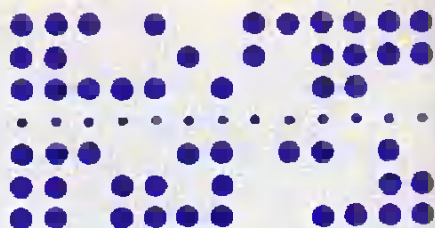


БИБЛИОТЕЧКА  
ПРОГРАММІСТА



С.А. АБРАМОВ  
Е.В. ЗИМА

# Начала информатики



БИБЛИОТЕЧКА  
ПРОГРАММИСТА

---

С. А. АБРАМОВ, Е. В. ЗИМА

# НАЧАЛА ИНФОРМАТИКИ



МОСКВА «НАУКА»  
ГЛАВНАЯ РЕДАКЦИЯ  
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ  
1989

Абрамов С. А., Зима Е. В. Начала информатики.— М.: Наука, Гл. ред. физ.-мат. лит., 1989.— 256 с. (Библиотечка программиста).— ISBN 5-02-013958-0.

Содержит систематизированное изложение основных понятий и методов информатики (вычислительной техники, программирования, численных методов, сортировки и поиска). Изложение основ программирования проводится на языке Паскаль, более перспективном для персональных компьютеров (по сравнению с языком Бейсик). Имеется много примеров и задач.

Для начинающих программистов, пользователей, студентов вузов, старшеклассников, преподавателей.

Табл. 1. Ил. 106.

## Научное издание

АБРАМОВ Сергей Александрович, ~~ЗИМА Евгений Викторович~~  
НАЧАЛА ИНФОРМАТИКИ

«Библиотечка программиста», выпуск 60

Заведующий редакцией А. С. Косов

Редактор О. И. Сухова

Художественный редактор Т. Н. Кольченко

Технический редактор Е. В. Морозова. Корректор И. Я. Кристаль

ИБ № 32686

Сдано в набор 20.10.88. Подписано к печати 24.07.89. Формат 84×108/32. Бумага тип. № 2. Гарнитура литературная. Печать высокая. Усл. печ. л. 18,44. Усл. кр.-отт. 18,65. Уч.-изд. л. 15,98. Тираж 150 000 экз. Заказ № 8—100. Цена 1 руб.

Ордена Трудового Красного Знамени издательство «Наука»  
Главная редакция физико-математической литературы  
117071 Москва В-71, Ленинский проспект, 15

Ордена Октябрьской Революции и ордена Трудового Красного Знамени  
МПО «Первая Образцовая типография» Государственного комитета  
СССР по делам издательств, полиграфии и книжной торговли. 113054  
Москва, Валуевская, 28

Отпечатано на полиграфическом комбинате ЦКЛКСМ Украины «Молодь»  
Ордена Трудового Красного Знамени издательство-полиграфического  
объединения ЦК ВЛКСМ «Молодая гвардия» 252119 г. Киев-119,  
ул. Пархоменко, 38—41.

А 1404000000—104  
053(02)—89 128-89

ISBN 5-02-013958-0

© Издательство «Наука»,  
Главная редакция  
физико-математической  
литературы, 1989

## ОГЛАВЛЕНИЕ

Предисловие . . . . .	5
Введение (предварительные сведения) . . . . .	7
<b>Глава I. Первое знакомство с программированием . . . .</b>	<b>14</b>
§ 1. О записи программы. Выражения . . . . .	14
§ 2. Операторы присваивания, ввода и вывода . . . .	18
§ 3. Простейшая программа . . . . .	22
§ 4. Условный оператор . . . . .	28
§ 5. Составной оператор. Дополнительные возможности вывода информации . . . . .	33
§ 6. Оператор цикла . . . . .	36
<b>Глава II. Простейшие алгоритмы приближенных вычислений, компьютерной графики, символьной обработки . . . . .</b>	<b>44</b>
§ 7. Приближенное решение уравнений . . . . .	44
§ 8. Целые числа . . . . .	51
§ 9. Дополнительные операции над целыми числами. Округление . . . . .	56
§ 10. Оператор цикла с параметром . . . . .	60
§ 11. Простейшие графические возможности компьютера	67
§ 12. Приближенное вычисление площади криволинейной трапеции . . . . .	78
§ 13. Оператор перехода. Пустой оператор . . . . .	86
§ 14. Вложенные операторы цикла . . . . .	89
§ 15. Составные условия . . . . .	97
§ 16. Тип <i>char</i> . . . . .	103
<b>Глава III. Вычисления с хранением последовательности значений. Моделирование. Сортировка . . . . .</b>	<b>110</b>
§ 17. Массивы. Пример математической модели в биологии	110
§ 18. Раскрашивание фигур. Графические построения с использованием массивов . . . . .	118
§ 19. Константы . . . . .	126
§ 20. Таблицы значений функций и линейная интерпо- ляция . . . . .	132
§ 21. Поиск элемента в упорядоченном массиве . . . .	141
§ 22. Упорядочивание (сортировка) массива . . . . .	147
<b>Глава IV. Матрицы . . . . .</b>	<b>152</b>
§ 23. Массивы массивов. Матрицы . . . . .	152
§ 24. Решение систем линейных уравнений . . . . .	159

Глава V. Файлы . . . . .	171
§ 25. Файлы. Файловые типы . . . . .	171
§ 26. Классификация файлов . . . . .	178
§ 27. Записи. Комбинированные типы . . . . .	183
Глава VI. Использование вспомогательных алгоритмов (процедуры и функции) . . . . .	189
§ 28. Процедуры без параметров. Параметры — переменные	189
§ 29. Параметры-значения . . . . .	197
§ 30. Построение диаграмм . . . . .	203
§ 31. Функции . . . . .	209
§ 32. Построение графиков функций . . . . .	214
Глава VII. Более сложные примеры алгоритмов и программ	221
§ 33. Лингвистический пример (перенос слова с одной строки на другую) . . . . .	221
§ 34. Дифференциальные уравнения . . . . .	228
§ 35. Некоторые задачи поиска . . . . .	243

## ПРЕДИСЛОВИЕ

В этой книге излагаются основные понятия и методы информатики. Главное внимание уделяется программированию, численным методам, графике, сортировке, поиску, вычислительной технике. Рассматриваются также, хотя и в меньшем объеме, вопросы информатики, имеющие отношение к моделированию и лингвистике.

Изложение основ программирования проводится с использованием языка Паскаль — одного из наиболее распространенных языков программирования. Точнее говоря, используется сокращенный вариант Паскаля. Сокращенным вариантом можно овладеть быстрее и легче, чем всем языком, а возможностей его вполне достаточно для написания довольно сложных программ (подчеркнем, что всякая программа, правильно написанная на сокращенном Паскале, будет правильной и в смысле полного Паскаля). Авторы считают, что при изучении начал информатики лучше познакомиться с настоящим языком программирования (например, в сокращенном объеме), чем тратить время на специальный «педагогический» язык, не имеющий применения в реальных исследованиях и разработках. Разумеется, выбор сокращенного варианта того или иного настоящего языка является серьезной методической проблемой.

Для первого знакомства с информатикой по предлагаемой книге можно ограничиться прочтением введения и первых трех ее глав, что под силу (как кажется авторам) учащимся старших классов общеобразовательной школы. Учащиеся школ с углубленным изучением математики и студенты техникумов, а также начинающие профессиональные программисты, смогут прочитать книгу до конца; § 24 и всю последнюю главу целесообразно рассматривать как дополнительный (факультативный) материал. Отметим, что программа курса информатики московской школы № 52 приблизительно соответствует содержанию книги.

Книга построена таким образом, что читатель довольно быстро привлекается к самостоятельному составлению осмысленных про-

грамм. Простейшие программы для компьютеров можно составлять уже после прочтения первых трех параграфов книги.

Изложение сопровождается большим числом примеров. Помимо этого каждый из параграфов снабжен задачами для самостоятельного решения: вначале идет несколько легких задач, затем — задачи более сложные. Некоторые задачи существенно углубляют материал параграфа, поэтому полезно по крайней мере прочитывать условия всех задач, сопровождающих параграф.

## ВВЕДЕНИЕ (ПРЕДВАРИТЕЛЬНЫЕ СВЕДЕНИЯ)

**История вычислительной техники.** В XVII веке был создан целый ряд вычислительных аппаратов—арифмометров. Исходные числа в этих аппаратах задавались поворотами наборных колес, вращение ручки приводило в движение различные шестерни и валики, в итоге специальные колеса с цифрами приобретали положение, соответствующее результату выполнения одной из простейших арифметических операций. Этой операцией могло быть сложение или вычитание, лучшие арифмометры позволяли выполнять также умножение и деление. Среди изобретателей арифмометров были, в частности, Б. Паскаль и Г. Лейбниц—выдающиеся ученые XVII века, обладающие широчайшим кругом научных интересов. Скудность возможностей первых вычислительных аппаратов не помешала Паскалю и Лейбницу высказать ряд интересных идей, касающихся роли вычислительной техники в научных исследованиях будущего. Так, Лейбниц писал о машинах, которые будут пригодны не только для работы с группами цифр, изображающими числа, но и с группами символов, изображающими формулы, тексты и т. д. Эти машины представлялись Лейбницу способными правильно выполнять действия логического характера (вывод формул и т. д.) подобно тому, как арифмометры правильно выполняют арифметические действия. Эта идея большинству современников Лейбница казалась полным абсурдом. В начале XVIII века взгляды Лейбница были осмеяны великим английским сатириком Дж. Свифтом в его знаменитом романе «Путешествия Гулливера». Лишь в XX веке стала понятной значительность этой идеи.

Поиск новых средств автоматизации вычислений продолжался. В XIX веке английский математик Ч. Бэббидж, занимавшийся составлением таблиц для навигации, разработал проект вычислительной машины (названной им «аналитической машиной»), в основе которого лежал принцип, имеющий громадное значение и для современной вычислительной техники—принцип программного управления работой вычислительной машины (об этом принципе мы поговорим позднее). Новаторская мысль Бэббиджа была подхвачена и развита его ученицей Адой Лавлейс, дочерью поэта Дж. Байрона.



Однако практическая реализация идеи Бэббиджа была в то время невозможной, так как эта идея существенно опережала технические возможности своего века.

Вычислительная техника оставалась весьма несовершенной до сороковых годов XX века: даже электрические арифмометры и большие машины, построенные на электромагнитных реле, затрачивали на умножение чисел по несколько секунд. Например, при кораблестроительных расчетах требовались месяцы и даже годы работы целых бюро. В сороковых годах XX века произошел коренной переворот в вычислительной технике. Появились вычислительные машины, в которых использовались уже не колеса, валики, электромагнитные реле и т. д., а электронные элементы. Этими элементами первоначально были радиолампы. Переход от электро-механических элементов к электронным сразу увеличил быстродействие вычислительной техники в сотни раз. Первой действующей электронной вычислительной машиной (ЭВМ) была машина ЭНИАК, изготовленная в США в конце 1945 г. Авторами проекта были Дж. Эккерт и Дж. Моучли.

В СССР вычислительная машина МЭСМ была создана в 1951 г. под руководством академика С. А. Лебедева. Впоследствии электронные вычислительные машины начали производить во многих странах.

Конструкции вычислительных машин непрерывно совершенствовались. В 60-х годах вместо радиоламп стали применяться транзисторы, которые впоследствии уступили место выполненным на поверхностях кристаллов микросхемам. Вычислительные машины стали меньше по размерам и значительно дешевле, их быстродействие возросло к настоящему времени до нескольких миллионов операций в секунду. Это открыло возможность широкого применения этих машин в науке, конструировании, планировании, управлении, справочных службах, делопроизводстве, обучении и т. д. — здесь можно перечислить очень многие сферы научной и практической деятельности человека.

В последние годы вычислительные машины во всем мире стали называться компьютерами (от английского слова computer — вычислитель). Этот термин принят и в нашей стране, хотя наряду с ним довольно часто употребляется прежнее название — ЭВМ (электронная вычислительная машина). Далее в этой книге мы будем пользоваться словом «компьютер».

Несмотря на то, что в названии «компьютер» как бы подразумевается, что машина рассматриваемого вида предназначена для выполнения исключительно вычислительных работ, на самом деле возможности современных компьютеров значительно шире: они могут выполнять графические работы, различные преобразования текстов и многое другое. Это, в частности, подтверждает справед-

ливость идей Лейбница, высказанных им около 300 лет тому назад.

**Наука информатика.** Стремительное развитие и широкое распространение вычислительной техники послужили предпосылками к появлению нового раздела науки, названного информатикой. Основные направления информатики связаны с разработкой специальных компьютерных методов решения сложных исследовательских и практических задач. В нашей стране большой вклад в становление и развитие информатики внесли академики В. М. Глушков, А. А. Дородницын, А. П. Ершов, Л. В. Канторович, М. В. Келдыш, М. А. Лаврентьев, С. Л. Соболев, А. Н. Тихонов.

Информатика долгое время рассматривалась как часть математики и развивалась усилиями математиков. И хотя специфичность компьютерных методов дает основание говорить об информатике как об особой науке, все же следует подчеркнуть, что эта наука существенно опирается на достижения математики. Последнее объясняется тем, что явления и процессы, которые изучаются естественными и техническими науками, экономические процессы и т. д. часто удается описать с помощью понятий математики — функций, систем уравнений, неравенств и др., и для получения конкретных сведений об изучаемых явлениях и процессах надо произвести некоторые действия (например, прибегая к помощи компьютера) над математическими объектами \*).

Название «информатика» происходит от слова «информация», которое, в свою очередь, означает сведения о чем-либо \*\*). Компьютер фактически предназначен для переработки информации. Исходная информация о некотором процессе, имеющая вид чисел, таблиц, графиков, текстов и т. д., может быть преобразована в другую информацию об этом же процессе. Например, информация о взаимном расположении планет может быть с помощью компьютера довольно быстро преобразована в информацию о расположении, которое будет наблюдаться через интересующее нас время. Информация о производственных возможностях предприятия может быть преобразована в информацию о таком распределении работы, которое обеспечивает эффективное использование всех имеющихся возможностей и т. д. Но для того, чтобы решение подобной задачи стало возможным, надо сначала разработать алгоритм и написать соответствующую программу для компьютера.

Обсудим, в общих чертах, понятия алгоритма и программы.

---

\*) Описание некоторого явления или процесса с помощью понятий математики часто называют математической моделью этого явления или процесса.

\*\*) За рубежом, особенно в англоязычных странах, вместо слова «информатика» часто употребляют слова «компьютерная наука» (computer science).

**Алгоритмы и программы.** Понятие алгоритма является одним из центральных понятий информатики. Слово алгоритм, в сущности, является синонимом слов способ, рецепт, и т. д. Можно говорить, в этом смысле, об алгоритме нахождения корней квадратного уравнения, заданного своими коэффициентами, или об алгоритме разложения натурального числа на простые множители. В основе этих алгоритмов лежат простейшие математические операции над числами. Такие алгоритмы называются численными. Довольно часто рассматриваются и нечисленные алгоритмы. Например, в роли исходных данных и результатов могут выступать последовательности символов—тексты, формулы и т. д., в роли операций—не операции сложения, умножения и подобные им, а операции приписывания одной последовательности к другой, операция замены по некоторой таблице одних символов на другие и т. д. Примером алгоритма, основывающегося на подобных операциях, являются алгоритмы преобразования текста в его код Морзе. В школьном курсе геометрии рассматриваются задачи на построение с помощью циркуля и линейки. В этих задачах требуется указать способ, или, другими словами, алгоритм построения искомой фигуры по исходным данным. В качестве исходных данных и результатов здесь выступают совокупности точек, прямых окружностей, а в качестве операций—проведение прямых линий и окружностей. Итак, алгоритм—это описанный со всеми подробностями способ получения удовлетворяющих поставленному условию результатов по исходным данным.

Поиски различных алгоритмов входили в круг важных задач во все время существования науки. Уже в древнейшие времена были получены способы нахождения площадей и объемов геометрических фигур и тел по их размерам. Одним из достижений античной науки было изобретение Евклидом (III век до н. э.) необычно остроумного способа быстрого нахождения наибольшего общего делителя двух натуральных чисел. Математики Древнего Востока изобрели десятичную систему и дали правила вычислений в этой системе. Сам термин «алгоритм» тоже имеет древнее происхождение, являясь латинизированной транскрипцией имени великого среднеазиатского ученого IX века Мухаммеда аль-Хорезми (буквально: Мухаммеда из Хорезма или Мухаммеда Хорезмского; Хорезм—название древнего государства на территории Узбекистана). В математическом трактате Мухаммеда аль-Хорезми формулировались, в частности, правила разнообразных вычислений.

Ряд важных для вычислительной практики алгоритмов был разработан в XVII—XIX веках И. Ньютоном, Л. Эйлером, К. Ф. Гауссом—крупнейшими математиками своего времени. Эти алгоритмы не утратили своего значения по сегодняшний день.

В настоящее время интерес к алгоритмам особенно велик благодаря упомянутой возможности использования компьютеров в технике, экономике, научных исследованиях и т. д. Дело здесь в том, что компьютер во время работы выполняет заданную программу, а программа является некоторым алгоритмом, записанным в специальных обозначениях. Соответствующую систему обозначений называют языком программирования. Точнее, язык программирования — это совокупность средств и правил представления алгоритма в виде, приемлемом для компьютера.

