определяются формулами: $f_0=f_1=1$; $f_n=f_{n-1}+f_{n-2}$ при $n=2, 3, \ldots$ . Определите величину $f_{40}$.

14. Вычислите величину $y=sin(1,0)+sin(1,1)+sin(1,2)+\ldots+sin(2,0)$.

15. Напишите программу печати таблицы перевода расстояний из дюймов в сантиметры для значений длин от 1 до 20 дюймов (1 дюйм = 2,54 см).

16. Присвойте значения элементам массива $N[i]$ по правилу $N[0]=100$, $N[1]=99,\ldots$, $N[100]=0$, после чего выведите на экран только четные элементы массива.

17. При падении тела с большой высоты зависимость скорости падения от времени описывается выражением $v(t) = v_0\left(1 - e^{-\frac{gt}{v_0}}\right)$, где $v_0$ – скорость падения в установившемся режиме. Разработайте программу расчета зависимости скорости от времени $t$ при $v_0$=80 м/с для $t$ в диапазоне от 0 до 20 с с шагом 1 с.

18. Математическая точка участвует одновременно в двух взаимно перпендикулярных гармонических колебаниях, уравнения которых $x = A_1 \cos wt$, $y = A_2 \cos\frac{w}{2}t$. Разработайте программу для расчета траектории движения точки в интервале времени $t$ от 0 до 10 с с шагом 1 с. Требуемые величины выберите самостоятельно.

19. Распечатайте таблицу умножения $N \times N$, выделяя для каждого числа две позиции. Число $N$ введите с клавиатуры.

20. Распечатайте таблицу умножения нечетных чисел $N \times N$, выделяя для каждого числа три позиции. Число $N$ введите с клавиатуры. Если $N<0$ или четное, предусмотрите вывод предупреждающего сообщения.

21. Имеется массив $A$ длиной 10 элементов. Разработайте программу, вычисляющую номер первого нулевого элемента. Если в массиве нет нулевых элементов, выдайте сообщение об этом.

22. Имеется массив $A$ длиной 15 элементов типа *integer*. Перепишите в массив $B$ из массива $A$ только четные элементы.

23. Имеются два массива $A$ и $B$ длиной $N$ элементов. Переменной $R$ присвойте значение *TRUE*, если более половины элементов массива $A$ равны элементам массива B, и значение *FALSE* в противном случае. Значение $N$ введите с клавиатуры.

24. Имеется литерный массив длиной $N$ элементов. Подсчитайте, сколько раз встретился символ «+» и сколько раз символ «*». Значение $N$ и элементы массива введите с клавиатуры.

25. Разработайте программу, которая в исходной строке *SS* убирает все начальные и концевые пробелы, а также заменяет группы символов из двух и более пробелов одним пробелом.

# 5. ПОДПРОГРАММЫ

В практике программирования довольно часто встречаются ситуации, когда одну и ту же группу операторов, реализующих определенную цель, требуется повторить без изменений в нескольких других местах программы. Чтобы избавиться от столь нерационального занятия, была предложена концепция подпрограмм.

*Подпрограммой* называется именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения по имени любое количество раз из различных мест программы. По назначению и правилам построения аналогичны программам. Отличие заключается, во-первых, в том, что программа вызывается с помощью команд операционной системы, а подпрограмма вызывается из так называемой вызывающей программы и, во-вторых, что программа ведет обмен информацией с внешней средой, а подпрограмма – с вызывающей программой.

В языке Delphi для организации подпрограмм используются процедуры и функции. Как и другие объекты языка Delphi (переменные, константы, типы), подпрограмма должна быть описана до того, как будет использована.

Описание и реализация встроенных (стандартных) процедур и функций выполняется разработчиком транслятора, поэтому их можно вызывать по имени без предшествующего описания.

Delphi предоставляет программисту достаточно широкий набор встроенных процедур и функций, которые реализуют наиболее часто используемые действия. В соответствии с областями применения различают 9 основных групп встроенных процедур и функций: арифметические, скалярные, преобразования типов, управления строками на экране, специальные, обработки строк, обработки файлов, управления памятью для динамических переменных, базовой и расширенной графики.

Процедуры и функции пользователя организуются самим программистом. Структура описания подпрограмм следующая:

заголовок;
[раздел описания констант];
[раздел описания типов];
[раздел описания переменных];
[раздел описания процедур и функций];
раздел операторов.

Таким образом, структура подпрограммы аналогична структуре основной программы. Отличие заключается лишь в правилах написания обязательного для подпрограмм заголовка и некоторых особенностях построения раздела операторов в случае описания функций.

*Процедура пользователя* представляет собой именованную группу операторов, реализующую общую часть задачи и вызываемую по имени из любого места раздела операторов программы.

Вызвать процедуру можно в разделе операторов, указав ее имя, и после него заключенный в круглые скобки список фактических параметров. Такая запись называется оператором вызова процедуры. Если процедура не имеет параметров, список вместе с круглыми скобками может быть опущен. После завершения работы процедуры выполняется следующий оператор.

*Функция пользователя* имеет существенные особенности. В отличие от процедуры функция после завершения своей работы передает в вызвавшую программу значение скалярного типа. Главное удобство использования функции заключается в том, что ее можно использовать в выражениях. Запись в выражении имени функции и списка фактических аргументов называется вызовом функции.

**Пример.**

*Sum(A,X);* {вызов процедуры *Sum* с аргументами *A* и *X*}

Программа, которая вызывает подпрограмму, называется вызывающей программой, а подпрограмма, которая вызвана, называется вызываемой. В свою очередь, подпрограмма может вызвать другую подпрограмму и т.д. Порядок выполнения операторов при использовании подпрограмм показан на рис. 5.1.

Подпрограмма может вызывать и саму себя (непосредственно или через цепочку вызовов других подпрограмм). Такие подпрограммы называются рекурсивными. Для их написаний необходимо глубокое понимание их принципа действия, но при этом могут быть получены очень изящные подпрограммы.

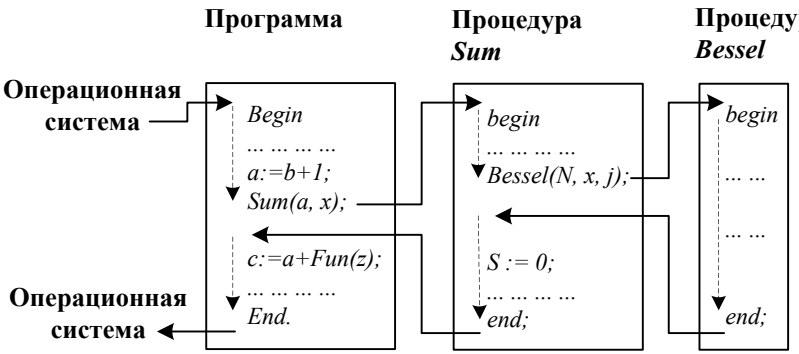



Рис. 5.1. Вложенные вызовы процедур

Заголовок процедуры имеет формат

*Procedure <имя> [(список формальных параметров)];* .

Заголовок функции имеет формат

*Function <имя> [(список формальных параметров)] : <тип>;*

Здесь *<имя>* – это идентификатор, который должен отличаться от других идентификаторов, используемых в программе. Список формальных параметров содержит описания формальных параметров в том же виде, как и в разделах описания констант и переменных. Слово *Const* в списке формальных параметров допускается опускать. Подпрограмма может совсем не иметь параметров, в этом случае список вместе с круглыми скобками может отсутствовать. Тип может быть любым, в том числе структурированным.

**Примеры.**

*Procedure Sort(A: integer; B: real C: booelan);*

{процедура с именем *Sort* имеет 3 параметра: константы целого, действительного и булевского типов};

*Procedure Summa(A,B: real; Var D :real);*

{процедура с именем *Summa* имеет 3 параметра: две константы и одну переменную};

*Procedure Tree;*

{процедура без параметров}

Те формальные параметры, которые являются для подпрограммы исходными данными, целесообразно объявлять как константы. Те параметры, которые являются выходными данными, обязатель-

но должны быть переменными. Может быть и третий случай, когда переменная перед вызовом подпрограмм определяет исходные данные, а после вызова получает выходные данные (исходные данные при этом портятся!).

Тип параметров в секции формальных параметров должен быть указан в виде определенного ранее идентификатора типа. Так, запись вида

*Procedure Spisok (IER: array[1..25] of real);*

является ошибочной. Тип переменной следует определить ранее в разделе описания типов:

*Type Gruppa = array[1..25] of real;*
*procedure Spisok (IER: Gruppa);*

Функции в качестве результата работы выдают значение, тип которого указывается через двоеточие после списка формальных параметров. Тип может быть только скалярным (например, *integer*, *real*, *boolean*, *char*), типом *string* или *pointer*.

**Пример.**

*Function Bessel(N: Integer; X:real): real;*

{функция с именем *Bessel* с двумя аргументами в виде констант, первый из них целого типа, второй – действительного. Тип результата – *real*}

*Function Exist: boolean;*

{функция без параметров}.

Рассмотрим особенности построения раздела операторов при описании подпрограммы. Единственной особенностью раздела операторов процедуры является то, что после последнего служебного слова *End* ставится не точка, обозначающая конец программы, а точка с запятой, отделяющая описание процедуры от других описаний.

Для функции имеется и второе отличие. Так как функция возвращает значение, то в блоке операторов этой функции должен быть хотя бы один оператор, который присваивает идентификатору функции требуемое значение. Если таких операторов несколько, то значением функции будет результат последнего присваивания идентификатору функции.

**Пример.**

*Function Norma(x, y : real): real;*
*Var*

```
  z: real;
Begin
  z: =Sqr(x) +Sqr(y); Norma:=Sqrt(z)
end; {of Norma}
```

При использовании подпрограмм важен вопрос соответствия *формальных* и *фактических* параметров. Во-первых, число фактических параметров при вызове подпрограммы должно совпадать с числом формальных параметров в описании. Нельзя опускать ни один из фактических параметров, даже если его значение при этом вызове не играет роли. Во-вторых, фактические и формальные параметры должны совпадать по типу.

Различен механизм передачи *параметров-констант* и *параметров-переменных*. В первом случае в подпрограмму передается значение этого аргумента, поэтому при вызове подпрограммы в качестве параметра-константы может быть указан идентификатор, выражение соответствующего вида, константа или указатель функции.

При использовании параметров-переменных в подпрограмму передается адрес памяти, на который указывает идентификатор, применяемый в качестве фактического параметра (т.е. ссылка на адрес памяти). Этот способ передачи параметров называется передачей параметров по ссылке. Фактически во время исполнения подпрограммы используется та область памяти, которая отведена в вызывающей программе для идентификатора фактического параметра. Таким образом, в качестве фактического параметра в случае параметров-переменных могут быть использованы только идентификаторы (возможно, с индексами).

**Примеры.**

```
Var
  a, b, x : real;
  c       : char;
Procedure fun1(x, y: real);
begin
……….
end;
Procedure fun2(x: real; Var y: real);
begin
……….
```

*end;*

*fun1(A, B);* {параметры-значения переменных}

*fun1(Sin(x)+2.0, 8);* {первый параметр – выражение, второй – константа типа *integer*, которая преобразуется транслятором к типу *real*}

*fun1(x);* {ошибка – пропущен один аргумент}

*fun2(x, y);* {переменной *y* может быть присвоено значение}

*fun2(c, x);* {ошибка – несоответствие типов}

*fun2(x, y+2.0);* {ошибка – второй фактический параметр не может быть выражением}.

*Область действия идентификаторов*. Как уже указывалось ранее, программа на языке Delphi имеет модульную структуру и может состоять из ряда вложенных друг в друга блоков. Основная программа – это самый крупный блок, который не входит ни в какой другой. Объекты, описанные в этом блоке, являются глобальными и могут использоваться во всех вложенных блоках. Вложенные блоки – это функции и процедуры. Описанные в них объекты локальны и недоступны во внешних блоках.

Встречаются случаи, когда один и тот же идентификатор описан и во внешнем блоке, и во внутреннем. В этом случае при входе во внутренний блок действует описание внутреннего блока, а объект внешнего блока становится недоступным (компьютер «забывает» его). После завершения исполнения внутреннего блока вновь становится доступным объект из внешнего блока.

Одной из самых распространенных ошибок в работе с подпрограммами является случайное использование глобальных переменных в процедурах и функциях. Например, если переменная *i* выбрана для управления циклом в основной программе и случайно была использована без внутреннего описания в подпрограмме, то работа программы будет в большинстве случаев полностью дезорганизована. Поэтому необходимо внимательно следить за тем, чтобы все переменные в подпрограммах были описаны, и по возможности избегать обозначения разных объектов одинаковыми идентификаторами.

Для правильного использования и определения области действия идентификаторов в программе необходимо соблюдать следующие правила:

1. Каждый идентификатор должен быть описан перед тем, как он будет использован.