Число используемых языков программирования сейчас достаточно велико. Среди наиболее распространенных следует в первую очередь назвать такие языки, как Фортран, Алгол, Бейсик, Лисп, Паскаль, Модула, Си, Ада \*). Язык программирования Паскаль, созданный в 70-х годах швейцарским ученым Н. Виртом, приобрел в последнее время значительную популярность. В Паскале сконцентрированы многие лучшие черты языков-предшественников. В нашем курсе информатики в качестве рабочего языка программирования будет использоваться именно Паскаль \*\*).

В связи с понятием программы уточним принцип программно-го управления работой компьютера (этот принцип, как говорилось выше, был сформулирован еще Бэббиджем в XIX веке): программа вводится в компьютер, и в зависимости от вида программы компьютер выполняет ту или иную работу. Этот принцип станет более понятным после рассмотрения общей схемы компьютера.

**Современные компьютеры.** Основными компонентами компьютера являются процессор, память, устройства ввода и вывода. С помощью устройства ввода программа и исходные данные попадают в память. Программа содержит последовательность инструкций, которую выполняет процессор. Результаты выполнения программы поступают в устройство вывода. В качестве устройства ввода, как правило, используется клавиатура (напоминающая клавиатуру пишущей машинки), в качестве устройства вывода — дисплей (телевизионный экран, на котором высвечиваются результаты выполнения программ) или принтер (устройство, печатающее результаты на бумаге).

Наряду с клавиатурой, дисплеем и принтером используются дисководы и магнитофоны — устройства, осуществляющие запись информации на магнитные носители (магнитные диски и магнитные ленты) и считывание информации с магнитных носителей. Наиболее распространенный вид дисков — гибкие диски (дискеты), круглые пластинки диаметра 133 или 89 мм с магнитным покрытием.

---

\*) Название Паскаль дано языку программирования в честь Блеза Паскаля, название Ада — в честь Ады Лавлейс.

\*\*) Точнее, сокращенный вариант Паскаля.

Один и тот же дисковод (магнитофон) может использоваться и как устройство ввода, и как устройство вывода. Это означает, что результаты, полученные на некотором этапе выполнения программы и записанные на магнитный диск (ленту), могут быть использованы на последующих этапах выполнения программы (или при выполнении другой программы) как исходные данные. Такая возможность становится особенно привлекательной в том случае, когда объем промежуточных результатов слишком велик, и все они не могут поместиться в памяти компьютера.

Персональный компьютер имеет небольшие размеры и может быть размещен на письменном столе (рис. 1). В его состав входят системный блок, содержащий процессор, память и дисководы (на

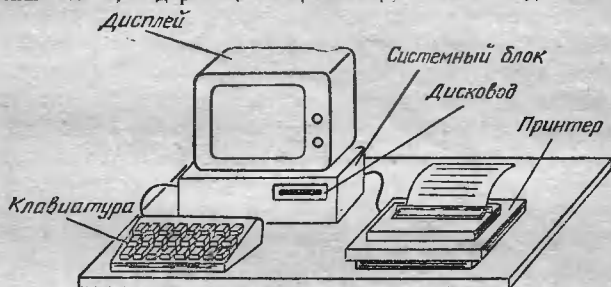


Рис. 1

рисунке видна прорезь для гибкого магнитного диска), дисплей, клавиатура и принтер \*).

Большой компьютер (рис. 2) используют одновременно несколько человек. В состав компьютера входит несколько терминалов (терминал — дисплей с клавиатурой, подключенный к компьютеру). Сам компьютер, как правило, находится в отдельном помещении. Терминалы же могут находиться в разных комнатах на рабочих местах сотрудников учреждения. Иногда к компьютеру подключаются терминалы, находящиеся в другом городе или даже на другом континенте. В этом случае используются каналы телефонной связи. В качестве магнитных носителей информации на больших компьютерах применяются массивные диски крупных размеров, а также магнитные ленты.

В дальнейшем изложении мы для определенности будем ориентироваться на персональный компьютер. Кроме того, на первых порах под устройством ввода будем подразумевать клавиатуру, под устройством вывода — дисплей.

**Программирование.** Раздел информатики, посвященный методам и приемам составления программ для компьютеров, называется

\*) Некоторые персональные компьютеры допускают подключение кассетного магнитофона.

программированием. Этот раздел очень важен, и изучение информатики обязательно должно включать в себя освоение программирования, так как в конечном счете именно программа позволяет применить компьютер для решения конкретной задачи.

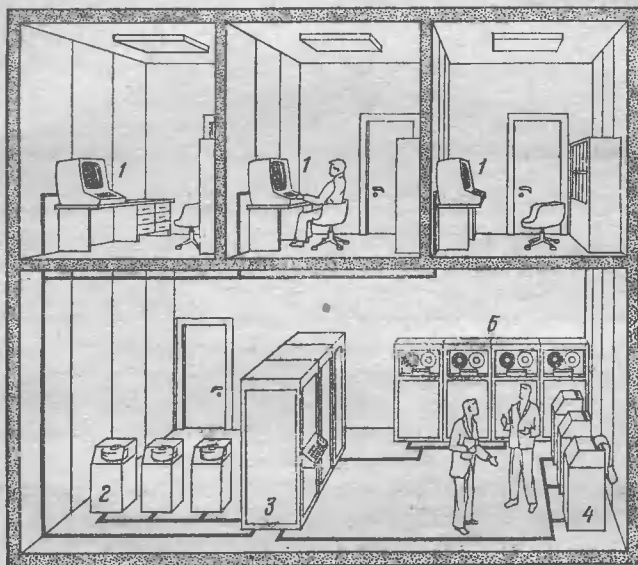


Рис. 2. Большой компьютер: 1—терминалы, 2—накопители на магнитных дисках, 3—центральный процессор, 4—устройства вывода информации, 5—накопители на магнитных лентах

В этой книге программированию уделяется первоочередное внимание (как уже говорилось, в качестве языка программирования выбран Паскаль). Попутно рассматриваются вопросы, относящиеся к ряду других разделов информатики.

## ГЛАВА I

# ПЕРВОЕ ЗНАКОМСТВО С ПРОГРАММИРОВАНИЕМ

### § 1. О записи программы. Выражения

Программа записывается в виде последовательности символов, к числу которых относятся латинские и русские буквы, арабские цифры, знаки препинания, знаки операций.

Для обозначения исходных данных и результатов вычислений употребляются *переменные*, которыми могут быть не только любые буквы —  $a, b, X, Y, ш, щ$  и т. д., но и последовательности символов вида  $x_1, x_2, time, Сила, alfa 100, ala2$  и т. д., которые состоят из букв и цифр и начинаются с буквы. Соответствующее исходное данное или результат вычисления называется значением переменной. Пока мы будем иметь дело с переменными, значениями которых являются числа. Числа в программе записываются в десятичной системе, вместо запятой пишется точка:  $0, -17, 0.26, 3.1415, +1.567, -0.18$  и т. д. Количество цифр в числе не может быть слишком большим; граница для этого количества в Паскале не оговаривается — она определяется характеристиками используемого компьютера. Это относится и к количеству букв и цифр в переменной.

Переменные и числа — простейшие частные случаи выражения. Более сложные выражения строятся из чисел и переменных с помощью знаков операций сложения, вычитания, умножения и деления. Эти знаки суть  $+, -, *, /$ . Кроме того, в выражении могут быть использованы круглые скобки и некоторые функции. Знак операции деления  $/$  позволяет записывать в строку выражения, которые традиционно записываются с выходом из строки: в Паскале пишут  $a/b, c/17, (a * x + b)/(c + d)$  и т. д. Знак операции умножения  $*$  нельзя опускать или заменять точкой. Допустимое для математического текста выражение  $0,5(x+7) \cdot (x+2) \cdot (x-3)$  в Паскале должно быть записано в виде  $0.5 * (x+7) * (x+2) * (x-3)$ . Знак — (минус) может употребляться и для изображения величины, противоположной данной:  $-x, -(a * b + y)$  и т. д. Нельзя размещать два знака операций рядом; последовательности символов

$3 * -2$ ,  $x1 / -x2$  — это не выражения, выражениями будут  $3 * (-2)$ ,  $x1 / (-x2)$ .

В выражении могут быть использованы следующие функции:

$abs(E)$  — модуль (абсолютная величина)  $E$ , т. е.  $|E|$ ;

$sqr(E)$  — квадрат (вторая степень)  $E$ , т. е.  $E^2$ ;

$sqrt(E)$  — квадратный корень из  $E$ , т. е.  $\sqrt{E}$ ;

$\left. \begin{matrix} \sin(E) \\ \cos(E) \end{matrix} \right\}$  — тригонометрические функции (аргументом служит радианная мера угла);

аргумент всегда заключается в скобки: мы пишем  $sqrt(sqr(b))$  —  $4 * a * c$  и т. д.

Функция  $sqr(E)$  не открывает дополнительных вычислительных возможностей, ее можно заменять произведением  $E$  на себя, но при громоздких  $E$  использование этой функции бывает удобным: проще написать  $sqr(x - delta + 0.17062)$ , чем  $(x - delta + 0.17062) * (x - delta + 0.17062)$ . Подчеркнем\*, что в Паскале нет операции возведения в произвольную степень  $n$ , поэтому, например,  $x^3$  записывают в виде  $x * x * x$  или  $sqr(x) * x$  и т. д.

При вычислении значений выражений действуют обычные правила старшинства операций: старшие операции — умножение и деление, следующие по старшинству — сложение и вычитание. Из двух операций одинакового старшинства первой выполняется та, знак которой в выражении встречается раньше. Круглые скобки изменяют этот естественный порядок: значением выражения  $(x + y) / 2$  будет половина суммы значений переменных  $x$  и  $y$ , в то же время значением выражения  $x + y / 2$  будет сумма значения  $x$  и половины значения  $y$ .

Последовательность букв и цифр, начинающаяся с буквы, называется *идентификатором*. Из рассмотренных примеров видно, что идентификатор не обязательно представляет собой переменную:  $\sin$ ,  $\cos$ ,  $abs$  и т. д. — это не переменные, а *имена функций* (их запрещается использовать в качестве переменных). В программах встречаются и другие виды идентификаторов.

## ЗАДАЧИ

1. Какие из следующих последовательностей символов\*) являются идентификаторами, а какие нет:

- а)  $x$ ; б)  $x_1$ ; в)  $x1$ ; г)  $x'$ ; д)  $x1x2$ ; е)  $ab$ ; ж)  $abcd$ ; з)  $\sin$ ;  
и)  $\sin x$ ; к)  $\sin(x)$ ; л)  $a = 1$ ; м)  $2a$ ;  
н) объем; о) об'ем; п)  $delta$ ; р)  $\max 15?$

---

\*) Точку или вопросительный знак в конце задачи и точку с запятой, а также запятую в конце отдельного пункта задачи следует понимать как знак препинания, который не включается в рассматриваемую последовательность символов.



2. Какие из следующих последовательностей символов являются числами в Паскале:

- а) 0; б)  $-5$ ; в)  $1/12$ ; г)  $3,14$ ; д)  $+7.7$ ; е)  $0.66\dots$ ; ж)  $0.(6)$ ;  
з)  $-0,815$ ; и)  $2+2.5$ ; к)  $\text{sqrt}(2)$ ; л)  $\pm 1$ ; м)  $\text{VIP}$

3. Какие числа и переменные содержатся в следующих выражениях, записанных по правилам Паскаля:

- а)  $2+x-y-1.7$ ; б)  $2*xy$ ; в)  $2+a-1/3$ ;  
г)  $1/2+1/3+1/4-0.2/0.5$

4. Записать по правилам Паскаля выражения:

а)  $\frac{x^2+y^2}{1-\frac{x^2-y^2}{2}}$ ; б)  $1+x+\frac{x^2}{2}$ ; в)  $1+|x|+|1+x|$ ;

г)  $\sqrt{1+\sqrt{|x|}}$ ; д)  $\frac{a+b}{c+d}-2.5$ ; е)  $\frac{a+b-1.7}{d}$ ;  
 $6+\frac{e+f+0.5}{5}$

ж)  $\frac{\sin a-b}{|b|+\cos b}+a$ ; з)  $\frac{1-\sqrt{1+|\sin|x||}}{2}$ ; и)  $\frac{1.2-9.8x}{1-y^2}$ .

5. Какие из следующих последовательностей символов являются выражениями, записанными по правилам Паскаля:

- а) 1; б)  $a$ ; в)  $ab$ ; г)  $1+|y|$ ; д)  $2xy$ ;  
е)  $-2*xy$ ; ж)  $xy2$ ; з)  $x^2+y^2$ ; и)  $-abs(x)+\sin y$ ;  
к)  $abs(x)+\cos(abs(y-1.7))$ ; л)  $x+y*-z$ ; м)  $1/-2+beta$

6. Переписать следующие выражения, записанные по правилам Паскаля, в традиционной математической форме:

- а)  $\text{sqrt}(a+b)-\text{sqrt}(a-b)$ ; б)  $a+b/(c+d)-(a+b)/c+d$ ;  
в)  $a*b/(c+d)-(c-d)/b*(a+b)$ ; г)  $1+\text{sqr}(\cos((x+y)/2))$ .

7. Вычислить значения выражений (с точностью до одной сотой):

- а)  $1/(2+1/4)$ ; б)  $1/(2+(1/4))$ ; в)  $\cos(0)$ ;  
г)  $\sin(1-0.5-1/2)+abs(1/(1/3-1))$ .

8. Дано выражение  $(x-1/2)*(y-3/10)=4/5$ .

а) Указать переменные и числа, содержащиеся в этом выражении.

б) Сколько операций требуется выполнить при вычислении значения этого выражения?

в) Заменить это выражение тождественно равным ему выражением, для вычисления значения которого требуется меньшее число операций.

9. Ниже приводятся выражения, записанные в традиционной математической форме, справа от которых даны выражения, записанные по правилам Паскаля. В каких случаях выражение, стоящее справа, является эквивалентом своего левого соседа:

а)  $\frac{b+\sqrt{b^2-4ac}}{2a}$   $(b+\text{sqrt}(\text{sqr}(b)-4*a*c))/2*a$ ;

б)  $\cos^2(x)$   $\text{sqrt}(\cos(x))$ ;

- в)  $\frac{a}{c} \cdot \frac{b}{d}$   $ab/cd$ ;  
 г)  $\frac{a}{c} \cdot \frac{b}{d}$   $((a/c) * b)/d$ ;  
 д)  $\sin x + \cos \frac{y}{2}$   $\sin(x) + \cos(y/2)$ ;  
 е)  $\frac{x+1}{y+1}$   $x+1/y+1$ ;  
 ж)  $\sin \frac{x+y}{2}$   $\sin(x+y/2)?$

10. Вычислить значения следующих функций при  $a=1$ :

- а)  $abs(a+1)$ ; б)  $\sqrt{a}$ ; в)  $\sqrt{a-3}$ ;  
 г)  $\sin(a-1)$ ; д)  $\cos(-2+2*a)$ .

11. Вычислить значения выражений (с точностью до одной сотой) при  $x=1$ ,  $y=-2$ ,  $a=2$ ,  $b=3$ :

- а)  $(x+y)/a*b$ ; б)  $(x+y)/(a+b)$ ; в)  $(x+y)/(a*b)$ ; г)  $b$ ;  
 д)  $\sin(\sin(\sqrt{x}-1)) + \cos(x*x*x-1) * \cos(abs(x-2)-1)/y*a + \sqrt{abs(y)-x}$ .

12. Ниже приводятся выражения, записанные в традиционной математической форме, справа от которых размещены некоторые выражения, записанные по правилам Паскаля. Расстановкой скобок в каждом из выражений, стоящих справа, добиться того, чтобы выражение стало эквивалентом своего левого соседа:

- а)  $\frac{a}{b \cdot \frac{c}{d \cdot \frac{e}{f \cdot h}}}$   $a/b * c/d * e/f * h$ ;  
 б)  $\frac{a+b}{x-2y}$   $a+b/x-2*y$ ;  
 в)  $a + \frac{b}{x-2} * y$   $a + b/x-2*y$ .

13. Удалить лишние скобки (ошибки в использовании лишних скобок нет, но они загромождают выражения):

- а)  $(x1/x2) * y$ ; б)  $(\sqrt{p} * q)/r$ ;  
 в)  $b + (a - (c/3))$ ; г)  $(a * (b/(c * (d/(e * f))))))$ .

14. Написать по правилам Паскаля выражение, значением которого является:

- а) периметр квадрата, площадь которого равна  $a$ ;  
 б) площадь равностороннего треугольника, периметр которого равен  $p$ ;  
 в) работа силы тяжести при перемещении тела массы  $m$  на расстояние  $s$ . Угол между вектором силы тяжести и направлением перемещения равен  $\alpha$   $fa$ .

15. Написать по правилам Паскаля несколько вариантов выражения, значение которого равно  $x^4$ . Для каждого из вариантов под-

считать количество умножений, требующихся при вычислении значения выражения (вычисление значения функции *sqr* требует одного умножения).

## § 2. Операторы присваивания, ввода и вывода

Главной частью программы является последовательность инструкций, которую должен выполнить компьютер. Инструкции, входящие в программу на Паскале, принято называть *операторами*.

В результате выполнения оператора присваивания переменной присваивается значение некоторого выражения. Примеры операторов присваивания:

```
a:=0
b:=c
x1:=(-b+sqrt(sqr(b)-4*a*c))/(2*a)
x:=x+1
```

Во всех случаях вначале вычисляется значение выражения, расположенного справа от комбинации символов  $:=$ , а затем вычисленное значение присваивается переменной, расположенной слева. Для того чтобы оператор присваивания мог быть выполнен, необходимо, чтобы все переменные, которые входят в выражение, имели некоторые значения. Последний из приведенных выше операторов присваивания предписывает увеличение значения  $x$  на единицу. Если до выполнения оператора  $x:=x+1$  переменная  $x$  имела значение 1.3, то после выполнения этого оператора значение  $x$  станет равным 2.3. Из этого примера видно, что в ходе выполнения программы переменные могут изменять свои значения.

Для ввода данных и вывода результатов используются операторы ввода и вывода. Они могут выглядеть, например, так:

```
read(a)
read(x1, x2, y)
write(x)
write(x+y, x-y)
write(x, 2*x-a, sqrt(x), y)
```

Оператор ввода состоит из идентификатора *read* (*read* по-английски означает читать) и следующего за ним в круглых скобках списка переменных. Число переменных в списке может быть любым: если переменных больше одной, то они разделяются запятыми. Оператор вывода состоит из идентификатора *write* (*write* означает писать) и следующего за ним в круглых скобках списка выражений.

При выполнении оператора ввода переменным присваиваются значения исходных данных. Те числа, которые являются исходными данными, надо своевременно набрать на клавиатуре компьютера.

Пусть на клавиатуре набрано число 3.6, тогда в результате выполнения оператора *read(a)* переменная *a* получит значение 3.6, после чего начнет выполняться следующий оператор программы.

Аналогично оператор *read(x1, x2, y)* сможет выполняться, когда на клавиатуре будут набраны три числа, разделенных запятыми, и т. д. Привлечение оператора ввода открывает возможность многократного использования одной и той же программы для вычислений с различными исходными данными.

Оператор вывода позволяет выделить из всего набора вычисленных величин те, которые служат ответом к решавшейся при выполнении программы задаче. Пусть был выполнен оператор *write(x)* и значение переменной *x* в момент выполнения равнялось 15.6. Пусть через некоторое время был выполнен оператор *write(z, t)* и в момент выполнения этого оператора значение *z* было равно  $-3.57$ , а значение *t* было равно  $-0.00017$ . Если больше никаких операторов не выполнялось, то итогом выполнения всей программы будет последовательность, состоящая из чисел 15.6,  $-3.57$  и  $-0.00017$ . Эта последовательность будет выведена компьютером на экран дисплея.

В результате выполнения оператора *write(x+y, x-y)* на экран будут выведены значения выражений  $x+y$  и  $x-y$ . Если в момент выполнения этого оператора значением переменной *x* было число 1.25, а значением переменной *y* — число 0.05, то на экране появится два числа: 1.3 и 1.2\*).

Следующие два оператора предписывают ввод трех чисел и вывод их суммы. Друг от друга операторы в программе отделяются точкой с запятой:

*read(a, b, c); write(a+b+c)*

Если надо ввести два данных значения *x* и *y* и вывести значения  $x(x+y)$  и  $y(x+y)$ , то можно воспользоваться операторами

*read(x, y); write(x\*(x+y), y\*(x+y))*

но с помощью оператора присваивания можно избежать повторного вычисления значения  $x+y$ :

*read(x, y); z:=x+y; write(x\*z, y\*z)*

(многократные вычисления одних и тех же значений нежелательны из-за замедления выполнения программы).

---

\*) Возможно, что числа будут выведены в так называемом нормализованном виде. Число 1.3 будет выглядеть так:  $+0.130000E+01$ , здесь  $+0.130000$  — мантисса, а  $+01$  — порядок данного числа. Запись  $+0.130000E+01$  расшифровывается как  $(+0.130000)10^{+1}$ , что эквивалентно 1.3. Число 0 будет записано в виде  $+0.000000E+00$ . Если *m* — мантисса ненулевого числа, то  $0.1 \leq |m| \leq 1$ . Знак мантиссы совпадает со знаком числа. Порядок — целое число. Количество цифр в мантиссе зависит от используемого компьютера.

Следующая последовательность операторов предписывает ввод длины окружности, а также вычисление и вывод радиуса этой окружности и площади круга, ограниченного этой окружностью:

$read(l); r:=l/(2 * 3.14); write(r, 3.14 * sqr(r))$

Строго говоря, переменная  $r$  для такой последовательности операторов не нужна, можно было бы написать

$read(l); l:=l/6.28; write(l, 3.14 * sqr(l)).$

Последовательность операторов — это еще не программа на паскале. В программу входят еще некоторые компоненты, о которых речь пойдет в следующем параграфе.

### ЗАДАЧИ

16. Какие из следующих последовательностей символов являются операторами присваивания:

- а)  $a:=b$ ; б)  $a=c+1$ ; в)  $a:b=sqr(2)$ ;  
г)  $a * x + b:=0$ ; д)  $z:=0$ ; е)  $z:=z+1$ ;  
ж)  $z:=z+1,2$ ; з)  $y:=y$ ; и)  $-y:=y^2$

17. Пусть значения переменных  $x$  и  $y$  равны, соответственно, 0.3 и -0.2. Какие значения будут иметь эти переменные после выполнения операторов присваивания:

- а)  $x:=x+2 * y$ ;  $y:=y/2$ ;  
б)  $y:=-y$ ;  $x:=x+y$ ;  $y:=y+1$ ;  
в)  $x:=1$ ; г)  $y:=x+y^2$

18. Задать в виде оператора присваивания следующие действия:

а) переменной  $z$  присвоить значение, равное полусумме значений переменных  $x$  и  $y$ ;

б) удвоить значение переменной  $a$ ;

в) значение переменной  $x$  увеличить на 0.1;

г) изменить знак значения переменной  $t$ .

19. Задать с помощью операторов присваивания следующие действия:

а) переменной  $a$  присвоить значение разности, а переменной  $b$  — полусуммы значений переменных  $x$  и  $y$ ;

б) переменной  $a$  присвоить значение удвоенного произведения значений переменных  $x$  и  $y$ , а переменной  $b$  — значение 0.

20. Написать оператор присваивания, в результате выполнения которого переменная  $y$  получает значение, равное значению переменной  $x$ , возведенному в пятую степень.

21. Какие из следующих последовательностей символов являются операторами ввода:

- а)  $read(x, y, z)$ ; б)  $read\ x, y, z$ ; в)  $read(x)$ ;  
г)  $x:=read(x)$ ; д)  $read(a; b)$ ; е)  $read(a, b+c)$ ?

22. Какие значения будут иметь переменные  $x$  и  $y$  в результате выполнения последовательности операторов

$read(x, y); t:=x; x:=y; y:=t,$

если последовательность исходных данных была составлена из чисел 5.2 и 18.7?

23. Какие из следующих последовательностей символов являются операторами вывода:

- а)  $write(x, y);$  б)  $write(x, x+1, x+2);$  в)  $read(a);$   
г)  $write(100);$  д)  $print(y, z);$  е)  $print y, z;$   
ж)  $write(x; y; z);$  з)  $write(x+2.2);$  и)  $write(x+2,2)?$

24. Какие числа будут выведены в результате выполнения последовательности операторов:

$x:=(sin(sqr(1)-1)+2*abs(-2))/cos(2-2);$   
 $y:=x*(sqr(2)); write(x, y)?$

25. Какие числа будут выведены в результате выполнения последовательности операторов

$read(x); x:=x-1.2;$   
 $x:=sqr(x-1)*x+1; write(x, 2-3*x)?$

если в качестве исходного данного использовалось число:

- а) 1.2, б) 2.2, в) 5.2, г) 10.2?

26. Всегда ли выполнение последовательностей операторов

$read(x, y); x:=x*x; write(x, y)$

и

$read(x, y); write(sqr(x)); write(y)$

при одних и тех же исходных данных приводит к выводу одних и тех же чисел? Тот же вопрос для последовательностей операторов

$read(x, y); z:=sqr(x)+1; write(x/z, y/z, x)$

и

$read(x, y); write(x/sqr(x)+1, y/sqr(x)+1, x)$

27. Две последовательности операторов

$u:=u+v; v:=2*v$

и

$v:=2*v; u:=u+v$

отличаются только порядком следования операторов. Верно ли, что для любых начальных значений  $u$  и  $v$  выполнение этих последовательностей операторов приводит к одинаковому изменению значений переменных  $u$  и  $v$ ?

28. Написать последовательность операторов, предназначенную для ввода двух данных чисел и последующего вычисления и вывода их среднего арифметического и среднего геометрического (предполагается, что данные числа неотрицательны).

29. Написать последовательность операторов, предназначенную для ввода значения  $v$  и последующего вычисления и вывода величины

$$\frac{120 + 12v^2 + v(60 + v^2)}{120 + 12v^2 - v(60 + v^2)}.$$

Позаботиться об экономии операций.

30. Какие числа будут выведены в результате выполнения последовательности операторов

*read(x, y); x := x + y; y := x - y; x := x - y; write(x, y)?*

если последовательность исходных данных была составлена из двух чисел:

а) 3.5 и 2.4; б) 6.7 и -10.1?

### § 3. Простейшая программа

Во всех рассмотренных примерах переменные принимают значения в множестве действительных чисел, такие переменные называются переменными типа *real* (*real* — по-английски действительный).

Кроме операторов, программа содержит описания переменных. Примеры описаний переменных:

*x: real*

*y, z, t: real*

После списка переменных идет двоеточие и идентификатор *real*. Каждая переменная должна быть описана, т. е. включена в описание. Все описания переменных соединяются вместе в совокупность описаний переменных: *var A<sub>1</sub>; ...; A<sub>k</sub>*; т. е. вслед за идентификатором *var* идет некоторое количество описаний переменных, после каждого из которых ставится точка с запятой. Пока в наших программах будет преимущественно по одному описанию. Мы будем писать

*var y: real;*

или

*var x1, x2, t: real;*

и т. д.

Идентификаторы *var* и *real* относятся к так называемым служебным словам, значение которых жестко зафиксировано в Паскале; эти служебные слова нельзя, например, использовать в ка-

честве переменных. Уже встречавшиеся идентификаторы *abs*, *sqr*, *sqrt*, *sin*, *cos*, *read*, *write* также являются служебными словами. Сочетание букв *var* является сокращением английского слова *variable*—переменная (далее переводы служебных слов будут даваться в подстрочных примечаниях без указания на то, что слова английские).

В дальнейшем мы рассмотрим еще несколько видов операторов и описаний. Но и с помощью уже рассмотренных операторов и описаний можно составить ряд программ \*).

Простейшая программа на Паскале схематически может быть изображена следующим образом:

```
program N (...);
  var A1; ...; Ak;
  begin P1; ...; Pn end. **)
```

Здесь *N*—имя программы (идентификатор), *A<sub>1</sub>, ..., A<sub>k</sub>*—описания переменных, *P<sub>1</sub>, ..., P<sub>n</sub>*—операторы. Вместо многоточия внутри скобок надо поместить либо идентификатор *input*, либо *output*, либо оба эти идентификатора через запятую: *input, output* \*\*\*). Идентификатор *input* означает, что в программе встретится оператор ввода, *output*—оператор вывода.

Операторы *P<sub>1</sub>, ..., P<sub>n</sub>* выполняются в порядке следования друг за другом.

Примеры. Программа *корни1* вычисления корней квадратного уравнения  $ax^2 + bx + c = 0$ , заданного коэффициентами *a*, *b* и *c* (предполагается, что  $a \neq 0$  и что дискриминант уравнения неотрицателен) \*\*\*\*):

```
program корни1(input, output);
  var a, b, c: real;
  begin read (a, b, c);
    write((-b + sqrt(sqr(b) - 4*a*c))/(2*a),
          (-b - sqrt(sqr(b) - 4*a*c))/(2*a))
  end.
```

---

\*) Отметим одно обстоятельство, существенное для практической работы на компьютере. Если мы пользуемся при написании программ лишь рассмотренными простейшими средствами, то исходные данные, т. е. числа, подготовленные для ввода, обязательно должны содержать десятичную точку: 15.02, -2.3, -127.0, 1.0, 0.0 и т. д. Мы вернемся к этому вопросу в § 8.

\*\*) Program—программа, begin—начало, end—конец.

\*\*\*) Input—ввод, output—вывод.

\*\*\*\*) Программа—это последовательность символов. При записи программы мы из соображений удобства разбиваем ее на строки. Это разбиение является условным, никакие знаки переноса не используются.



При выполнении этой программы придется дважды вычислить значение  $\sqrt{\sqrt{b}-4ac}$  и дважды — значение  $2a$ . Поэтому более разумным вариантом будет программа

```
program корни2 (input, output);  
  var a, b, c, d, e: real;  
  begin read(a, b, c);  
        d:=sqrt(sqrt(b)-4*a*c); e:=2*a;  
        write((-b+d)/e, (-b-d)/e)  
  end.
```

Если это почему-либо удобно, можно одно длинное описание заменить совокупностью более коротких. В программе *корни2* можно было бы написать, например,

```
var a, b, c: real; d, e: real;
```

важно лишь, чтобы каждая переменная была описана и входила только в одно описание.

Пример. Программа вычисления площади треугольника по трем сторонам  $a_1, a_2, a_3$ :

```
program площадь (input, output);  
  var a1, a2, a3, p: real;  
  begin read(a1, a2, a3);  
        p:=(a1+a2+a3)/2;  
        write(sqrt(p*(p-a1)*(p-a2)*(p-a3)))  
  end.
```

Площадь вычисляется по формуле Герона.

Начальная часть программы, т. е.

```
program N(...);
```

называется *заголовком*. Обратим внимание на то, что между служебным словом *program* и именем программы ставится пробел. Пробел является символом (печатным знаком). На клавиатуре компьютера есть специальная клавиша пробела. Пробелы в программе нужно обязательно ставить между всеми теми идентификаторами и числами, которые не разделены другими знаками. В нашем последнем примере программы пробелы присутствуют между идентификаторами *program* и *площадь*, между *var* и *a1*, между *begin* и *read*.

Как уже говорилось, при записи программы мы из соображений удобства разбиваем ее на строки. Строку можно начинать с нескольких пробелов, добиваясь этим, например, того, что некоторые служебные слова будут располагаться одно под другим. Аналогично и некоторые операторы могут быть записаны столбиком. Это может облегчить чтение программы.

Еще одно замечание, касающееся уже выполнения программ. При выполнении программы вычисления могут проводиться прибли-

женно: числа в Паскале имеют конечное число знаков после десятичной точки, и округления неизбежно возникают, например, при вычислении значений  $\sqrt{2}$ ,  $\sin(1)$  и т. д.

Программа вводится в память компьютера с клавиатуры. Для того чтобы программа выполнялась, надо дополнительно ввести специальный текст, служащий для компьютера сигналом к началу выполнения программы. Довольно часто этим текстом служит служебное слово *RUN* \*), но, конечно, этот текст может быть и другим — многое зависит от того, какой именно компьютер используется.

Выполнение программы начинается с некоторого подготовительного этапа работы процессора. В течение этого этапа, в частности, выясняется, имеются ли в программе так называемые синтаксические ошибки, т. е. отступления от правил записи программы на языке Паскаль. В случае когда синтаксические ошибки отсутствуют, рассматриваются содержащиеся перед операторами описания переменных и отводится место в памяти для размещения значений каждой из переменных. Всякий раз, когда позднее во время выполнения программы некоторая переменная будет получать значение, это значение будет помещаться в закрепленное за данной переменной место в памяти, а предыдущее значение этой переменной будет стираться. Если же переменная встретится в выражении, то при вычислении его значения произойдет обращение за значением этой переменной в отведенное ей место памяти.

Если ошибок в программе нет, то она выполнится до конца и результаты можно будет увидеть на экране. Рассмотрим теперь ситуацию, когда в программе имеются ошибки. Это довольно типичная ситуация, и поэтому всегда, на любом компьютере, предусматривается возможность исправления ошибок в программах. Работа по исправлению ошибок, внесению изменений и т. д. называется *редактированием* программы. Над записанным в виде последовательности строк текстом программы можно выполнять такие действия, как удаление, замена или вставка символа внутри строки, удаление, замена или вставка целой строки или группы строк, перестановка строк и т. д.

Отсутствие синтаксических ошибок в программе вовсе не означает, что программа верна. Так, если в рассмотренной в этом параграфе программе *площадь* заменить идентификатор  $\sqrt{2}$  на  $\sqrt{q}$ , то программа останется синтаксически правильной (по-прежнему будет удовлетворять правилам записи программ на Паскале).

Однако выполнение этой программы будет приводить к неверному результату. Обнаружить подобную ошибку значительно сложнее, чем синтаксическую (синтаксические ошибки, как уже сказано,

---

\*) Run — бежать.

обнаруживаются на подготовительном этапе работы процессора, предшествующем непосредственному выполнению программы).

Программа, перед тем как она будет применена для решения поставленной задачи, должна быть отлажена. Цель отладки состоит в том, чтобы выявить и устранить допущенные ошибки. Одним из способов отладки является тестирование, суть которого состоит в следующем. Программа выполняется несколько раз с различными исходными данными. При этом в качестве исходных данных вводятся такие значения, для которых заранее известны результаты (такие наборы исходных данных называются тестами). Результаты, полученные при выполнении программы, сравниваются с известными результатами. Несовпадение их между собой свидетельствует о наличии ошибок в программе. Помочь обнаружить место ошибки в программе может вывод промежуточных результатов.