2. Областью действия идентификатора является блок, в котором он описан (в том числе и во вложенных блоках).

3. Все идентификаторы в блоке должны быть уникальными, т.е. не повторяться.

4. Один и тот же идентификатор может определять в разных блоках разные объекты. Например, в одной из подпрограмм *I* обозначает число точек, а в другой – электрический ток.

5. Если идентификатор программы пользователя совпадает с именем стандартной процедуры или функции, то последние недоступны в пределах области действия подпрограммы, объявленной пользователем, т.е. стандартная функция игнорируется, а выполняется подпрограмма пользователя.

Особую внимательность необходимо проявлять при использовании глобальных переменных в связи с тем, что легко перепутать область действия идентификаторов, возможна непредсказуемая порча переменных и так далее. Рекомендуется, чтобы подпрограммы использовали только внутренние (локальные) объекты, и обмен информацией осуществлялся только через список параметров.

Однако в некоторых случаях использование глобальных переменных может принести существенный выигрыш по длине программы и ее эффективности. В этих случаях следует проявлять особую осторожность в используемых обозначениях объектов.

## ПРАКТИЧЕСКАЯ РАБОТА № 5

### 5.1. Цель работы

Приобретение навыков составления подпрограмм пользователя и обращения к подпрограммам.

### 5.2. Задача работы

Рассчитать две зависимости напряжения на выходе электрической цепи (варианты схем приведены в табл. 3.1, см. практическую работу №3):

1. Зависимость напряжения на выходе электрической цепи $U_2(t)$ от времени $t$ при условии, что на ее входе воздействует гармоническое напряжение $e(t)=E\times Cos(2\pi f t + \varphi)$, где $f$ – частота (Гц), а $\varphi$ – начальная фаза. Значения сопротивлений, амплитуду $E$, частоту $f$ и начальную $\varphi$ гармонического напряжения источника выбрать самостоятельно.

2. Зависимость напряжения $U_2$ от сопротивления нагрузки при постоянном напряжении на входе, равном двум значениям: +1 В и –1 В. В качестве сопротивления нагрузки считать: в вариантах 1, 3, 4, 10, 11, 12 – сопротивление *R4,* в вариантах 2, 7, 8, 9 – сопротивление *R3,* в вариантах 5, 6 – сопротивление *R2.*

### 5.3. Домашнее задание

1. Изучить правила описания подпрограмм типов *procedure* и *function*, особенности применения каждого типа подпрограмм.

2. Выполнить формализацию задачи. Обратить внимание, что рассчитываемая цепь и расчетные соотношения полностью соответствуют заданиям практических работ №3 и 4.

Постановка задачи аналогична практической работе №4, но дополняется требованием расчета нагрузочной характеристики цепи, которая представляет собой зависимость напряжения на выходе цепи $U_2$ от сопротивления нагрузки при фиксированном напряжении на входе.

Как при расчете зависимости от времени, так и при расчете зависимости от сопротивления нагрузки, необходимо рассчитывать напряжение на выходе при заданном напряжении на входе и заданных номиналах сопротивлений. Эту часть программы удобно оформить в виде подпрограммы-функции. Для проведения вычислений организуются два цикла. В первом из них изменяется время (пределы изменения и шаг выбрать в соответствии с величиной периода входного напряжения). Во втором – изменяется значение сопротивления нагрузки от 0 до ∞ (как задать ∞ в расчетах с помощью компьютера?). В обоих циклах для расчета $U_2$ вызывается одна и та же разработанная подпрограмма, но с различными параметрами.

3. Разработать алгоритм и программу для решения задачи.

4. Составить тест-таблицу для проверки программы.

5. Подготовиться к ответам на контрольные вопросы.

### 5.4. Лабораторное задание

1. Создать и сохранить в файле текст программы, разработанной при выполнении домашнего задания.

2. Выполнить отладку и тестирование программы.

3. Построить графики рассчитанных зависимостей, проанализировать результаты и сделать выводы.

### 5.5. Содержание отчета

1. Название, цель и задача работы.
2. Формальная постановка задачи, расчетные выражения.
3. Алгоритм и программа для решения поставленной задачи.
4. Тест-таблица.
5. Текст программы, анализ ошибок, графики рассчитанных зависимостей.

### Контрольные вопросы

1. Оформите в виде подпрограммы-функции определение наименьшего общего делителя двух натуральных чисел.
2. Напишите подпрограмму для вычисления функции $R(\tau)=e^{-\beta\tau}$.
3. Напишите подпрограмму-процедуру для вычисления функции $U(t)=U_0\cos(\omega t+\varphi)$. Формальные параметры процедуры: $U_0$, $\omega$, $t$.
4. Оформите в виде подпрограммы-функции *sinus*($x$) вычисление значения синуса путем суммирования ряда $x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots$. Выберите критерий окончания цикла. Сравните результаты вычислений с помощью разработанной и стандартной функции *sin*($x$).
5. Оформите в виде подпрограммы-функции *cosinus*($x$) вычисление значения косинуса путем суммирования ряда $1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots$. Выберите критерий окончания цикла. Сравните результаты вычислений с помощью разработанной и стандартной функции *cos*($x$).
6. Оформите в виде подпрограммы-функции *arctg(x)* вычисление функции арктангенса путем суммирования ряда $x-\frac{x^3}{3}+\frac{x^5}{5}-\frac{x^7}{7}+\frac{x^9}{9}-\ldots$. Выберите критерий окончания цикла. Сравните результаты вычислений с помощью разработанной и стандартной функции *atan*($x$).
7. Оформите в виде подпрограммы-функции *logarifm*($x$) вычисление значения натурального логарифма путем суммирования ряда $(x-1)-\frac{(x-1)^2}{2}+\frac{(x-1)^3}{3}-\frac{(x-1)^4}{4}\ldots$. Выберите критерий окончания цикла. Сравните результаты вычислений с помощью разра-

ботанной и стандартной функции *ln*(*x*).

8. Составьте подпрограмму-процедуру для вычисления полиномов Эрмита по рекуррентной формуле $H_{n+1} = 2(xH_n + nH_{n-1})$, где $H_0 = 1, H_1 = x$. Оператор обращения к этой процедуре должен иметь вид *HERMIT*(*n, x ,H*), где *n* – порядок полинома, *x* – аргумент, *H* – переменная, которой присваивается значение полинома.