Рассмотрим в качестве примера программу *площадь*, в которой идентификатор *sqr* заменен на *sqr*. Выполним эту программу при исходных данных, равных 3, 4, 5 (результатом выполнения программы должно быть число, равное площади треугольника со сторонами 3, 4, 5). Полученный при выполнении программы результат — число 1296 не совпадает с известным заранее — числом 6\*). Вставим в программу оператор вывода значения переменной *p* — *write(p)* и выполним ее еще раз с теми же исходными данными. Результатом выполнения программы в этом случае будут служить два числа 6 и 1296 (полупериметр треугольника и его площадь). Видно, что полупериметр вычислен верно. Значит, ошибка связана с видом выражения

$$sqr(p*(p-a1)*(p-a2)*(p-a3)).$$

Если программа предназначена для многократного использования, то ее можно переписать из памяти компьютера на магнитный диск или магнитную ленту. Впоследствии эту программу можно будет вводить уже не с клавиатуры, а с диска или ленты. Такой ввод происходит очень быстро.

## ЗАДАЧИ

31. Даны  $x, y$ . Получить \*\*)  $\frac{|x| - |y|}{1 + |xy|}$ .

32. Даны два числа. Найти среднее арифметическое кубов этих чисел и среднее геометрическое модулей этих чисел.

\*) В соответствии с подстрочным примечанием на с. 23, исходные данные должны иметь вид 3.0, 4.0, 5.0. Результат должен быть приближенно (но с большой степенью точности) равен 6.

\*\*) Здесь и далее в формулировках задач для краткости будем писать «получить» вместо «написать программу, позволяющую получить»; то же самое касается слов «вычислить», «найти» и т. д.

33. Вычислить расстояние между двумя точками с данными координатами  $x_1, y_1$  и  $x_2, y_2$ .

34. По двум данным катетам найти гипотенузу и площадь прямоугольного треугольника.

35. Даны числа  $x, y$ . Вычислить их сумму, разность и произведение.

36. Даны  $a, b, \gamma$ . Найти площадь треугольника, две стороны которого равны  $a$  и  $b$ , а угол между этими сторонами равен  $\gamma$ . Считать, что  $\gamma$  —

а) радианная мера угла;

б) градусная мера угла.

37. Дано  $\varphi$ . Найти площадь сектора, радиус которого равен 13.75, а соответствующий центральный угол равен  $\varphi$ . Считать, что  $\varphi$  —

а) радианная мера угла;

б) градусная мера угла.

38. Даны  $v_1, v_2, t_1, t_2$ . Пусть смешано  $v_1$  литров воды температуры  $t_1$  с  $v_2$  литрами воды температуры  $t_2$ . Вычислить объем и температуру образовавшейся смеси.

39. Дана длина ребра куба. Найти площадь грани, площадь полной поверхности и объем этого куба.

40. Треугольник задан координатами  $x_1, y_1, x_2, y_2, x_3, y_3$  своих вершин:

а) найти периметр треугольника;

б) найти площадь треугольника.

41. Даны  $x, y, z$ . Вычислить  $u, v$ , если:

$$а) u = \sin \left| (y - \sqrt{|x|}) \left( x - \frac{y}{z^2 + \frac{x^2}{4}} \right) \right|, \quad v = \cos \left( z^2 + \frac{x^2}{4} \right);$$

$$б) u = \frac{1 + \sin^2(x+y)}{2 + \left| x - \frac{2x}{1 + |\sin(x+y)|} \right|}, \quad v = x - \frac{x^2}{1 + \sin^2(x+y)}.$$

42. Даны  $c, d$ . Вычислить

$$\frac{\sin^3 |cx_1^3 + dx_2^3 - cd|}{\sqrt{(cx_1^3 + dx_2^3 - x_1)^2 + 3.14}} + \operatorname{tg}(cx_1^3 + dx_2^3 - x_1),$$

где  $x_1$  — больший, а  $x_2$  — меньший корень уравнения  $x^2 - 3x - |cd| = 0$ .

43. Дано  $x$ . Получить значения  $1 - 2x + 3x^2 - 4x^3$  и  $1 + 2x + 3x^2 + 4x^3$ . Позаботиться об экономии операций.

44. Дано  $x$ . Вычислить  $2x^4 - 3x^3 + 4x^2 - 5x + 6$ . Позаботиться об экономии операций.

45. Дано  $a$ . Не используя никаких функций и никаких операций кроме умножения, получить:

а)  $a^8$  за три операции;

- б)  $a^{10}$  за четыре операции;
- в)  $a^7$  за четыре операции;
- г)  $a^{15}$  за пять операций (указание:  $a^{15} = (a^5)^3$ ).

## § 4. Условный оператор

Если мы хотим, чтобы переменной *max* присвоилось наибольшее из значений переменных  $x_1$  и  $x_2$ , то надо сравнить значения  $x_1$  и  $x_2$  и в зависимости от результата сравнения выполнить либо оператор  $\text{max} := x_1$ , либо  $\text{max} := x_2$ . Действия такого рода задаются условным оператором:

*if B then  $P_1$  else  $P_2$  \*),*

где  $B$  — условие,  $P_1$  и  $P_2$  — операторы. Если  $B$  соблюдается, то выполняется  $P_1$ , иначе выполняется  $P_2$ .

В качестве условий используются отношения. Отношения представляют собой записи равенств и неравенств. Примеры отношений:

$$a = b$$

$$x < a - b$$

$$\text{sqr}(b) - 4 * a * c > 0.$$

Для решения задачи о присваивании переменной *max* наибольшего из значений  $x_1$  и  $x_2$  достаточно выполнить условный оператор

*if  $x_1 > x_2$  then  $\text{max} := x_1$  else  $\text{max} := x_2$*

На клавиатурах современных компьютеров нет знаков  $\neq$ ,  $\geq$ ,  $\leq$ , и вместо этих знаков используются соответственно комбинации  $< >$ ,  $> =$ ,  $< =$ . Вместо  $c + d \neq e + f$  получается  $c + d < > e + f$ , вместо  $x \geq y$  и  $a - b \leq 0$  — соответственно  $x > = y$ ,  $a - b < = 0$ . В общем случае отношение — это два выражения, разделенных одним из знаков  $=$ ,  $>$ ,  $<$  или же одной из комбинаций знаков  $< >$ ,  $> =$ ,  $< =$ . В программах могут, например, встретиться условные операторы

*if  $a < = b$  then  $a := a - 1$  else  $b := b - 1$*

*if  $a < > b$  then  $a := a + b$  else  $a := a - b$*

и т. д.

Рассмотрим примеры программ, включающих в себя условные операторы. Пусть задана функция

$$y = \begin{cases} 0, & \text{если } x \leq 0; \\ x^3, & \text{если } x > 0. \end{cases}$$

Требуется написать программу вычисления значения  $y$  по значению  $x$ . Программа может выглядеть так:

---

\*) If — если, then — то, else — иначе.

```

program A(input, output);
  var x, y: real;
  begin read(x);
    if  $x \leq 0$  then  $y := 0$  else  $y := x * x * x$ ;
    write(y)
  end.

```

Пример. Пусть значение  $y$  зависит от значения  $x$ , график зависимости приведен на рис. 3. Программа вычисления значения  $y$  по значению  $x$ :

```

program Б(input, output);

```

```

  var x, y: real;

```

```

  begin read(x);

```

```

    if  $x < 2$  then  $y := x$  else

```

```

      if  $x < 3$  then  $y := 2$  else  $y := -x + 5$ ;

```

```

    write(y)

```

```

  end.

```

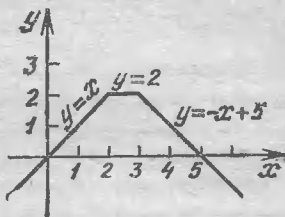


Рис. 3

Если соблюдается условие  $x < 2$ , то  $y$  получит значение, равное значению  $x$ , и это значение затем будет выведено. Если условие  $x < 2$  не соблюдается, то значение  $y$  будет определено выполнением условного оператора

```

if  $x < 3$  then  $y := 2$  else  $y := -x + 5$ 

```

Допускается сокращенная форма условного оператора:

```

if B then P

```

где  $B$  — условие,  $P$  — оператор. В случае если  $B$  соблюдается, должен быть выполнен оператор  $P$ ; если же  $B$  не соблюдается, его выполнять не нужно.

Пример. Пусть даны два числа. Если первое больше второго по абсолютной величине, то необходимо уменьшить первое в пять раз. Иначе оставить числа без изменения. Программа:

```

program B(input, output);

```

```

  var x, y: real;

```

```

  begin read(x, y);

```

```

    if  $\text{abs}(x) > \text{abs}(y)$  then  $x := x/5$ ;

```

```

    write(x, y)

```

```

  end.

```

Пример. Даны три числа  $x, y, z$ . Требуется найти  $\max(x, y, z)$  — наибольшее из этих чисел (среди  $x, y, z$  могут быть и равные числа: например,  $\max(3, -1, 3) = 3$ ,  $\max(7, 7, 7) = 7$  и т. д.). Программа:

```

program M (input, output);
  var x, y, z, t: real;
  begin read (x, y, z); t:=x;
        if t < y then t:=y;
        if t < z then t:=z; write(t)
  end.

```

После выполнения первого условного оператора переменная  $t$  имеет значение наибольшего из первых двух данных чисел, после выполнения второго условного оператора — значение наибольшего из трех данных чисел.

Зафиксируем следующее ограничение. Оператор, который располагается непосредственно после служебного слова *then* не может быть условным (в то время как оператор, располагающийся после *else* может быть любым, и, в частности, условным — см. программу Б этого параграфа). Без этого ограничения пришлось бы рассматривать операторы вроде следующего:

```
if a > 0 then if a < 2 then a:=2 else a:=3
```

и было бы неясно, к какому из двух *if* относится единственное *else*. При использовании условного оператора после *else* никаких двусмысленностей не возникает.

## ЗАДАЧИ

46. Является ли условным оператором последовательность символов:

- а) *if x < y then x:=0 else y:=0*,
- б) *if x > y then x:=0 else read (y)*,
- в) *if x >= y then x:=0; y:=0 else write (z)*,
- г) *if a < b then 100 else z:=5*,
- д) *if a < b < c then z:=z+1*,
- е) *if sqrt(z) <= 3.17 then z:=z+1*,
- ж) *if a < > b then z:=z+1; a:=b+1*?

47. Ответить на следующие вопросы.

а) Какой оператор не разрешается использовать вслед за служебным словом *then*?

б) Каковы формы условного оператора? В чем их различие?

48. Какие из следующих вложенных условных операторов допустимы:

- а) *if x+y < z then x:=x+1 else if y > z then z:=0 else y:=0*,
- б) *if x+y < z then if y > z then z:=0 else y:=0 else z:=0*,
- в) *if x+y < z then if y > z then z:=0 else y:=0*?

49. Даны  $x, y$  ( $x \neq y$ ). Получить числа  $x, y$  в следующем порядке: вначале большее, потом меньшее.

50. Даны два числа. Вывести первое число, если оно больше второго, и оба числа, если это не так.

51. Даны  $x, y, z$ . Найти

а)  $\max(x+y+z, xyz)+3$ ,

б)  $\min(x^2+y^2, y^2+z^2)-4$ . \*)

52. Дано  $x$ . Выяснить, верно ли, что  $0 \leq x < 1$ . Если верно, то должно быть выведено число 1, в противном случае — число 0.

53. Дано  $x$ . Вычислить  $y$ , если:

а)  $y = \begin{cases} x^2 & \text{при } -2 \leq x \leq 2, \\ 4 & \text{в противном случае;} \end{cases}$

б)  $y = \begin{cases} x^2+4x+5 & \text{при } x \leq 2, \\ \frac{1}{x^2+4x+5} & \text{в противном случае;} \end{cases}$

в)  $y = \begin{cases} 0 & \text{при } x \leq 0, \\ x & \text{при } 0 < x \leq 1, \\ x^4 & \text{в противном случае;} \end{cases}$

г)  $y = \begin{cases} 0 & \text{при } x \leq 0, \\ x^2-x & \text{при } 0 < x \leq 1, \\ x^2-\sin \pi x^2-1 & \text{в противном случае.} \end{cases}$

54. Дано действительное  $a$ . Для функций  $y=f(x)$ , графики которых представлены на рис. 4 а—г, вычислить  $f(a)$ .

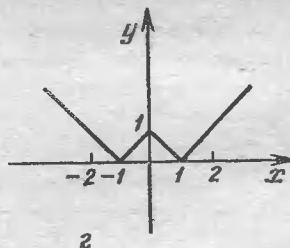
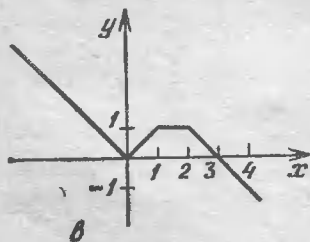
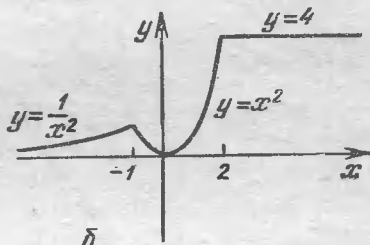
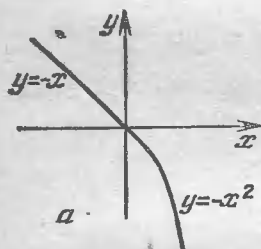


Рис. 4

\*)  $\min(a, b, c, \dots)$  — наименьшее из чисел  $a, b, c, \dots$ . Среди  $a, b, c, \dots$  могут быть и равные числа: например,  $\min(2, 2, 8) = 2$ ,  $\min(-1, -1) = -1$  и т. д.



55. По условному оператору, устанавливающему зависимость значения переменной  $y$  от значения переменной  $x$ , построить график этой зависимости. В качестве оператора рассмотреть:

- а) if  $x < 1$  then  $y := x * x * x$  else  $y := 2 - x$ ,
- б) if  $x < 1$  then  $y := \text{sqr}(x + 2)$  else if  $x \leq 3$  then  $y := 2 * x + 3$  else  $y := 7.5$ ,
- в) if  $x > 2$  then  $y := x - 4$  else if  $x > -1$  then  $y := -x$  else  $y := -1/x$ ,
- г) if  $\text{abs}(x) > 10$  then  $y := x$  else if  $x > 0$  then  $y := 2 * x - 10$  else  $y := 10$ .

56. Даны переменные  $x, y$ . Выяснить, принадлежит ли точка с координатами  $(x, y)$ :

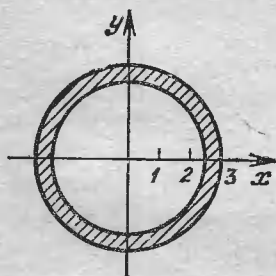


Рис. 5

а) кругу единичного радиуса с центром в начале координат;

б) кольцу с центром в начале координат с внешним радиусом 3 и с внутренним радиусом 2.5 (рис. 5).

57. Даны два числа. Заменить второе число нулем, если оно не меньше первого, и оставить его прежним, если это не так. Первое число оставить без изменения.

58. Убедиться, что программу  $M$  из настоящего параграфа (поиск наибольшего из трех чисел  $x, y, z$ ) можно было бы написать без привлечения дополнительной переменной  $t$ :

```

program M1(input, output);
  var x, y, z: real;
  begin read(x, y, z);
        if x < y then x := y;
        if x < z then x := z;
        write(x)
  end.

```

Убедиться также в том, что программа может быть написана с использованием всего двух переменных  $x$  и  $y$ . Написать такую программу.

59. Найти наименьшее из трех данных чисел.

60. Даны три числа. Возвести в квадрат те из них, значения которых неотрицательны. Отрицательные числа оставить без изменения.

## § 5. Составной оператор.

### Дополнительные возможности вывода информации

В условном операторе после *then* и *else* можно помещать по одному оператору. Однако часто необходимо в зависимости от результата проверки некоторого условия выполнить ту или иную группу операторов. Паскаль предоставляет возможность сделать из группы операторов один *составной оператор*. Структура составного оператора:

```
begin P1; P2; ...; Pk end
```

где  $P_1, P_2, \dots, P_k$  — любые операторы. Как частный случай составного оператора имеем *begin P end*, где  $P$  — любой оператор. Существенно, что оператор *begin P end* при любом операторе  $P$  не является условным и может быть размещен после *then*.

Пусть квадратное уравнение  $ax^2 + bx + c = 0$  задано коэффициентами  $a, b$  и  $c$  ( $a \neq 0$ ).

Ниже следует программа, при выполнении которой исследуется дискриминант уравнения, выводится число 0, если дискриминант отрицателен, и выводится пара корней, если дискриминант неотрицателен:

```
program корни3(input, output);  
  var a, b, c, d: real;  
  begin read(a, b, c);  
        d := sqr(b) - 4*a*c;  
        if d < 0 then write(0)  
          else begin d := sqrt(d);  
                   a := 2*a;  
                   write((-b+d)/a, (-b-d)/a)  
                 end  
  end.
```

В условном операторе этой программы после *else* помещен составной оператор.

Программа Б, рассмотренная в предыдущем параграфе, может быть переписана так:

```
program Б1(input, output);  
  var x, y: real;  
  begin read(x);  
        if x < 3 then begin  
          if x >= 2 then y := 2 else y := x  
        end  
        else y := -x + 5;  
        write(y)  
  end.
```

Если  $x < 3$ , то выполняется оператор

*begin if  $x \geq 2$  then  $y := 2$  else  $y := x$  end*

который оказалось возможным разместить после *then* благодаря тому, что он является составным оператором и не является условным.

Составной оператор может использоваться не только внутри условного оператора, но и всюду в программе, наравне с ранее рассмотренными операторами.

Вернемся к программе *корни3* этого параграфа. Если уравнение не имеет корней, то выполнение программы *корни3* завершается выводом числа 0. Есть, однако, возможность выдать сообщение об отсутствии корней в словесной форме: для этого достаточно оператор вывода, расположенный после *then*, заменить оператором *write* ('*корней нет*'). В результате выполнения этого оператора вывода на экране появится текст *корней нет*.

Вообще, в операторе вывода могут указываться не только выражения, но и любые тексты, каждый из которых ограничен с обеих сторон символом ' (штрих); будет выведен сам текст без штрихов. Так, если второй оператор вывода в программе *корни3* заменить на *write*(' $x_1 = ', (-b + d)/a, ', x_2 = ', (-b - d)/a$ '), то выполнение программы при наличии корней будет приводить к высвечиванию на экране легко читаемой информации

$x_1 = \dots, x_2 = \dots$

(вместо многоточий появятся, разумеется, настоящие значения корней). С помощью пробелов можно отделять друг от друга группы символов или числа в строке экрана. В рукописном тексте программы пробел часто изображают символом  $\square$ . Этот символ особенно удобен, если надо указать какое-то определенное количество пробелов, идущих подряд:

*write*(' $x_1 = ', (-b + d)/a, ', \square\square\square x_2 = ', (-b - d)/a$ )

В результате выполнения последнего оператора на экране будет высвечено

$x_1 = \dots, x_2 = \dots$

с конкретными числами вместо многоточий.

В Паскале имеется оператор вывода *writeln*, который выполняется так же, как *write*, с той разницей, что после его выполнения происходит переход на новую строку экрана. Переход на новую строку без вывода значений, в соответствии со сказанным, может быть предписан с помощью оператора *writeln*(' '), или, более четко, *writeln*(' $\square$ ') — напечатать пробел, т. е. ничего не напечатать, и перейти на другую строку. Последовательность операторов вывода

*writeln*(' $\square$ ');

*writeln*(' $x_1 = ', (-b + d)/a$ );

*writeln('x2=','(-b-d)/a')*

задает вывод корней в следующем виде:

*x1 = ...*

*x2 = ...*

с конкретными числами вместо многоточий. Если бы не было самого первого оператора *writeln('□')*, то это могло бы привести к появлению *x1 = ...* в строке, частично уже заполненной ранее:

*. . . . . x1 = ...*

*x2 = ...*

что может оказаться нежелательным.

## ЗАДАЧИ

61. Возможно ли, что среди операторов, входящих в составной оператор, встречается составной оператор?

62. Указать составные операторы среди следующих последовательностей символов:

а) *begin begin a := 0 end end*

б) *begin a := 0 end*

в) *a := 0*

г) *if a < b then begin a := 0; b := 0 end*

63. Условный оператор

*if  $\text{sqr}(x) > 2$  then*

*begin if  $x > 2$  then  $y := x*x*x$  else  $y := 8$  end*

*else  $y := 4*\text{sqr}(x)$*

устанавливает зависимость значения *y* от *x*. Построить график этой зависимости.

64. а) Изменить программу *A* из предыдущего параграфа так, чтобы результат печатался на отдельной строке экрана в виде

*y = ...*

б) Изменить программу *площадь* из § 3 так, чтобы результат печатался на отдельной строке экрана в виде

*площадь равна ...*

(вместо многоточий должны помещаться конкретные результаты вычислений).

65. Как надо изменить последний оператор вывода в программе *корни3*, чтобы результаты печатались в виде

*x = ...*

*1*

*x = ...*

*2*

(вместо многоточий должны быть настоящие значения корней)?

66. Даны три числа  $a, b, c$ . Выяснить, верно ли, что  $a < b < c$ .  
 Ответ получить в текстовой форме: *верно* или *неверно*.

67. Написать программу полного исследования совокупности корней биквадратного уравнения  $ax^4 + bx^2 + c = 0$ . Если корней нет, то должно быть выведено текстовое сообщение об этом, иначе должны быть выведены два или четыре корня.

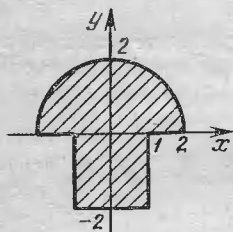


Рис. 6

68. Даны  $x, y$  ( $x \neq y$ ). Меньшее из этих двух чисел заменить их полусуммой, а большее — их удвоенным произведением.

69. Даны  $x, y$ . Выяснить, принадлежит ли точка  $(x, y)$  «грибу», изображенному на рис. 6.

70. Даны положительные  $a, b, c, d$ . Выяснить, можно ли прямоугольник со сторонами  $a, b$  уместить внутри прямоугольника со сторонами  $c, d$  так, чтобы каждая из сторон одного прямоугольника была параллельна или перпендикулярна каждой стороне второго прямоугольника.  
 Ответ получить в текстовой форме: *можно* или *нельзя*.

71. Даны положительные  $a, b, c, x$ . Выяснить, пройдет ли кирпич с ребрами  $a, b, c$  в квадратное отверстие со стороной  $x$ . Просовывать кирпич в отверстие разрешается только так, чтобы каждое из его ребер было параллельно или перпендикулярно каждой из сторон отверстия. Ответ получить в текстовой форме: *можно* или *нельзя*.

72. Даны три числа  $a, b, c$ . Удвоить каждое из данных чисел, если  $a \geq b \geq c$  и заменить числа их модулями в противном случае.

73. Даны  $x, y$ . Если  $x$  и  $y$  отрицательны, то каждое значение заменить его модулем; если отрицательно только одно из них, то оба значения увеличить на 0.5; если оба значения неотрицательны, то оба значения увеличить в 10 раз.

74. Если сумма трех попарно различных чисел  $x, y, z$  меньше единицы, то меньшее из  $x, y$  заменить полусуммой  $y$  и  $z$ , иначе большее из  $x$  и  $z$  заменить на  $y^4$ .

75. Даны положительные  $x, y, z$ . Выяснить, существует ли треугольник с длинами сторон  $x, y, z$ . Ответ получить в текстовой форме: *существует* или *не существует*.

## § 6. Оператор цикла

Множественно повторяемые действия могут быть заданы *оператором цикла*

*while B do P \*)*

\*) While — пока, do — выполнять.

где  $B$  — условие (отношение),  $P$  — любой оператор (называемый телом цикла). Выполняется выписанный оператор цикла так: проверяется условие  $B$ , и если оно соблюдается, то выполняется  $P$ , а затем вновь проверяется условие  $B$  и т. д. Как только на очередном шаге окажется, что условие  $B$  не соблюдается, то выполнение оператора цикла прекратится.

Если значение  $x$  положительно, то выполнение оператора цикла

*while*  $x \leq 0$  *do*  $x := x + 1$

прекратится после первой же проверки условия  $x \leq 0$ , и значение переменной  $x$  не изменится. Если же значение  $x$  не положительно, то к этому значению будет добавляться по единице до тех пор, пока значение не станет положительным. Предположим, например, что до выполнения этого оператора цикла переменная  $x$  имела значение 2.6. Тогда после выполнения оператора цикла переменная  $x$  будет по-прежнему иметь значение 2.6. Если же переменная  $x$  имела значение  $-3.7$ , то после выполнения оператора цикла она, как нетрудно проверить, будет иметь значение 0.3.

Рассмотрим некоторые программы, содержащие операторы цикла.

Пусть даны числа  $a$ ,  $b$  ( $a > 1$ ), и надо получить все члены бесконечной последовательности  $a$ ,  $a^2$ ,  $a^3$ , ... меньшие числа  $b$ . Программа:

```
program cmeneni(input, output);
  var a, b, c: real;
  begin read(a, b); writeln ('┐'); c := a;
    while c < b do
      begin
        writeln(c); c := c*a
      end
    end.
```

При выполнении этой программы переменная  $c$  последовательно принимает значения  $a$ ,  $a^2$ ,  $a^3$ , ... Изменение значения  $c$  происходит до тех пор, пока оно не станет больше или равно значению  $b$ . Если с самого начала  $a \geq b$ , то не будет выведено ни одного члена последовательности  $a$ ,  $a^2$ ,  $a^3$ , ...

Наш очередной пример касается вычисления  $\sqrt{u}$  для данного  $u \geq 0$ . Рассмотрим бесконечную последовательность  $x_1, x_2, x_3, \dots$ , образованную по следующему закону:

$$x_1 = \frac{u+1}{2}; \quad x_i = \frac{1}{2} \left( x_{i-1} + \frac{u}{x_{i-1}} \right), \quad i = 2, 3, \dots$$

Оказывается, что члены этой последовательности с увеличением номера  $i$  все меньше и меньше отличаются от  $\sqrt{u}$  (этот факт был

известен Герону Александрийскому V в. н. э.). Пусть, например,  $u=2$ . Вычисление первых членов последовательности  $x_1, x_2, x_3, \dots$  даст нам

$$x_1 = \frac{2+1}{2} = 1.5,$$

$$x_2 = \frac{1}{2} \left( 1.5 + \frac{2}{1.5} \right) \approx 1.4166666,$$

$$x_3 = \frac{1}{2} \left( 1.4166666 + \frac{2}{1.4166666} \right) \approx 1.4142156,$$

$$x_4 = \frac{1}{2} \left( 1.4142156 + \frac{2}{1.4142156} \right) \approx 1.4142135.$$

Напомним, что  $\sqrt{2} = 1.41421356\dots$ . Указанное свойство последовательности  $x_1, x_2, x_3, \dots$  часто используется для вычисления  $\sqrt{u}$  на компьютерах (для вычисления значения функции  $\text{sqr}(t)$ ). С целью демонстрации возможностей оператора цикла мы напишем программу вычисления  $x_i$  такого, что  $|x_i^2 - u| < 0.000001$ :

```
program kek(input, output);
  var u, x: real;
  begin read(u); x := (u + 1)/2;
    while abs(sqr(x) - u) >= 0.000001 do
      x := 0.5*(x + u/x);
    write(x)
  end.
```

При выполнении программы переменная  $x$  последовательно принимает значения  $x_1, x_2, \dots$ .

Пример. Напишем программу приближенного вычисления суммы

$$1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots *).$$

По условию задачи считается, что нужное приближение получено, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше, чем данное малое положительное число  $\varepsilon$ —это и все последующие слагаемые уже не надо учитывать.

На первый взгляд программа должна предписывать поочередное получение слагаемых путем возведения  $x$  в соответствующую степень, вычисления факториала и последующего деления одного значения на другое. Но надо всегда стараться так применять оператор цикла, чтобы каждый следующий шаг максимально исполь-

---

\*) Величина  $n!$  называется « $n$  факториал», ее значение равно  $1 \cdot 2 \cdot \dots \cdot n$ . Значение  $n!$  с увеличением  $n$  быстро растет. В частности, для любого  $x$  можно найти  $n$  такое, что  $n! > x^n$ .

зовал сделанное на предыдущих шагах. Если уже получено  $x^{i-1}/(i-1)!$ , то для вычисления  $x^i/i!$  достаточно домножить предыдущий результат на  $x/i$ .

Естественным будет следующее решение:

```
program сумма(input, output);
  var x, eps, i, y, s: real;
  begin read(x, eps);
    s := 0; y := 1; i := 0
    while abs(y) >= eps do
      begin
        s := s + y; i := i + 1;
        y := y * x / i
      end;
    write(s)
  end.
```

При выполнении этой программы переменная  $i$  последовательно принимает значения  $0, 1, 2, \dots$ , переменная  $y$  принимает значения  $1, \frac{x}{1!}, \frac{x^2}{2!}, \dots$ , переменная  $s$  в свою очередь принимает значения  $0, 1, 1 + \frac{x}{1!}, 1 + \frac{x}{1!} + \frac{x^2}{2!}, \dots$

Один из наиболее трудных моментов при составлении циклических программ — это присваивание переменным нужных начальных значений перед выполнением оператора цикла. Эти значения требуют тщательного подбора.

Пример с приближенным вычислением суммы — очень важный, и это связано с тем, что многие встречающиеся в математике, физике и других науках функции могут быть представлены бесконечными суммами:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots,$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

и т. д. Это означает, например, что если для данного  $x$  взять первое слагаемое бесконечной суммы

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots,$$

затем взять сумму первых двух слагаемых, сумму первых трех слагаемых и т. д., то мы получим бесконечную последовательность  $s_1, s_2, s_3, \dots$ , члены которой для больших номеров все меньше и меньше отличаются от  $\sin x$ .

Многие операторы цикла обладают той особенностью, что для некоторых (или даже для всех) начальных значений переменных



выполнение этих операторов не завершается. Так, если до выполнения оператора цикла

$while\ a < 1\ do\ a := 2*a$

переменная  $a$  имела меньшее или равное нулю значение, то сколько бы раз ни выполнялся оператор  $a := 2*a$ , неравенство  $a < 1$  будет оставаться в силе. Опасность такого «зацикливания» выполнения программы делает необходимой обстоятельную проверку всех деталей программы, содержащей операторы цикла.

В рассмотренных примерах приближенного вычисления (квадратного корня и бесконечной суммы) можно усмотреть применение некоторого общего приема: начальные члены той или иной бесконечной последовательности вычисляются один за другим, затем в определенный момент это вычисление прекращается, и последний вычисленный член объявляется результатом вычисления. Один из возникающих здесь вопросов — на каком именно члене бесконечной последовательности надо прерывать вычисления. Вопрос этот решается в разных конкретных задачах по-разному. Но в любом случае должно быть гарантировано завершение вычислений, или, точнее говоря, завершение выполнения оператора цикла.

## ЗАДАЧИ

76. Является ли оператором цикла:

а)  $while\ x < 0\ do\ x := x + 0.5$

б)  $while\ x < 0\ do\ x := x - 100$

в)  $while\ 0 < y < 1\ do\ y := \sqrt{y} + 0.01$

г)  $while\ a > 0\ do\ y := 2*y$

д)  $while\ a > b\ do\ a := a - 1; b := b + 1?$

77. Может ли завершиться выполнение оператора цикла, начало которого выглядит так:

$while\ \text{abs}(x) + 1 > 0.793\ do\ \dots?$

78. Для каких начальных значений переменной  $x$  завершится выполнение оператора цикла:

а)  $while\ x < 1.3\ do\ x := \sqrt{x}$

б)  $while\ \text{abs}(x) >= 1\ do\ x := x - 1$

в)  $while\ 2*x > x\ do\ x := x - 1$

г)  $while\ \sqrt{x} >= 0\ do\ x := \sin(x) + 1.315?$

79. Как надо изменить программу *степени* настоящего параграфа, чтобы в результате выполнения программы получался первый член последовательности  $a, a^2, a^3, \dots$ , больший или равный  $b$ ?

80. Переделаем программу *степени* настоящего параграфа следующим образом:

```

program cm(input, output);
  var a, b, c, d: real;
  begin read(a, b); c := a;
    while c < b do
      begin
        d := c; c := c * a
      end;
  write(d)
end.

```

Предположим, что  $b > a > 1$ . Какая задача решается программой *cm*?

81. Дано положительное  $a$ . Найти:

а) наибольшее число вида  $\frac{1}{2^n}$ ,  $n \geq 0$ , меньшее  $a$ ;

б) наименьшее число вида  $\frac{1}{3^n}$ ,  $n \geq 0$ , большее  $a$  (предполагается, что  $a < 1$ ).

82. Дано действительное  $a$ . Найти среди чисел  $1, 1 + \frac{1}{2}, 1 + \frac{1}{2} + \frac{1}{3}, \dots$  первое, большее  $a$ .

Отметим, что такое число может быть найдено, сколь бы большим ни было число  $a$ . Обозначим  $H_1 = 1, H_2 = 1 + \frac{1}{2}, H_3 = 1 + \frac{1}{2} + \frac{1}{3} \dots$ . Рассмотрим  $H_k$  для  $k = 2^m$ :

$$H_1 = 1,$$

$$H_2 = 1 + \frac{1}{2}.$$

$$H_4 = H_2 + \frac{1}{3} + \frac{1}{4} > H_2 + \frac{1}{4} + \frac{1}{4} = H_2 + \frac{1}{2} = 2$$

$$H_8 = H_4 + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} > H_4 + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = H_4 + \frac{1}{2} > 2\frac{1}{2}.$$

Продолжая эти рассуждения, можно показать, что  $H_{2^m} \geq 1 + \frac{m}{2}$ . Следовательно, среди  $H_{2^m}$  ( $m = 0, 1, \dots$ ) встречаются сколь угодно большие числа.

Интересно, что все члены «похожих» последовательностей

$$1, 1 + \frac{1}{2^2}, 1 + \frac{1}{2^2} + \frac{1}{3^2}, 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2}, \dots$$

$$1, 1 - \frac{1}{2}, 1 - \frac{1}{2} + \frac{1}{3}, 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4}, \dots$$

оказываются меньшими, чем 2 (см. далее задачу 85).

83. Дано действительное  $x$ . Вычислить приближенное значение бесконечной суммы:

$$а) x + \frac{x^2}{2} + \frac{x^3}{3} + \dots \quad (|x| < 1),$$

$$б) x - \frac{x^2}{2} + \frac{x^3}{3} - \dots \quad (|x| < 1),$$

$$в) x + \frac{x^3}{3} + \frac{x^5}{5} + \dots \quad (|x| < 1),$$

$$г) \frac{x-1}{x} + \frac{(x-1)^2}{2x^2} + \frac{(x-1)^3}{3x^3} + \dots \quad (x > 1/2),$$

$$д) \frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \dots \quad (x > 1).$$

Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше 0.00001.