9. Составьте подпрограмму-функцию, переводящую значение угла из градусов в радианы и наоборот. Обращение к функции должно иметь вид *Grad_To_Rad*(*Ugol*, *D*). При входном параметре *D*, равном «>», должен производиться перевод из градусов в радианы, а при *D*, равном «<» – обратный перевод.

10. Даны два действительных массива *A* и *B*, содержащие по 10 элементов каждый. Оформите в виде процедуры нахождение суммы максимальных элементов этих массивов. Входные параметры – *A* и *B*, выходной параметр – *C* (сумма максимальных элементов).

11. Даны три одномерных целочисленных массива *A*, *B* и *C* по 5 элементов каждый. Оформите в виде функции нахождение суммы минимальных элементов этих массивов.

12. Заданы две квадратные матрицы размером 4×4. Исключите из диагоналей нулевые элементы, заменив их единицей. Оформите в виде функции нахождение произведения элементов диагоналей.

13. Нормируйте квадратную матрицу размером 6×6, разделив все элементы матрицы на значение максимального элемента матрицы. Оформите в виде подпрограммы-процедуры.

14. Даны три вещественные квадратные матрицы *A*, *B* и *C* размером 4×4. Оформите в виде подпрограммы-процедуры с входными параметрами *A*, *B* и *C* вывод на экран элементов той из матриц, у которой наибольшая сумма элементов главной диагонали.

15. Оформите в виде подпрограммы-функции вычисление значения: $t = \begin{cases} \dfrac{\min(b_i)}{\max(a_i)} + \dfrac{\max(c_i)}{\min(b_i + c_i)}, & \textit{если } \min(a_i) < \max(b_i), \\ \max(b_i + c_i) + \min(c_i) & \textit{иначе}, \end{cases}$

где *a, b, c* – массивы действительных чисел из 10 элементов.

16. Напишите процедуру, которая присваивает переменной *t* значение *true*, если уравнение $ax^2+bx+c=0$ имеет действительные корни, и значение *false* в противном случае.

17. Заданы следующие описания:

*type Strana = (‘Австрия’, ‘Алжир’, ‘Аргентина’, ‘Египет’, ‘Италия’, ‘Россия’, ‘Швеция’, ‘США’);*
*continent = (‘Америка’, ‘Африка’, ‘Европа’);*

Напишите функцию *cont*(*S*), определяющую континент, на котором находится страна *S*.

18. Заданы следующие описания:

*Const n = 1000;*
*type vector = array[1..n] of real;*
*var a, b ,c, d : vector;*

Напишите процедуру *Sum*(*x, y, z*), которая присваивает вектору *z* сумму векторов *x* и *y*. Используйте ее для вычисления *d=a+b+c*.

19. Разработайте подпрограмму-функцию, которая принимает значение *TRUE*, если во входном слове нет повторяющихся букв, и значение *FALSE* в противном случае.

20. Задан массив *Z* типа *char*, состоящий из 500 элементов. Разработайте подпрограмму-функцию, вычисляющую, сколько раз в массиве встречается символ «*?*».

21. Составьте подпрограмму, выдающую название дня недели по его порядковому номеру, например, при *n*=3 должен вырабатываться результат «*среда*». Обращение к функции имеет вид *DAY*(*n*).

22. Составьте подпрограмму для вычисления номера строки и номера столбца матрицы *A* размером $N \times M$, на пересечении которых находится наибольший по абсолютной величине элемент этой матрицы.

23. Разработайте процедуру *maxmin*(*x, y*), присваивающую параметру *x* большее из вещественных чисел *x* и *y*, а параметру *y* – меньшее, и используйте ее для перераспределения значений *a*, *b* и *c* так, чтобы стало $a > b > c$.

24. Разработайте подпрограмму-функцию, вычисляющую сумму элементов одномерного массива. Обращение к этой подпрограмме должно иметь вид *sum*(*A*), где *A* – идентификатор массива длиной 100 элементов типа *real*.

25. Задано целое число *P* и массив *A* размерностью 50 элементов типа *integer*. Разработайте подпрограмму-функцию, вычисляющую отношение $Z = \frac{x}{y}$, где *x* – сумма элементов массива *A*, удовлетворяющих условию $A_i < P$, *y* – сумма элементов массива A, удовлетворяющих условию $A_i > P$.

# 6. ОБРАБОТКА ФАЙЛОВ

Компьютерная программа предназначена для обработки информации, которая поступает в программу из «внешней среды» и выдается программой во «внешнюю среду». Такой «внешней средой» для программы являются так называемые периферийные устройства, к которым относятся дисплей, клавиатура, накопители на магнитных дисках, принтер, устройство связи с объектом управления и т.д. Связь программы с периферийными устройствами осуществляется через файлы. Существуют файлы с прямым и последовательным доступом. Стандартом языка Delphi предусматривается обработка только файлов последовательного доступа.

Файл с последовательным доступом представляет собой совокупность расположенных последовательно записей (строк), число которых заранее неизвестно. Каждая запись состоит из элементов записи и заканчивается обязательным признаком «конца записи». Чаще всего в Delphi элементами записи являются литеры (символы), поэтому записи называются также строками, а признак окончания записи – «конец строки»

*End of line* (*Eoln*).

Обычно в качестве конца строки применяется символ «возврат каретки», соответствующий нажатию клавиши <ENTER>.

Весь файл завершается специальной служебной записью «конец файла»

*End of file* (*Eof*).

При вводе данных с клавиатуры такой записью является одновременное нажатие клавиш <Ctrl / Z>.

Существуют *файлы для чтения*, из которых можно только читать, и *файлы для записи*, в которые можно писать информацию. Очевидно, в файле для чтения должна иметься требуемая информация, а в файле для записи должно быть достаточно места для помещения требуемой информации. При обработке файлов используется понятие текущий указатель файла, который показывает строку и номер позиции в строке. При записи на это место будет записана литера, а при вводе – введена указанная литера (может быть, *Eoln* или *Eof*). При операции чтения или записи текущий указатель перемещается на один элемент записи вправо, а при необходимости и в начало следующей строки.

Методика работы с файлами в языке Delphi заключается в следующем. Вначале для обработки каждого файла описывается файловая переменная. Затем эта переменная связывается с конкретным периферийным устройством, или, иначе, файлу назначается конкретное периферийное устройство. После этого файл открывается. Открыть файл можно для записи или чтения. В открытый файл помещаются или считываются записи. После окончания работы файл закрывается.

Рассмотрим описание файлов. Для описания файлов необходимо описать их тип. Это производится следующим образом:

*type <тип файла> = file of <тип компонентов>;* .

Здесь, как и обычно, *<тип файла>* – идентификатор, *<тип компонентов>* – это любой простой или структурированный тип (например, тип *record*).

**Пример.**

*type rfile = file of real;*
*ima = file of integer;* .

В файл *rfile* можно выводить (или читать) числа типа *real*, а в файл *ima* – числа типа *integer*.

Чаще всего в языке Delphi используются текстовые файлы, которые определены следующим образом:

*type text = file of char;* ,

т.е. тип *text* является стандартным и описывать его не надо. Очевидно, что и любые числа также можно записывать в файлы этого типа в виде «изображения числа».

После того как определен тип файла, можно описать файловую переменную с помощью описания

*Var <имя> : < тип файла>;* .

**Пример.**

```
Var
  in: rfile;
  out: text;
  F: file of integer;
```

С помощью файловой переменной *in* можно обрабатывать файлы с числами типа *real*, *out* – текстовый файл, а *F* – с числами типа *integer*.

Для обработки стандартных файлов, которыми обычно являются

клавиатура (файл для ввода) и экран дисплея (файл для вывода), в языке Delphi описаны (поэтому пользователю их описывать не надо) файловые переменные *input* и *output*.

Методика связывания файловой переменной с конкретным периферийным устройством в значительной степени зависит от реализации, т.е. от типа компьютера и используемого транслятора.

Связь файловой переменной с периферийным устройством осуществляется с помощью вызова процедуры *Assign* вида

*procedure Assign (F: файловый тип; U: string);*

Здесь параметр *U* содержит информацию об устройстве, с которым связывается файл. Эта строка может принимать следующие значения:

- *''* (пустая строка) – стандартное устройство ввода и вывода;

- *'CON'* – консоль, посредством которой выводимая информация посылается на экран дисплея, а вводимая информация принимается с клавиатуры. Обычно консоль совпадает со стандартными устройствами ввода / вывода;

- *'LPT1', 'LPT2', 'LPT3'* – устройства печати. Если устройство только одно, то используется *'LPT1'*. Для него также возможно использовать *'PRN'*;

- *'COM1', 'COM2'* – устройства, представляющие собой последовательные телекоммуникационные порты;

- *'NUL'* – пустое устройство. Его рекомендуется применять, если результаты вывода не используются;

- *'спецификация файла на диске'* – указывает файл на внешнем носителе информации. Как обычно, спецификация должна содержать имя и тип файла. Если необходимо, указывается диск и путь.

**Примеры.**

*Assign(out, 'PRN');* {файл *out* связывается с принтером};

*Assign(in,'c:\user\stud\Ivanov\inf.dat');* {файловая переменная *in* связывается с файлом *inf.dat* на диске *c:* в каталоге *\user\stud\Ivanov*}.

Для того чтобы открыть файл, используются следующие процедуры:

*procedure Reset(F: файловый_тип [, длина_записи: word]);*

*procedure Rewrite(F: файловый_тип [, длина_записи: word]);*

*procedure Append(Var F: файловый_тип);*

Здесь квадратные скобки обозначают, что второй параметр – длина записи (в байтах) – может быть опущен. В этом случае будет использована стандартная длина записи.

*Процедура Reset* открывает существующий файл *F* для чтения. Текущий указатель файла устанавливается в начало файла. Если файл *F* отсутствует или уже открыт для записи, генерируется сообщение об ошибке.

*Процедура Rewrite* открывает файл *F* для записи. Информация в существующем файле затирается или создается новый файл. Текущий указатель файла устанавливается в начало.

*Процедура Append* открывает существующий файл *F* для записи так, чтобы его можно было продолжить. При этом текущий указатель файла устанавливается в конец и последующие операции записи дополняют существующий файл.

Необходимо отметить, что стандартные файлы *Input* и *Output* открываются системой языка Delphi автоматически, поэтому заботиться об их открытии не нужно.

**Примеры.**

*Reset(in);* {файл *in* открывается для чтения};

*Rewvrite(out);* {файл *out* открывается для новой записи};

*Append(out);* {файл *out* открывается для продолжения записи}.

Рассмотрим процедуры обработки файлов. Это уже известные процедуры

*procedure Read([Var F: text;] V1 [, V2, ...]);*

*procedure ReadLn([Var F: text;] V 1[, V2, ...]);*

*procedure Write([Var F: text;] P1 [, P2, ...]);*

*procedure WriteLn([Var F: text;] P1 [, P2, ...]);*

В квадратных скобках указаны необязательные параметры. Таким образом, первый параметр – файловую переменную *F* – можно опустить. Если это сделано, то ввод осуществляется из стандартного файла *Input* (клавиатура), а вывод – в стандартный файл *Output* (дисплей).

Параметры *V1, V2,...* могут быть литерного (*char*), целого (*integer, byte, word*), действительного (*real*) типа, а также строкового типа (*string*). В любом случае в файле всегда содержится литерная

информация. Параметры типа *real* и *integer* при выводе автоматически преобразуются в литерное представление соответствующего числа, а при вводе – наоборот, из литерного представления в число.

Поэтому при записи или считывании текущий указатель файла перемещается вперед на 1 символ для литер и на длину изображения числа при вводе чисел и останавливается на первом после числа пробеле, запятой или записи *Eoln*.

Параметры типа *string* вводятся от текущего указателя до конца строки.

При выводе параметры *P1, P2,...* могут быть записаны в форме *V1, V2,...* или *V1:M,...* или *V1:M:N*.

Здесь *V1, V2,...* имеют тот же смысл, что и при вводе. *M* и *N* – константы или переменные типа *integer*, определяющие соответственно количество символов для записи числа и число значащих цифр после точки.

При вводе элемента *Z* типа *string* вводится *Length(Z)* символов. Соответственно, текущий указатель файла перемещается на *Length(Z)* позиций вперед.

Процедуры *ReadLn* и *WriteLn* после окончания операций ввода / вывода осуществляют перевод строки и устанавливают текущий указатель файла в начало следующей строки.

**Примеры.**

*Write(out, R1, R2);* {осуществляет вывод в файл *out* значений двух переменных *R1, R2*};

*WriteLn(out, R1, R2);* {осуществляет вывод в файл *out* значений двух переменных *R1, R2*, после этого происходит перевод строки};

*Read(in, c);* {осуществляется вввод из файла *in* значения переменной *c*};

*ReadLn(in)*; {в файле *in* выполняется перевод строки, это может быть сделано, например, чтобы пропустить одну строку в файле *in*}.