84 Дано действительное  $\varepsilon > 0$ . Используя выписанные в тексте параграфа бесконечные суммы, составить программы для вычисления синуса и косинуса данного числа  $x$ . Нужно приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного  $\varepsilon$ .

В силу причин, связанных с представлением чисел в памяти компьютера, такое вычисление  $\sin x$  и  $\cos x$  может быть выполнено не для всех значений  $x$ . Оно не может, например, быть выполнено для таких  $x$ , значения модуля которых достаточно велико. Однако оно заведомо может быть выполнено при  $|x| \leq \pi/4$ .

85. Вычислить приближенно значение бесконечной суммы (справа от каждой суммы дается ее точное значение, с которым можно сравнить полученный ответ):

$$а) 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots \quad \frac{\pi^2}{6},$$

$$б) 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots \quad 0.6931478\dots,$$

$$в) 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots \quad \frac{\pi}{4},$$

$$г) \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \dots \quad 1,$$

$$д) \frac{1}{1 \cdot 3} + \frac{1}{2 \cdot 4} + \frac{1}{3 \cdot 5} + \dots \quad \frac{3}{4},$$

$$е) \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{2 \cdot 3 \cdot 4} + \frac{1}{3 \cdot 4 \cdot 5} + \dots \quad \frac{1}{4}.$$

Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа  $\varepsilon$ .

86. Изменить программу как так, чтобы:

а) значение  $x_1$  было данным положительным числом (последовательность  $x_1, x_2, \dots$  позволяет получить сколь угодно точные

приближения числа  $\sqrt[n]{u}$  не только при  $x_1 = \frac{u+1}{2}$ , но и при любом положительном  $x_1$ , если по-прежнему  $x_i = \frac{1}{2} \left( x_{i-1} + \frac{u}{x_{i-1}} \right)$  ( $i=2, 3, \dots$ );

б) вычислялось значение  $x_i$  такое, что  $|x_i^2 - u| < \varepsilon$ , где  $\varepsilon$  — данное положительное число.

87. Рассмотрим бесконечную последовательность  $x_1, x_2, \dots$ , образованную по следующему закону:

$$x_1 = \frac{u+n-1}{2},$$

$$x_i = \frac{1}{n} \left( (n-1)x_{i-1} + \frac{u}{x_{i-1}^{n-1}} \right) \quad (i=2, 3, \dots)$$

где  $u$  — данное неотрицательное действительное число,  $n$  — натуральное число. Эта последовательность позволяет получить сколько угодно точные приближения числа  $\sqrt[n]{u}$  (в конце книги мы вернемся к этому вопросу). По аналогии с программой *как* составить программы для приближенного вычисления:

а)  $\sqrt[3]{u}$ ,      б)  $\sqrt[4]{u}$ .

88. Дано действительное  $b > 0$ . Последовательность  $a_1, a_2, \dots$  образована по следующему закону:  $a_1 = 1$ ,  $a_i = a_{i-1}^2 + 1$  ( $i=2, 3, \dots$ ). Требуется получить все  $a_1, a_2, \dots$ , меньшие или равные  $b$ .

89. Дано действительное  $b > 0$ . Последовательность  $a_1, a_2, \dots$  образована по следующему закону:  $a_1 = b$ ,  $a_i = a_{i-1} - \frac{1}{\sqrt[i]{i}}$ ,  $i=2, 3, \dots$ . Найти первый отрицательный член последовательности  $a_1, a_2, \dots$ .

90. Дано действительное  $b < 0$ . Последовательность  $a_1, a_2, \dots$  образована по следующему закону:  $a_1 = b$ ;  $a_i = \frac{a_{i-1} + 1}{i - \sin^2 i}$ ,  $i=2, 3, \dots$ . Найти первый неотрицательный член последовательности  $a_1, a_2, \dots$ .

## ГЛАВА II

### ПРОСТЕЙШИЕ АЛГОРИТМЫ ПРИБЛИЖЕННЫХ ВЫЧИСЛЕНИЙ, КОМПЬЮТЕРНОЙ ГРАФИКИ, СИМВОЛЬНОЙ ОБРАБОТКИ

#### § 7. Приближенное решение уравнений

Рассмотрим задачу, для решения которой часто привлекаются компьютеры — задачу нахождения корней уравнений вида  $f(x)=0$ , где  $f(x)$  — некоторая функция. Хорошо известны формулы, позволяющие находить корни линейных и квадратных уравнений, т. е. таких уравнений вида  $f(x)=0$ , для которых  $f(x)$  представляет

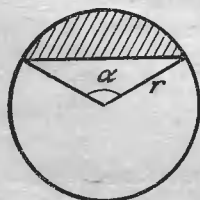


Рис. 7

собой функцию  $kx+b$  или  $ax^2+bx+c$  с конкретными числовыми коэффициентами  $k$ ,  $b$  или  $a$ ,  $b$ ,  $c$ . Однако этими простейшими алгебраическими уравнениями не исчерпывается круг тех уравнений, которые возникают в математике, физике, технике и т. д. Обнаруживается, что для решения многих представляющих практический интерес уравнений невозможно воспользоваться какими-то готовыми формулами.

Обратимся к примеру. Площадь кругового сегмента с радиусом  $r$  и с заданным в радианах центральным углом  $\alpha$  (рис. 7) равна  $\frac{r^2}{2}(\alpha - \sin \alpha)$ .

Рассмотрим задачу определения величины центрального угла по данному радиусу  $r$  и площади  $s$ . Получаем уравнение

$$x - \sin x - \frac{2s}{r^2} = 0.$$

По смыслу задачи  $0 < x < 2\pi$  и  $0 < s < \pi r^2$ .

Здесь не видно путей к получению точной формулы, т. е. такого выражения, значением которого для допустимых значений  $r$  и  $s$  было бы значение корня уравнения. Однако это не делает

положение безвыходным, так как можно воспользоваться алгоритмом приближенного решения уравнений такого рода. Один из алгоритмов будет описан ниже. Перед его описанием отметим, что практические задачи часто и не требуют точного нахождения корней. Если задача о вычислении значения центрального угла решается в процессе конструирования некоторого механизма, для которого нужна имеющая форму кругового сегмента деталь данной площади, то решением должна быть не формула (выражение, зависящее от  $r$  и  $s$ ), а явно указываемое число, которое может отличаться от истинного значения корня на некоторую малую величину.

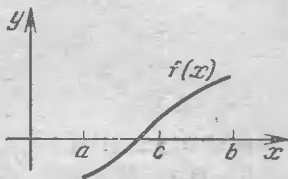


Рис. 8

Рассмотрим задачу в следующей постановке. Дано уравнение  $f(x) = 0$  и такие числа  $a, b$  ( $a < b$ ),\* что значения  $f(a)$  и  $f(b)$  имеют разные знаки:  $f(a)f(b) < 0$ , а график функции  $y=f(x)$  на отрезке  $[a, b]$  является непрерывной линией, т. е. линией, которую можно провести, не отрывая карандаша от бумаги (рис. 8). Понятно, что уравнение  $f(x)=0$  имеет корень на отрезке  $[a, b]$ , так как график функции  $y=f(x)$  обязательно пересечет ось абсцисс (пока рассмотрим случай единственного корня, отложив обсуждение общего случая до конца параграфа).

Требуется определить значение корня с погрешностью, не превосходящей данного положительного числа  $\varepsilon$ . Иначе говоря, если  $v$  — истинное значение корня, т. е. если  $a \leq v \leq b$  и  $f(v)=0$ , то требуется определить число  $w$  такое, что  $a \leq w \leq b$  и  $|v-w| < \varepsilon$ .

При такой постановке для решения задачи может быть применен алгоритм деления отрезка пополам. Взяв середину отрезка  $[a, b]$ , т. е. точку оси абсцисс с координатой  $c=(a+b)/2$ , можно сузить диапазон поиска корня: перейти от отрезка  $[a, b]$  к отрезку  $[a, c]$  или  $[c, b]$  в зависимости от знака  $f(c)$ : если  $f(a)f(c) < 0$ , то перейти к отрезку  $[a, c]$ , если  $f(a)f(c) > 0$ , то перейти к отрезку  $[c, b]$ . (В ситуации, изображенной на рис. 8, следует перейти к отрезку  $[a, c]$ .) Если затем найти середину меньшего отрезка и вычислить для нее значение функции  $f(x)$ , то можно будет вновь сузить диапазон поиска и т. д.

Каждый такой шаг уменьшает в два раза длину отрезка, в границах которого заведомо имеется корень уравнения  $f(x)=0$ . После нескольких шагов получится отрезок, длина которого будет меньше данного числа  $\varepsilon$ . Любая точка такого отрезка (например один из его концов) подходит в качестве решения поставленной задачи.

Рис. 9, на котором показано несколько этапов применения алгоритма, демонстрирует переходы от больших отрезков к мень-

шим. Цифра, надписанная над отрезком, показывает, какое по счету деление пополам позволило перейти к этому отрезку.

Может случайно оказаться, что для  $c = (a + b)/2$  получится  $f(c) = 0$ . Поэтому зафиксируем следующее правило для выбора

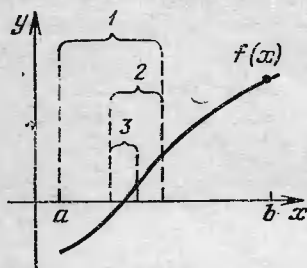


Рис. 9

меньшего отрезка, заведомо содержащего корень уравнения  $f(x) = 0$ : если  $f(a)f(c) \leq 0$ , то перейти к рассмотрению отрезка  $[a, c]$ , в противном случае — к рассмотрению отрезка  $[c, b]$ .

Напишем схему (набросок) программы, реализующей этот алгоритм. Здесь значения переменных  $a$  и  $b$  изменяются так, что остаются в силе неравенства  $a < b$  (или, иными словами,  $b - a > 0$ ) и

$f(a)f(b) \leq 0$ . Значением переменной  $fa$  является  $f(a)$ ;  $c$  обозначает середину отрезка  $[a, b]$ , переменная  $fc$  принимает значение, равное  $f(c)$ :

```

program делн (input; output);
  var a, b, c, eps, fa, fc, ... : real;
  begin ввести или явно определить a, b, eps и другие дан-
        ные задачи;
        вычислить f(a), присвоив это значение переменной fa;
        while b - a >= eps do
          begin c := (a + b)/2
                вычислить f(c), присвоив это значение переменной fc;
                if fa * fc <= 0 then b := c
                else
                  begin a := c; fa := fc end
                end;
          write (a)
        end.
  end.

```

То, что здесь написано, не является, разумеется, готовой для ввода в компьютер программой. Но, исходя из конкретного вида функции  $f(x)$ , эту схему можно легко превратить в настоящую программу. Вновь рассмотрим задачу определения центрального угла по площади сегмента и радиусу круга. Здесь  $a = 0$ ,  $b = 2\pi$ ,  $f(a) = -2s/r^2$ ,  $f(b) = 2\pi - 2s/r^2$ , при этом  $f(b) > 0$ , так как должны быть выполнены неравенства  $0 < s \leq \pi r^2$ . В программе имеет смысл описать переменную  $u$  и присвоить ей значение  $\frac{2s}{r^2}$  — это избавит от многократных вычислений этого значения.

Программа:

```
program делн (input, output);
  var a, b, c, eps, fa, fc, r, s, u: real;
  begin read (eps, r, s);
    u := (2 * s) / sqrt(r); a := 0; b := 2 * 3.14159;
    fa := -u;
    while b - a >= eps do
      begin c := (a + b) / 2; fc := c - sin(c) - u;
        if fa * fc <= 0 then b := c
        else begin a := c; fa := fc end
      end;
    write (a)
  end.
```

Обратим внимание на следующее обстоятельство. Если функция  $f(x)$  имеет непрерывный график на отрезке  $[a, b]$  и если  $f(a)f(b) < 0$ , то уравнение  $f(x) = 0$ , разумеется, имеет корень на отрезке  $[a, b]$ , но таких корней может быть несколько (см. рис. 10). Тогда выполнение программы, составленной по данной выше схеме, позволит все-таки получить один из этих корней—это следует из того, что после деления отрезка пополам мы всегда выбираем из двух меньших отрезков такой, в котором обязательно имеется корень уравнения  $f(x) = 0$ .

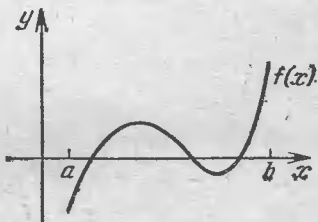


Рис. 10

Как правило, перед тем как прибегать к помощи подобных программ, пытаются определить границы  $a$  и  $b$  настолько точно, чтобы в отрезке  $[a, b]$  содержался ровно один корень. Составляя программу для конкретного уравнения  $f(x) = 0$ , иногда имеет смысл предусмотреть ввод с помощью оператора *read* границ  $a$  и  $b$ . Это позволит, задавая различные  $a$  и  $b$ , получать различные корни. Для получения границ  $a$  и  $b$  нет общего правила. Часто их помогают найти физические или геометрические соображения, так было в рассмотренной задаче определения величины центрального угла. Иногда удается выяснить картину расположения корней с помощью эскиза графика функции  $f(x)$  и т. д.

В ходе выполнения программы *делн* значения функции  $f(x)$  могут вычисляться компьютером приближенно (для такой функции, как  $x - \sin x + 2s/r^3$ , это вполне естественно). Строго говоря, по приближенному значению  $f(x)$  не всегда можно установить, справедливо ли неравенство  $f(x) \leq 0$ . Это касается, главным образом, случая, когда значение  $f(x)$  мало и погрешность вычисления значения



$f(x)$  превосходит само это значение. Пусть, например, в результате приближенного вычисления  $f(x)$  получилось значение 0.0000015, а погрешность вычисления равна 0.000002. Истинное значение функции может быть равно 0.0000035, но может быть равно и  $-0.0000005$ , это и служит причиной затруднений. Аналогично и проверка неравенства  $f(a)f(c) \leq 0$  затруднена, если значения  $f(a)$  и  $f(c)$  вычислены с ощутимой погрешностью. И в этом параграфе, и в подобных случаях в дальнейшем мы будем считать, что результаты приближенного вычисления  $f(a)$  и  $f(c)$  позволяют правильно отвечать на вопрос, верно ли, что  $f(a)f(c) \leq 0$ . Однако еще раз подчеркнем, что возможны нежелательные последствия этого допущения. Многие специалисты по математической логике вообще возражают против употребления слова «алгоритм», когда речь идет о решении уравнений вида  $f(x)=0$  путем последовательных делений отрезка пополам.

### ЗАДАЧИ

91. Найти с точностью 0.001 корни уравнений на указанных отрезках:

а)  $x^2 \cos 2x + 1 = 0$ ,  $\left[0, \frac{\pi}{2}\right]$ ;

б)  $x^3 + x^2 + x + 1 = 0$ ,  $[-2, 1]$ ;

в)  $x^5 - 0.3|x-1| = 0$ ,  $[0, 1]$ ;

г)  $2x - \cos x = 0$ ,  $\left[0, \frac{\pi}{2}\right]$ ;

д)  $0.9x - \sin \sqrt{x} - 0.1 = 0$ ,  $[0, 1.5]$ ;

е)  $\operatorname{tg} x - \frac{x+1}{2} = 0$ ,  $\left[0, \frac{\pi}{4}\right]$ .

92. Вычислив (без помощи компьютера) значение функции  $f(x)$  для ряда целых значений  $x$ , определить отрезок, содержащий хотя бы один корень уравнения  $f(x)=0$ , где  $f(x)$  — это  $x^3 - 2x^2 - 7x - 5$ .

93. Можно ли применять алгоритм деления отрезка пополам, если уравнение  $f(x)=0$  и соответствующий отрезок  $[a, b]$  имеют вид:

а)  $x^2 - 5 = 0$ ,  $[0, 3]$ ;

б)  $\sin x - 0.2 = 0$ ,  $\left[0, \frac{\pi}{2}\right]$ ;

в)  $\frac{1}{x} = 0$ ,  $[-1, 1]$ ;

г)  $x^4 + \cos x - 2 = 0$ ,  $[0, 2]$ ;

д)  $x^4 - 1 = 0$ ,  $[-5, 2]$ ?

94. Какой из корней уравнения  $f(x)=0$  будет найден в результате применения алгоритма деления отрезка пополам, если

отрезок  $[a, b]$ —это  $[0, 8]$ ,  $\varepsilon < 1$ , а график функции  $y=f(x)$  изображен на рис. 11 а—б.

95. Описания в тексте настоящего параграфа схемы программы *делп* таковы, что значение корня вычисляется с недостатком. Изменить программу так, чтобы:

- а) вычисление проводилось с избытком;
- б) недостаточность или избыточность приближения была непредсказуемой и зависела бы от конкретных  $a, b, \varepsilon$  и самой функции.