Как уже указывалось, число записей в файле заранее неизвестно. Для того чтобы узнать, когда же файл закончился, используется специальная функция

*function Eof(F: файловый тип): boolean;*

которая вырабатывает значение *TRUE*, если текущий указатель указывает на конец файла. Таким образом, весь файл можно прочитать следующим образом:

```
While NOT Eof(in) do
begin
  Read(in, c);
  ……..{обработка c}
end;
```

Кроме того, иногда полезно узнать, не является ли следующий символ, который будет прочитан из файла, признаком конца строки. Это можно сделать с помощью функции

*function Eoln(f: файловый тип): boolean;*

которая вырабатывает значение *TRUE*, если текущий указатель установлен на конец строки. Например, прочитать строку до конца можно следующим образом:

```
While NOT Eoln(in) do
begin
  Read(in, c),
  ……{обработка c}
end;
ReadLn(in); {когда строка закончилась – переход на следующую}
```

После того как работа с файлом закончена, его надо закрыть с помощью процедуры

*procedure Close(f: файловый_тип);* .

Например, вызов

*Close(out);*

закрывает файл *out*.

При закрытии выходного файла в него помещается служебная запись *Eof*, имя файла заносится в оглавление диска и производится ряд служебных операций по обработке периферийных устройств.

При закрытии вводного файла освобождаются файловая переменная и периферийное устройство, и их можно использовать для других целей.

Стандартные файлы *input* и *output* закрываются автоматически, и заботиться об их закрытии нет необходимости.

При работе с файлами необходимо внимательно следить за очередностью ввода и вывода информации и за типами обрабатываемых переменных. Например, запись файла

1.0 *Eoln*

может быть преобразована в действительное число при считывании действительного числа или введена как последовательность литер «1», «.» и «0». В связи с этим рекомендуется для считывания информации из файла использовать тот же порядок вызовов процедур и их аргументов, какой был при записи файла с помощью процедур записи.

Например, если файл записан последовательностью операторов

……
*for i:= 1 to 100 do WriteLn(out, X, Y, Z);* ,

то желательно и считывать из него информацию таким же образом

……
*for i:=1 to 100 do ReadLn(out, X, Y, Z);* .

## ПРАКТИЧЕСКАЯ РАБОТА № 6

### 6.1. Цель работы

Получение навыков работы с файлами на алгоритмическом языке Delphi и обработка текстовой информации.

### 6.2. Задача работы

Разработать программу, позволяющую обрабатывать файл сведений о студентах в соответствии с вариантом задания.

**Вариант 1.** Разработать программу, позволяющую добавлять новых студентов в список.

**Вариант 2.** Разработать программу, позволяющую удалять студентов из списка в связи с их отчислением.

**Вариант 3.** Разработать программу для составления списка студентов, обучающихся в группе *X*.

**Вариант 4.** Разработать программу для поиска студентов старше *X* лет по состоянию на текущий день.

**Вариант 5.** Разработать программу для поиска студента, успевающего лучше всех (рекомендуется распечатать все его оценки). В качестве критерия выбрать среднее арифметическое всех оценок.

**Вариант 6.** Разработать программу для поиска студента, успевающего хуже всех (рекомендуется распечатать все его оценки). В качестве критерия выбрать среднее арифметическое всех оценок.

**Вариант 7.** Разработать программу для составления списка успевающих студентов (не имеющих задолженностей).

**Вариант 8.** Разработать программу для составления списка неуспевающих студентов (имеющих задолженности).

**Вариант 9.** Разработать программу для печати письма студентам, имеющим более трех задолженностей, с предупреждением об отчислении, если «хвосты» не будут сданы до срока *X* (рекомендуется указать предметы и циклы, по которым имеются задолженности).

**Вариант 10.** Разработать программу для корректировки ведомости после пересдачи студентом *X* предмета *Y* цикла *Z* на *N* баллов.

**Вариант 11.** Разработать программу для корректировки ведомости, если студентка *X* сменила фамилию на *Y*.

**Вариант 12.** Разработать программу для составления итоговой ведомости, в которой итоговая оценка по предмету равна среднеарифметической по трем циклам. Округление производится по обычным правилам арифметики.

Во всех вариантах заданий используется файл *USP.DAT*, содержащий сведения об успеваемости студентов в некотором семестре. Сведения должны быть записаны в соответствии со следующей структурой (указаны номера позиций символов с строке):

```
1          13   25          37  38          49          54          60          66          72          77 Eoln
|Фамилия|Имя |Отчество|Пол|Дата рожд.|Группа|Высш. мат.|Физика|История|Информат.|
```

Вторая и последующие строки

```
1          13   25          37  38          49          54          60          66          72          77 Eoln
|Иванов   |Иван |Иванович |М  |05 10 1990 |Р-67 |3   4   5 |5   2   3 |4   4   4 |5   5   5 |
```

Пол кодируется буквами *М* или *Ж*. Дата рождения записывается в формате: *ДД.ММ.ГГГГ*. Успеваемость по каждому предмету записывается в виде оценок по пятибалльной шкале по трем циклам.

В результате обработки необходимо создать новый файл *NEW.DAT*, содержащий систематизированные сведения в соответствии с вариантом задания.

### 6.3. Домашнее задание

1. Изучить основные процедуры и функции, используемые для работы с файлами.

2. Выполнить формализацию задачи. Требования к разрабатываемой программе следующие. Она должна обеспечить ввод ис-

ходных данных (фамилии, оценки и так далее). Затем с помощью этой же программы необходимо производить чтение информации из созданного входного файла. По ходу чтения должна выполняться обработка прочитанных сведений. Если найдены требуемые записи, то они (может быть, после необходимой корректировки) должны быть перенесены в выходной файл.

Для облегчения разработки программы следует использовать типы, объявленные пользователем, в особенности тип *record*. Для обработки и сравнения переменных типа *string* желательно использовать стандартные процедуры и функции.

3. Разработать алгоритм и программу для решения задачи.

4. Составить тесты для проверки программы.

5. Подготовиться к ответам на контрольные вопросы.

### 6.4. Лабораторное задание

1. Создать и сохранить в файле текст программы, разработанной при выполнении домашнего задания.

2. Выполнить отладку и тестирование программы.

3. Создать на диске файл *USP.DAT*, содержащий несколько строк в соответствии с подразд. 7.2.

4. Произвести обработку информации из исходного файла *USP.DAT* и записать ее в новый файл *NEW.DAT*. Для контроля правильности работы вывести содержание исходного и нового файлов на экран монитора или на принтер.

### 6.5. Содержание отчета

1. Название, цель и задача работы.

2. Формальная постановка задачи.

3. Алгоритм и программа для решения поставленной задачи.

4. Тест.

5. Текст программы, содержимое исходного и нового файлов, анализ ошибок.

### Контрольные вопросы

1. Разработайте подпрограмму для копирования файлов.

2. Разработайте программу, которая определяет максимальную длину строк в текстовом файле *F*.

3. Напишите программу, которая подсчитывает число строк в текстовом файле *F*.

4. Разработайте процедуру *delcopy(F1, F2: text)*, переписываю-

щую содержимое файла *F1*, но без пустых строк, в файл *F2*.

5. Исходный текстовый файл разделите на 3 части, равные по количеству строк, и запишите их в 3 различных файла.

6. Имеется текстовый файл *F1.txt*, содержащий связный литерный текст. Разработайте программу, которая выводит на экран каждое предложение с новой строки.

7. Составьте процедуру, которая преобразует исходный текстовый файл *FVR* со строками различной длины в файл *FFIX* со строками длиной 80 символов (длинные строки разбиваются на несколько строк, короткие дополняются до 80 символов пробелами).

8. В текстовом файле *F1.txt* строки имеют длину *L1* (например, 80 символов). Разработайте подпрограмму для переформатирования текста таким образом, чтобы в выходном файле *F2.txt* были строки длиной *L2* (например, 60 символов).

9. Напишите процедуру (и программу для ее проверки), которая построчно печатает содержимое непустого текстового файла *T*, вставляя в начало печатаемой строки ее порядковый номер. Номер должен занимать 4 позиции и отделяться от строки пробелом.

10. Составьте логическую подпрограмму-функцию, определяющую, имеется ли в текстовом файле *SLV* заданное слово.

11. Напишите программу, которая из файла *ProgTST.pas* типа *text* считывает исходный текст программы на языке Delphi, в начало каждой строки подставляет её номер (в виде комментария языка Delphi) и переписывает полученный текст в новый файл *ProgOUT.pas*.

12. В файле *PPG* типа *text* записано несколько текстов программ на языке Delphi. Первая строка каждой программы начинается ключевым словом *Program* и следующим за ним через пробел именем программы. Разработайте процедуру, печатающую имена всех программ.

13. В файле *FPS* типа *text* записана программа на языке Delphi. Составьте процедуру, выдающую сообщения об ошибке, если в тексте программы встретился оператор *end*, без предшествующего ему оператора *begin*.

14. В файле *PPF* типа *text* записаны тексты подпрограмм на языке Fortran, которые начинаются с ключевых слов *SUBROUTINE* или *FUNCTION*. Составьте процедуру, которая распечатывает тексты каждой подпрограммы с новой страницы, по 40 строк на листе, с нумерацией страниц вверху посередине листа в формате «– 1 –».

15. В файле *F1* типа *text* записана программа на языке Fortran. Комментарии в программе отмечаются символом «*C*» в первой позиции строки. Разработайте подпрограмму-функцию, подсчитывающую число строк операторов программы (без комментариев).

16. Имеется файл *SK* типа *text*. Составьте подпрограмму-функцию типа *boolean*, проверяющую, правильно ли расставлены круглые и квадратные скобки (т.е., совпадает ли число левых и правых скобок) в тексте файла.

17. В текстовом файле *F1.dat* содержатся действительные числа. Перепишите в файл *F2.dat* положительные, а в файл *F3.dat* – отрицательные числа.

18. Имеется файл *C* типа *file of integer*. Составьте логическую подпрограмму-функцию для определения, является ли количество чисел в файле четным.

19. В файле *FINT* типа *file of integer* записана последовательность целых чисел. Составьте подпрограмму-функцию, определяющую, сколько чисел в последовательности больше по величине суммы своих «соседей» (предыдущего и последующего числа).

20. Дано описание *type ryad = file of 0..999;* Напишите логическую подпрограмму-функцию, проверяющую, упорядочены ли по возрастанию элементы непустого ряда, содержащегося в файле.

21. Файл *EF* имеет тип *file of integer*. Разработайте подпрограмму-функцию, определяющую, сколько раз в записанной в файле последовательности чисел меняется их знак.

22. В файле *FR* типа *file of real* записана последовательность действительных чисел. Составьте подпрограмму-функцию, вычисляющую среднее арифметическое чисел, записанных в файле.

23. Составьте процедуру (и программу для ее проверки), копирующую строки из одного текстового файла в другой файл в обратном порядке.

24. Разработайте подпрограмму-функцию логического типа, определяющую, является ли содержащаяся в файле *F99* целая квадратная матрица размером 9×9 элементов «магическим квадратом», в котором сумма элементов во всех строках и столбцах одинакова.

25. Запишите в файл типа *file of integer* шестизначные номера всех «счастливых билетов». («Счастливым» называется билет, в котором сумма цифр первой половины номера равна сумме цифр второй половины номера.)

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК


1. **Федосов А.Г.** Delphi 3.0 для всех. – 3-е изд. – М.: КомпьютерПресс, 1998. – 543 с.

Содержится описание интегрированной среды разработчика, визуальных компонентов, техники программирования и использования среды Delphi для разработки Windows-приложений. Особое внимание уделено практическому программированию – в книге приведено множество примеров использования компонентов и функций Windows API.

Рекомендуется для самостоятельного обучения и в качестве справочного пособия для широко круга программистов.

2. **Фаронов В.В.** Delphi 4: учеб. курс.– М.: Нолидж, 1998.– 448 с.

Рассмотрены основные сведения о среде программирования Delphi.

Рассчитана на читателей, не знакомых с программированием вообще или имеющих небольшой опыт программирования.

3. **Фаронов В.В.** Delphi 5: учеб. курс. – М: Нолидж, 2000. – 605 с.

Даны начальные сведения о системе программирования Delphi.

Рассчитана на широкий круг читателей, как начинающих программистов, так и имеющих опыт программирования.

4. **Стивенс Р.** Delphi: готовые решения. – М: ДМК, 2001. – 378 с.

Рассматриваются типичные и наихудшие случаи реализации алгоритмов. Подробно описаны важнейшие элементы алгоритмов хранения и обработки данных.

Предназначена для начинающих программистов.

5. **Климова Л.М.** Delphi 7: основы программирования, решение типовых задач: самоучитель. – 3-е изд. – М: КУДИЦ-ОБРАЗ, 2006. –480 с.

Содержит описание компонентов, необходимых для решения типовых задач в среде программирования Delphi.