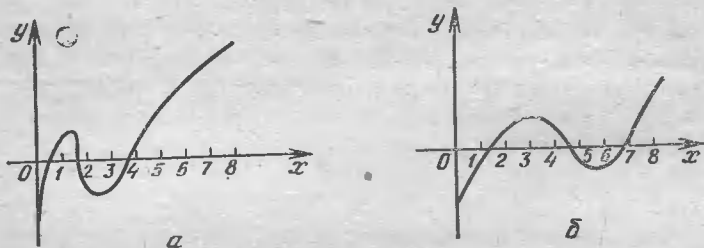


Рис. 11

96. Пусть к некоторому уравнению  $f(x) = 0$  применяется алгоритм деления отрезка пополам, причем отрезок  $[a, b]$ —это  $[1, 7]$ ,  $\varepsilon=0.1$ . Сколько значений функции  $f(x)$  будет найдено в ходе применения алгоритма? Тот же вопрос для отрезка  $[0, 10]$ .

97. Что получится, если попытаться применить программу, написанную в соответствии со схемой из данного параграфа, к уравнению  $f(x)=0$ , в котором функция  $f(x)$  такова, что  $f(x) > 0$  для всех  $x$  из отрезка  $[a, b]$ ?

98. Изменим в алгоритме деления отрезка пополам условие, при выполнении которого завершается процесс вычисления: вместо условия  $b-a < \varepsilon$  рассмотрим условие  $|f(c)| < \varepsilon$ , где  $c$ —середина отрезка  $[a, b]$ . Если это условие выполнено, то  $c$ —приближенное значение корня. В задаче определения центрального угла, соответствующего сегменту данной площади, это новое условие обеспечит выбор такого центрального угла, для которого площадь сегмента отличается от заданной меньше, чем на  $\varepsilon$ .

а) Переписать схему программы *делп* и саму программу, рассматривая условие  $|f(c)| < \varepsilon$ .

б) Аналогичным образом использовать в схеме программы и в самой программе условие  $b-a+|f(c)| < \varepsilon$ .

99. Пусть  $u > 0$ ,  $n$ —натуральное. Значение  $\sqrt[n]{u}$  можно найти, решив уравнение  $x^n - u = 0$ . Для того чтобы применить метод деления отрезка пополам, надо сначала найти  $a$  и  $b$  такие, что функция  $x^n - u$  для  $x=a$  и  $x=b$  принимает значения разных знаков.

Какие числа можно взять в качестве  $a$  и  $b$ ? Для данного  $u$  определить  $\sqrt[3]{u}$ ,  $\sqrt[4]{u}$ .

100. На воду опущен шар радиуса  $r$ , изготовленный из вещества плотности  $\rho$  ( $\rho < 1$ ). Найти расстояние от центра шара до поверхности воды.

Указание. При составлении программы воспользоваться формулой объема шарового сегмента высоты  $h$ :  $V = \frac{1}{3} \pi h^2 (3r - h)$  (см. рис. 12).



Рис. 12

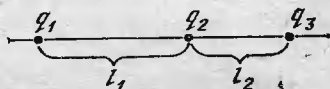


Рис. 13

101. На прямой расположены три положительных заряда величины  $q_1$ ,  $q_2$ ,  $q_3$  так, как показано на рис. 13. Определить расстояния от заряда  $q_1$  до точек, в которых равнодействующая сил отталкивания зарядами  $q_1$ ,  $q_2$ ,  $q_3$  некоторого четвертого положительного заряда равна нулю.

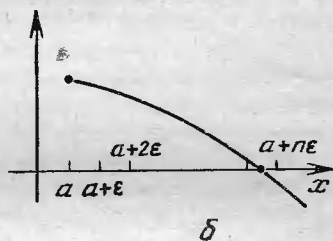
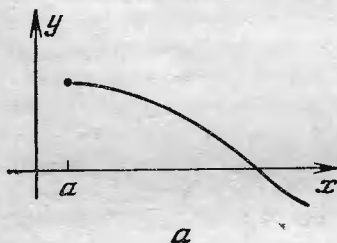


Рис. 14

102. Пусть график функции  $y = f(x)$  является при  $x > a$  непрерывной линией ( $a$  — некоторое число). Пусть  $f(a) \neq 0$  и пусть известно, что функция  $f(x)$  меняет знак при возрастании  $x$  (рис. 14а).

Корень уравнения  $f(x) = 0$  можно попытаться найти, вычисляя последовательно значения  $f(a)$ ,  $f(a + \varepsilon)$ ,  $f(a + 2\varepsilon)$ , ... до первого  $f(a + n\varepsilon)$ , знак которого отличается от знака  $f(a)$  (рис. 14б). Приближенное значение корня принимается равным  $a + n\varepsilon$ . Реализовать этот подход в программе вычисления  $\sqrt[3]{u}$  (т. е. решения уравнения  $x^3 - u = 0$ ), где  $u$  — данное положительное действительное число,  $\varepsilon = 0.001$ ,  $a = 0$ . Недостатком этого подхода является

большой объем вычислений. Кроме того, возможны ситуации, аналогичные изображенной на рис. 15.

103. Сообщим без доказательства несколько фактов, касающихся приведенных алгебраических уравнений  $n$ -й степени вида  $f(x) = 0$ , где  $f(x) = x^n + u_{n-1}x^{n-1} + \dots + u_1x + u_0$  ( $u_0, u_1, \dots, u_{n-1}$  — действительные числа). Если такое уравнение имеет действительные корни, то каждый из корней по модулю меньше, чем  $M = 1 + \max(|u_0|, |u_1|, \dots, |u_{n-1}|)$ , т. е. принадлежит отрезку  $[-M, M]$ . Если степень уравнения нечетна, то  $f(-M)f(M) < 0$ . Всякое уравнение нечетной степени имеет по крайней мере один действительный корень, а уравнение четной степени (например квадратное) может не иметь действительных корней.

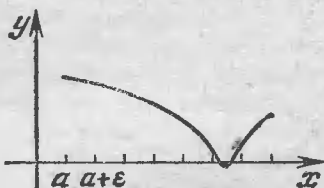


Рис. 15

Не прибегая к помощи компьютера, для каждого из следующих уравнений найти по приведенной формуле границу  $M$  для модулей корней и выяснить, верно ли, что  $f(-M)f(M) < 0$ :

- а)  $x - 2.5 = 0$ ; б)  $x^2 - 3x + 2 = 0$ ;  
в)  $x^3 - 2x^2 + 2x + 1 = 0$ ; г)  $x^7 - x^4 + x + 1 = 0$ .

104. Даны действительные  $p, q, r$ . Найти с точностью 0.0001 какой-нибудь один корень уравнения  $x^3 + px^2 + qx + r = 0$ . Использовать факты, сообщенные в условии предыдущей задачи.

105. Эта задача является продолжением двух предыдущих. Пусть уравнение третьей степени  $x^3 + px^2 + qx + r = 0$  имеет действительный корень  $\alpha$ . Тогда если это уравнение имеет еще какие-то действительные корни, то они совпадают с корнями квадратного уравнения  $x^2 + (\alpha + p)x + (\alpha^2 + p\alpha + q) = 0$ . Это можно доказать, проверив, что если  $\alpha^3 + p\alpha^2 + q\alpha + r = 0$ , то  $x^3 + px^2 + qx + r = (x - \alpha)(x^2 + (\alpha + p)x + (\alpha^2 + p\alpha + q))$ , и вспомнив, что произведение двух сомножителей равно нулю тогда и только тогда, когда по крайней мере один из сомножителей равен нулю. Используя это вместе с материалом, содержащимся в условиях двух предыдущих задач, написать программу нахождения приближенных значений всех действительных корней уравнения  $x^3 + px^2 + qx + r = 0$ , заданного коэффициентами  $p, q, r$  (вначале найти корень с точностью 0.0001, затем исследовать и решить квадратное уравнение).

## § 8. Целые числа

В § 3 было приведено схематическое изображение программы. Тогда нам было известно всего лишь три вида операторов. В дальнейшем понятие оператора существенно расширилось, теперь же

у нас появится новый вид описания переменных, в котором вместо *real* пишется *integer* \*), например:

*i, j: integer*

В данном случае мы имеем дело с описанием переменных целого типа, или переменных типа *integer*. Применявшиеся ранее описания были описаниями переменных действительного типа, или переменных типа *real*.

Переменные типа *integer* могут принимать лишь целые значения, а переменные типа *real* — как целые, так и нецелые. Числа в Паскале тоже разделяются на целые, или имеющие тип *integer*, и действительные, или имеющие тип *real*: если в записи числа использована точка, то оно действительное, в противном случае — целое.

Примеры целых чисел:

0, -3, 17, +193, -1000000, 5

Примеры действительных чисел:

3.14, -0.0001, 0.6, 17.0, -19.19191919, 5.0

Объясним причину, по которой выделяются переменные и числа типа *integer*. Уже говорилось, что значения выражений, в которые входят переменные, числа типа *real* и некоторые функции, могут вычисляться приближенно, т. е. с небольшой погрешностью. Значение выражения  $0.1 * 10$  может вычисляться, например, как 0.999999 или 1.000001, поэтому, как правило, не имеет смысла проверять выполнение равенства между действительными значениями выражений. В то же время операции сложения, вычитания и умножения над целыми числами производятся абсолютно точно, и результатами этих операций снова являются целые числа. Целочисленное значение выражений может быть сохранено в точном виде для дальнейших вычислений присваиванием этого значения переменной типа *integer*.

Программа вычисления  $n!$  (т. е. произведения  $1 \cdot 2 \cdot \dots \cdot n$ ):

```
program факт1(input, output);  
  var n, i, p: integer;  
  begin read(n); p:=1; i:=0;  
    while i < n do  
      begin i:=i+1; p:=p*i end;  
    write('n! = ', p)  
  end.
```

Здесь вычисление проводится в  $n$  шагов. Переменная  $p$  последовательно принимает значения  $1, 1 \cdot 2, 1 \cdot 2 \cdot 3, \dots, 1 \cdot 2 \cdot \dots \cdot n$ . Если бы

---

\*) Integer — целый.

переменные  $n$ ,  $i$ ,  $p$  были описаны как переменные типа *real*, то, после того как в произведение вошло  $n$  сомножителей, переменная  $i$  могла бы иметь значение, чуть меньшее  $n$  и в итоге имели бы приближенное значение  $(n+1)!$ .

Еще один вариант программы вычисления факториала:

```
program факт2(input, output);
  var n, p: integer;
  begin read(n); p:=1;
    while n > 0 do
      begin p:=p*n; n:=n-1 end;
    write('n!=', p)
  end.
```

Если переменная типа *integer* получает значение в результате выполнения оператора ввода, то вводимое с клавиатуры число не должно содержать десятичной точки (ср. с подстрочным примечанием на с. 23).

Рассмотрим пример программы, в которой встречаются переменные разных типов: вычисление суммы  $1 + \frac{1}{2} + \dots + \frac{1}{n}$

```
program C(input, output);
  var i, n: integer; s: real;
  begin read(n); s:=0; i:=0;
    while i < n do
      begin i:=i+1; s:=s+1/i end;
    write('s=', s)
  end.
```

Описания переменных разных типов могут произвольно чередоваться друг с другом. В последней программе мы могли бы написать

```
var s: real; i, n: integer;
```

Займемся программой нахождения чисел Фибоначчи. Числа Фибоначчи — это члены последовательности  $u_0, u_1, u_2, \dots$ , образованной по следующему закону  $u_0=0$ ;  $u_1=1$ ;  $u_i=u_{i-1}+u_{i-2}$ ,  $i=2, 3, \dots$ , т.е. члены последовательности 0, 1, 1, 2, 3, 5, 8, ... Напишем программу нахождения  $u_n$  по данному натуральному  $n \geq 2$ . Воспользуемся тем, что если переменные  $v$  и  $w$  имеют, соответственно, значения  $u_i$  и  $u_{i-1}$ , то, для того чтобы эти переменные приняли значения  $u_{i+1}$  и  $u_i$ , достаточно выполнить операторы присваивания

```
r:=v+w; w:=v; v:=r.
```

Программа:

```

program  $\Phi$ (input, output);
  var n, v, w, r, i: integer;
  begin read(n);
    w:=0; v:=1; i:=1;
    while i < n do
      begin r:=v+w; w:=v; v:=r; i:=i+1
      end;
    write(v)
  end.

```

### ЗАДАЧИ

106. Можно ли быть уверенным, что результаты выполнения действий  $0.1 + 0.3 + 0.1$  и  $0.2 + 0.3$  совпадут?

107. Дано натуральное  $n$ . Вычислить:

а)  $2^n$ ,

б)  $\left(1 + \frac{1}{1^2}\right) \left(1 + \frac{1}{2^2}\right) \dots \left(1 + \frac{1}{n^2}\right)$ ,

в)  $\sqrt[n]{2 + \sqrt[n]{2 + \dots + \sqrt[n]{2}}}$ ,  
 $n$  корней

г)  $\frac{1}{\sin 1} + \frac{1}{\sin 1 + \sin 2} + \dots + \frac{1}{\sin 1 + \dots + \sin n}$ ,

д)  $\frac{\cos 1}{\sin 1} \cdot \frac{\cos 1 + \cos 2}{\sin 1 + \sin 2} \cdot \dots \cdot \frac{\cos 1 + \dots + \cos n}{\sin 1 + \dots + \sin n}$ .

108. Дано: действительное  $a$ , натуральное  $n$ . Вычислить:

а)  $a^n$ ,

б)  $a(a+1) \dots (a+n-1)$ ,

в)  $\frac{1}{a} + \frac{1}{a(a+1)} + \dots + \frac{1}{a(a+1) \dots (a+n)}$ ,

г)  $\frac{1}{a} + \frac{1}{a^2} + \frac{1}{a^3} + \dots + \frac{1}{a^{2n}}$ .

109. Дано: натуральное  $n$ , действительное  $x$ . Вычислить:

а)  $\sin x + \sin^2 x + \dots + \sin^n x$ ,

б)  $\sin x + \sin x^2 + \dots + \sin x^n$ ,

в)  $\sin x + \sin \sin x + \dots + \underbrace{\sin \sin \dots \sin x}_{n \text{ раз}}$ .

110. Дано: действительные  $a, h$ , натуральное  $n$ . Вычислить  $f(a) + f(a+h) + f(a+2h) + \dots + f(a+nh)$ , где  $f(x) = (x^2 + 1) \cos x$ .

111. Пусть  $a_0 = 1$ ;  $a_k = k a_{k-1} + \frac{1}{k}$  ( $k = 1, 2, \dots$ ). Дано натуральное  $n$ . Получить  $a_n$ .

112. Пусть  $x_0=c$ ;  $x_1=d$ ;  $x_k=qx_{k-1}+rx_{k-2}+b$ ,  $k=2, 3, \dots$ . Дано: действительные  $q, r, b, c, d$ , натуральное  $n$  ( $n \geq 2$ ). Получить  $x_n$ .

113. Пусть  $a_0=a_1=1$ ;  $a_k=a_{k-2}+\frac{a_{k-1}}{2^{k-1}}$  ( $k=2, 3, \dots$ ). Найти произведение  $a_0a_1\dots a_{14}$ .

114. Дано: действительные  $a, b$ , натуральное  $n$  ( $b > a$ ). Получить  $(f_1+\dots+f_n)h$ , где  $h=\frac{b-a}{n}$ ,  $f_i=\frac{a+(i-0.5)h}{1+(a+(i-0.5)h)^2}$  ( $i=1, \dots, n$ ).

115. Дано целое  $m > 1$ . Получить наибольшее целое  $k$ , при котором  $4^k < m$ .

116. Вернемся к числам Фибоначчи (см. программу  $\Phi$  настоящего параграфа). Дано целое  $n$ . На этот раз на  $n$  мы накладываем ограничение  $n \geq 0$ , которое отличается от наложенного в тексте параграфа:  $n \geq 2$ .

а) Получить  $u_n$ .

б) Получить  $u_0, \dots, u_n$ .

117. Последовательность  $f_0, f_1, \dots$  образуется по закону:  $f_0=0$ ;  $f_1=1$ ;  $f_i=f_{i-1}+f_{i-2}+2^i$ ,  $i=2, 3, \dots$ . Дано целое  $n \geq 0$ . Получить  $f_0, f_1, \dots, f_n$ .

118. Дано: действительное  $x$ , натуральное  $n$ . Вычислить:

а)  $(x-2)(x-4)\dots(x-2n)$ ,

б)  $\frac{(x-2)(x-4)\dots(x-2n)}{(x-1)(x-3)\dots(x-2n-1)}$ .

119. Пусть надо получить значения функции  $\sin x$  для  $x=1, 1.1, 1.2, \dots, 2$ . В чем недостаток программы

```
program синус(input, output);
  var x: real;
  begin x:=1
    while x <= 2 do
      begin
        writeln(sin(x)); x:=x+0.1
      end
    end.
```

Предложить другой (более надежный) вариант.

Указание. Использовать в качестве тела цикла оператор

```
begin writeln(sin(1+0.1*i)); i:=i+1 end
```

120. Цилиндр объема 1 имеет высоту  $h$ . Определить радиус основания цилиндра для значений  $h$ , равных 0.5, 1, 1.5, ..., 4.5, 5.



## § 9. Дополнительные операции над целыми числами. Округление

Кроме уже названных в § 1 операций, в Паскале есть еще две операции, которые обозначаются *div* и *mod* \*). Эти операции имеют по два целых операнда (аргумента); если значения  $a$  и  $b$  неотрицательны и  $b \neq 0$ , то  $a \text{ div } b$  и  $a \text{ mod } b$  — это частное и остаток, возникающие при делении  $a$  на  $b$  с остатком. Например,  $17 \text{ div } 3 = 5$ ,  $17 \text{ mod } 3 = 2$ ,  $8 \text{ div } 2 = 4$ ,  $8 \text{ mod } 2 = 0$ ,  $1 \text{ div } 5 = 0$ ,  $1 \text{ mod } 5 = 1$  и т. д.