Предназначена для студентов технических специальностей, школьников старших классов, может использоваться для самообразования.

6. **Фаронов В.В.** Delphi: программирование на языке высокого уровня: учебник для студ. вузов. – СПб.: Питер, 2003. – 639 с.

Изложен язык программирования Delphi, приемы программирования в среде Delphi, ее главные составные части – интерфейс, галереи компонентов, технология COM и система Model Maker.

Может быть использована в качестве пособия для первоначального изучения среды языка Delphi и для пополнения знаний в области применения языка Delphi.

7. **Архангельский А.Я.** Приемы программирования в Delphi. – М.: БИНОМ, 2003. – 780 с.

Содержится справочный материал по всем версиям Delphi, снабженный подробными комментариями и программами. Подробно рассмотрена работа с исключениями, текстовыми и двоичными файлами, строками разных типов, массивами, множествами, записями, классами.

Как справочник полезна пользователям любой квалификации.

8. **Баас Р., Фервай М., Гюнтер Х.** Delphi 5: Пер. с нем. – Киев: Ирина: BHV, 2000. – 494 с.

Рассмотрены ключевые моменты объектно-ориентированного программирования, подробно описана интегрированная среда разработки приложений, способы использования и создания компонентов.

Предназначена для специалистов с различным уровнем подготовки.

9. **Delphi**: советы программистов / Под ред. В. Озерова. – СПб.: Символ, 2002. – 908 с.

Приведены рекомендации по составлению программ различного назначения в среде языка Delphi.

Полезна и начинающим программистам, и специалистам.

10. **Карпов Б.** Delphi: спец. справочник. –СПб.: Питер, 2002. – 684 с.

Содержится справочный материал по основным элементам языка Delphi. Подробно описан процесс создания баз данных, Windows-приложений, динамических библиотек. Приведены коды ошибок выполнения программ.

Рекомендована как начинающим программистам, так и специалистам.

11. **Осипов Д.** Delphi: профессиональное программирование. – СПб.: Символ-Плюс, 2006. – 1055 с.

Изложены основы языка программирования Delphi, рассмотрен процесс разработки собственных компонентов, изложено программирование на Win 32 API, особенности построения сетевого программного обеспечения.

Рассчитана на подготовленных пользователей ПК, желающих самостоятельно научиться программированию в среде Delphi.

12. **Сорокин А.В.** Delphi: разработка баз данных. – СПб.: Питер, 2005. – 474 с.

Подробно рассмотрены все наиболее распространенные серверы баз данных.

Рассчитана на начинающих программистов и тех, кто хочет углубить свои знания в сфере разработки баз данных.

13. **Епанешников А.М., Епанешников В.А.** Delphi: проектирование СУБД. – М.: Диалог-МИФИ, 2001. – 528 с.

Рассматриваются вопросы проектирования систем управления базами данных на основе средств среды визуального программирования Delphi. Приведен пример создания системы управления кадровой базы данных.

Предназначена для начинающих изучение вопросов создания СУБД в Delphi.

14. **Тюкачев Н.** Delphi 5: создание мультимедийных приложений. – СПб.: Питер, 2001. – 398 с.

Собран и систематизирован справочный материал о графических возможностях Delphi, который проиллюстрирован многочисленными примерами.

Для желающих научиться создавать графические изображения средствами Delphi.

15. **Бобровский С.** Delphi 5: учеб.курс. – СПб.: Питер,2000. –638 с.

Особое внимание уделено принципам и практическим приемам создания сетевых приложений для разных архитектур и разработке программ, поддерживающих основные протоколы Интернет.

Не требует специальной подготовки, может использоваться как пособие для изучения основ сетевых технологий и как справочник по компонентам Delphi.

# ПРИЛОЖЕНИЕ

**Образец оформления отчета о выполнении практической работы**

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ РАДИОТЕХНИЧЕСКИХ СИСТЕМ И УПРАВЛЕНИЯ
Кафедра АиРПУ

ОТЧЕТ
о выполнении практической работы №____
(название практической работы)

Выполнил студент гр. ___ _____
(Фамилия, имя, отчество)

Принял доц. каф. АиРПУ _____
(Фамилия, имя, отчество)

Таганрог 2014

---------- (с нового листа) ----------

Цель работы:

Задача работы:

Формализация задачи:

Вывод математических соотношений:

Алгоритм и программа:

Тесты:

Результаты:

Выводы:

**СОДЕРЖАНИЕ**


ВВЕДЕНИЕ ……..…..... 3

1. ИСПОЛЬЗОВАНИЕ КОМПЬЮТЕРНОЙ ТЕХНИКИ ДЛЯ РЕШЕНИЯ ИНЖЕНЕРНЫХ И НАУЧНЫХ ЗАДАЧ …….. 4
2. ЛИНЕЙНЫЕ АЛГОРИТМЫ …….. 18
3. ВЕТВЯЩИЕСЯ АЛГОРИТМЫ ……. 27
4. ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ. ТАБУЛИРОВАНИЕ ФУНКЦИЙ …… 37
5. ПОДПРОГРАММЫ ……. 44
6. ОБРАБОТКА ФАЙЛОВ ……..…… 55

БИБЛИОГРАФИЧЕСКИЙ СПИСОК …… 66

ПРИЛОЖЕНИЕ …… 68

Учебное издание

**КОШКИДЬКО ВЛАДИМИР ГЕОРГИЕВИЧ**
**ПАНЫЧЕВ АНДРЕЙ ИВАНОВИЧ**
**ДЯТЛОВ ПАВЕЛ АНАТОЛЬЕВИЧ**

# ПРАКТИКУМ ПО ИНФОРМАТИКЕ

Часть 1

Учебное пособие

*Ответственный за выпуск* Кошкидько В.Г.
*Редактор* Надточий З.М.
*Корректор* _____

Формат $60\times84^{1}/_{16}$ Подписано в печать _._____.2014 г.

Усл. п. л. – 4,3 Уч.-изд. л. – 4,2

Заказ № _____ Тираж **50** экз.

---

Издательство Южного федерального университета
344091 г. Ростов – на –Дону, пр. Стачки, 200/1
Тел. 8(863)247-80-51
Отпечатано в Секторе обеспечения полиграфической продукцией кампуса в г. Таганроге отдела полиграфической, корпоративной и сувенирной продукции ИПК КИБИ МЕДИА ЦЕНТРА ЮФУ
Тел. 8(8634)371717
ГСП 17А, Таганрог, 28, Энгельса, 1