В общем случае для целых  $a$  и  $b$ ,  $b \neq 0$ , определение операций таково:

$$a \text{ div } b = \begin{cases} \left[ \frac{a}{b} \right], & \text{если } \frac{a}{b} \geq 0; \\ - \left[ \left| \frac{a}{b} \right| \right], & \text{если } \frac{a}{b} < 0, \end{cases}$$

[...] означает целую часть числа,

$$a \text{ mod } b = a - ((a \text{ div } b) * b).$$

Эти операции одного старшинства с умножением и делением, что важно иметь в виду при вычислении значений выражений. Операцию *mod* можно использовать, чтобы узнать, кратно ли целое  $a$  целому  $b$ . А именно:  $a$  кратно  $b$  тогда и только тогда, когда  $a \text{ mod } b = 0$ .

Пример программы, использующей введенные операции: алгоритм Евклида нахождения наибольшего общего делителя одновременно не равных нулю целых чисел  $a$ ,  $b$  ( $a \geq b \geq 0$ ).

```
program E(input, output);  
  var a, b, c: integer;  
  begin read(a, b);  
    while b > 0 do  
      begin c := a mod b; a := b; b := c end;  
    write('НОД = ', a)  
  end.
```

**Пояснение.** Алгоритм Евклида нахождения наибольшего общего делителя (НОД) неотрицательных целых чисел основан на следующих свойствах этой величины. Пусть  $a$  и  $b$  — одновременно не равные нулю целые числа, и пусть  $a \geq b$ . Тогда если  $b = 0$ , то  $\text{НОД}(a, b) = a$ , а если  $b \neq 0$ , то для следующих чисел  $a$ ,  $b$  и  $c$ , где  $c$  — остаток от деления  $a$  на  $b$ , выполнено равенство

---

\*) Div — сокращение слова division — деление, mod — сокращение латинского слова modulus — мера.

$\text{НОД}(a, b) = \text{НОД}(b, c) *$ ). Например,

$$\text{НОД}(15, 6) = \text{НОД}(6, 3) = \text{НОД}(3, 0) = 3.$$

В Паскале есть две функции округления:  $\text{trunc}(x)$ ,  $\text{round}(x)$  \*\*). Первая из них — это  $x$  без «дробных цифр»:  $\text{trunc}(3.14) = 3$ ,  $\text{trunc}(-3.14) = -3$ ,  $\text{trunc}(3) = 3$  и т. д. Функция  $\text{round}(x)$  дает округление  $x$  до ближайшего целого

$$\text{round}(x) = \begin{cases} \text{trunc}(x + 0.5), & \text{при } x \geq 0, \\ \text{trunc}(x - 0.5), & \text{при } x < 0, \end{cases}$$

$\text{round}(3.14) = 3$ ,  $\text{round}(3.7) = 4$ ,  $\text{round}(-3.14) = -3$ ,  $\text{round}(7) = 7$  и т. д. Значениями функций  $\text{trunc}$  и  $\text{round}$  всегда являются числа типа *integer*.

Пример. Напомним, что целая часть  $[x]$  действительного числа  $x$  определяется как наибольшее целое, не превосходящее  $x$ ; так  $[3.14] = 3$ ,  $[3] = 3$ ,  $[-3.14] = -4$ ,  $[-3] = -3$ . Для вычисления целой части можно воспользоваться программой

```
program Ц(input, output);
  var x: real; n: integer;
  begin read(x); n := trunc(x);
        if n > x then n := n - 1;
        write ('[x] = ', n)
  end.
```

Пример. Вычисление «расстояния» от данного действительного числа  $x$  до ближайшего целого, т. е. модуля разности между  $x$  и этим же числом, округленным до ближайшего целого задается оператором  $y := \text{abs}(x - \text{round}(x))$ .

Надо иметь в виду, что если слева от комбинации символов  $:=$  помещается переменная типа *integer*, то справа от этой комбинации символов может помещаться лишь такое выражение, которое либо вообще не содержит переменных типа *real*, чисел типа *real*, знака операции деления  $/$ , функций  $\sin$ ,  $\cos$ ,  $\text{sqrt}$ , либо содержит их только под знаками функций  $\text{trunc}$  и  $\text{round}$ . В силу принятого в Паскале способа выполнения арифметических операций и способа вычисления значений функций, значения этих и только этих выражений будут иметь тип *integer*. Такие выражения мы назовем выражениями со значениями типа *integer*. Справа от комбинации символов  $:=$  может размещаться любое выражение, коль скоро слева от этого знака помещена переменная типа *real*. Таким обра-

---

\*) Последнее равенство имеет место, потому что у чисел  $a$ ,  $b$  и у чисел  $b$ ,  $c$  при любом  $c$  вида  $a = bq$  ( $q$  — целое) один и тот же набор общих делителей. Следовательно, у чисел  $a$ ,  $b$  и чисел  $b$ ,  $c$  один и тот же наибольший общий делитель.

\*\*) Типы — сокращение от truncate — усекать, отрезать; round — круглый.

зом, если переменные  $i, j$  имеют тип *integer*, а переменные  $r, s$  — тип *real*, то недопустимы операторы присваивания:

$i := 3.14, i := r, j := 4/2, i := \text{sqrt}(\text{abs}(j)) + 2$  и т. д.;

допустимы операторы присваивания:

$r := 3.14, r := i, r := 4/2, r := \text{sqrt}(s),$

$i := \text{round}(4/2),$

$i := \text{abs}(j), i := i * j, i := i \text{ div } j,$

$j := 1 + \text{trunc}(1.5 + \sin(2.7 + r)).$

## ЗАДАЧИ

121. Верно ли, что  $b * a \text{ div } b = a$  тогда и только тогда, когда  $a$  кратно  $b$ ?

122. Дано действительное число  $x$ . Получить целую часть числа  $x$ , затем — число  $x$ , округленное до ближайшего целого, затем —  $x$  без дробных цифр.

123. Определить, является ли данное целое число четным.

124. Определить, верно ли, что при делении целого неотрицательного числа  $a$  на целое положительное число  $b$  получается остаток  $r$  или  $s$ .

125. Дано натуральное  $n \leq 100$ , определяющее возраст человека в годах. Дать для этого числа наименование *год, года* или *лет*. Например, 1 *год*, 23 *года*, 45 *лет* и т. д.

126. Поле шахматной доски может быть указано парой натуральных чисел, каждое из которых не превосходит восьми: первое число — это номер вертикали (при счете слева направо), второе — номер горизонтали (при счете снизу вверх). Даны натуральные  $k, l, m, n$ , каждое из которых не превосходит восьми.

а) Выяснить, являются ли поля  $(k, l)$  и  $(m, n)$  полями одного цвета.

б) На поле  $(k, l)$  расположен ферзь. Угрожает ли он полю  $(m, n)$ ?

в) Выяснить, можно ли с поля  $(k, l)$  одним ходом ладьи попасть на поле  $(m, n)$ . Если нет, то выяснить, как это можно сделать за два хода (указать поле, на которое приводит первый ход).

127. а) Дан номер года. Найти число дней в этом году.

б) Даны натуральные  $n, m$  ( $n \leq m$ ). Определить, сколько среди чисел  $n, n+1, \dots, m$  таких, которые являются номерами високосных годов.

*Указание.* В современном (григорианском) календаре каждый год, номер которого делится на 4, является високосным, за исключением тех, которые делятся на 100 и не делятся на 400. Например, 1900 год — не високосный, 2000 год — високосный и т. д.

128. Исследование проблемы «вечного календаря», т. е. проблемы быстрого определения дня недели, на который падает некоторая дата (число, месяц, год), показало, что если дата лежит в диапазоне от 1582 по 4902 гг., то номер дня недели (воскресенье имеет номер 0, понедельник — номер 1, ..., суббота — номер 6) равен остатку от деления на 7 значения выражения

$$[2.6m - 0.2] + d + y + [y/4] + [c/4] - 2c,$$

где  $d$  — номер дня в месяце,  $m$  — номер месяца в году, нумерация начинается с марта (март имеет номер 1, апрель — номер 2, ..., декабрь — номер 10, январь и февраль считаются месяцами с номерами 11 и 12 предыдущего года);  $y$  — число, образованное двумя младшими цифрами года,  $c$  — число, образованное двумя старшими цифрами года (15, ..., 49);  $[x]$  по-прежнему означает целую часть числа  $x$ .

Даны натуральные  $a$ ,  $b$ ,  $c$ , которые обозначают число, месяц и год. Определить день недели, на который падает указанная дата. Считать, что номер года лежит в диапазоне от 1582 по 4902.

129. Часовая стрелка образует угол  $\varphi$  с лучом, проходящим через центр циферблата и через точку, соответствующую 12 часам,  $0 < \varphi \leq 2\pi$ . Определить значение угла для минутной стрелки, а также количество часов и полных минут.

130. Дано действительное  $a$ . Вычислить  $f(a)$ , где  $f$  — периодическая функция с периодом 1.5, равная на отрезке  $[0, 1.5]$ :

а) функции  $x^3 - 2.25x$ ;

б) функции, график которой изображен на рис. 16.

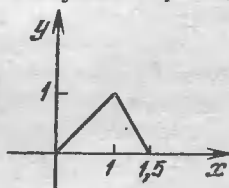


Рис. 16

**Указание.** Показать, что если периодическая функция  $f(x)$  с периодом  $t$  задана на отрезке  $[0, t]$ , то для любого  $x$  значение  $s = x - [x/t] \cdot t$  принадлежит отрезку  $[0, t]$ , и значение  $f(s)$ , в силу периодичности функции, равно  $f(x)$ . Способ вычисления целой части, основанной на операциях Паскаля, см. в программе Ц настоящего параграфа.

131. Для чисел Фибоначчи  $u_0, u_1, \dots$  (см. задачу 116) справедлива формула Бине:

$$u_k = \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^k - \frac{1}{\sqrt{5}} \left( \frac{1 - \sqrt{5}}{2} \right)^k \quad (k = 0, 1, \dots).$$

Так как  $\left| \frac{1 - \sqrt{5}}{2} \right| < 1$ , то для больших  $k$  выполнено приближен-

ное равенство  $u_k = \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^k$ . Вычислить и округлить до

ближайшего целого все числа  $\frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^k$  ( $k=0, 1, \dots, 15$ ),

а также вычислить  $u_0, u_1, \dots, u_{15}$  по формулам  $u_0=0; u_1=1; u_k=u_{k-1}+u_{k-2}$  ( $k=2, 3, \dots$ ) и сравнить результаты.

132. Дано натуральное  $n$ .

а) Сколько цифр в числе  $n$ ?

б) Чему равна сумма его цифр?

в) Найти первую цифру числа  $n$ .

133. Дано натуральное  $n$ . Выбросить из записи числа  $n$  цифры 1 и 5, оставив прежним порядок остальных цифр. Например, из числа 59015518 должно получиться 908.

134. Даны натуральные  $m, n$  ( $m < n$ ). Найти числитель и знаменатель несократимой правильной дроби  $\frac{p}{q}$  такой, что  $\frac{p}{q} = \frac{m}{n}$ .

135. Реализовать в виде программы следующий вариант алгоритма Евклида нахождения наибольшего общего делителя двух натуральных чисел: пока числа не станут равными, большее заменяется на разность большего и меньшего чисел. Когда числа станут равными одному и тому же числу  $d$ , то вычисления прекращаются:  $d$  равно наибольшему общему делителю исходных чисел. Обоснование этого варианта совпадает с тем, которое дано в подстрочном примечании на с. 57: достаточно положить  $q=1$ .

Отметим, что этот вариант основывается на очень простых операциях, но число операций больше того, которое требуется вариантом алгоритма Евклида, разобранным в тексте параграфа.

## § 10. Оператор цикла с параметром

Привлечение переменных типа *integer* позволяет использовать еще один оператор цикла, который называется *оператором цикла с параметром* \*):

*for i := A to B do S\*\*),*

здесь  $i$  — некоторая переменная типа *integer*, которая называется параметром цикла,  $A$  и  $B$  — выражения со значением типа *integer*,  $S$  — оператор (тело цикла).

Оператор цикла с параметром выполняется так. Сначала вычисляются значения выражений  $A$  и  $B$ . Получаются два целых числа, которые мы обозначим, соответственно, через  $\bar{A}$  и  $\bar{B}$ . Если  $\bar{A} \leq \bar{B}$ , то переменная  $i$  последовательно принимает значения, рав-

---

\*) Разбавившийся в § 6 оператор цикла будем называть *оператором цикла с условием*.

\*\*) For — для, to — к.

ные  $\bar{A}$ ,  $\bar{A}+1$ ,  $\bar{A}+2$ , ...,  $\bar{B}$ , и для каждого из этих значений выполняется оператор  $S$ . Если  $\bar{A} > \bar{B}$ , то оператор  $S$  не будет выполнен ни разу и выполнение оператора цикла с параметром сразу же закончится.

Пусть перед выполнением оператора цикла

```
for i:=1 to n do s:=s+i*i
```

переменная  $s$  имела значение 0 и пусть  $n > 0$ . Тогда после выполнения переменной  $s$  примет значение  $1^3 + 2^3 + \dots + n^3$ .

Программа вычисления  $n!$ :

```
program факт3(input, output);
  var n, i, p: integer;
  begin read(n); p:=1;
    for i:=1 to n do p:=p*i;
    write('n!=', p)
  end.
```

В § 8 уже рассматривались два варианта программы вычисления факториала *факт1*, *факт2*. Обе эти программы были написаны с использованием операторов цикла с условием. Вариант *факт3* несколько проще, так как в нем не пришлось выписывать операторы, описывающие изменение значений  $i$ .

Оператор цикла с условием необходимо применять в тех случаях, когда нет простого способа заранее определить число выполнений оператора  $S$ , исходя из значений переменных. Например, он очень удобен для описания алгоритма Евклида (программа *E* из § 9) алгоритма деления отрезка пополам (программа *делп* из § 7) и написания целого ряда программ, аналогичных программам *квк* и *сумма* из § 6. Но в тех случаях, когда число необходимых шагов легко определяется по исходным данным (например, видно, что это число равно значению переменной  $n$ ), оператор цикла с параметром выглядит, как правило, проще.

Имеется еще один вариант оператора цикла с параметром:

```
for i:=A downto B do S*).
```

Здесь  $i$  принимает последовательно значения  $\bar{A}$ ,  $\bar{A}-1$ , ...,  $\bar{B}^{**}$ , и для каждого из них выполняется оператор  $S$ , если же  $\bar{A}$  меньше  $\bar{B}$ , то оператор  $S$  не выполняется ни разу.

\*) Down to — вниз к.

\*\*) Мы по-прежнему обозначаем через  $\bar{A}$  и  $\bar{B}$  те значения выражений  $A$  и  $B$ , которые они имеют в начальный момент выполнения оператора цикла.

Еще один вариант программы вычисления факториала:

```
program факм4(input, output);  
  var n, i, p: integer;  
  begin read(n); p:=1;  
    for i:=n downto 2 do p:=p*i;  
    write('n!=',p)  
  end.
```

Рассмотрим еще примеры программ с операторами цикла с параметром.

Программа определения наименьшего среди чисел  $k^3 \sin\left(n + \frac{k}{n}\right)$  ( $k=1, 2, \dots, n$ ):

```
program наим(input, output);  
  var elem, min: real; k, n: integer;  
  begin read(n); min:=sin(n+1/n);  
    for k:=2 to n do  
      begin elem:=k*k*k*sin(n+k/n);  
        if elem < min then min:=elem  
      end;  
    writeln('min=',min)  
  end.
```

При выполнении этой программы наименьшее число определяется за  $n$  шагов: после выполнения  $k$ -го шага значение переменной  $min$  равно наименьшему числу среди первых  $k$  чисел последовательности. Первый шаг состоит в том, что переменной  $min$  присваивается значение первого члена последовательности. Остальные  $n-1$  шагов выполняются с помощью оператора цикла. Каждый следующий шаг выполняется с учетом значения, полученного на предыдущем шаге.

Программа вычисления суммы 70 данных действительных чисел:

```
program с70(input, output);  
  var a, s: real; i: integer;  
  begin s:=0;  
    for i:=1 to 70 do  
      begin read(a); s:=s+a end;  
    writeln('s=',s)  
  end.
```

В программе с70 строку `for i:=1 to 70 do` можно заменить строкой `for i:=70 downto 1 do` или строкой `for i:=0 to 69 do` — от этого ничего не изменится, так как сами значения параметра цикла при выполнении программы не используются, и существенным является только число шагов цикла.

Программу *c70* имеет смысл несколько преобразовать: программа требует многократного ввода данных, поэтому было бы хорошо, если бы во время ее выполнения на экран подавались какие-то сигналы о том, что пора указывать новое число. Такие сигналы иногда называют приглашениями. В качестве приглашения будем выводить на экран вопросительный знак. Получится программа:

```
program c70(input, output);
  var a, s: real; i: integer;
  begin s:= 0; writeln('┐');
    for i:= 1 to 70 do
      begin writeln('?');
        read (a); s:= s+a;
        writeln('┐')
      end;
    writeln('s=', s)
  end.
```

Во время выполнения программы на экране появится следующее:

```
?
┐
?
┐
...
?
┐
s=...
```

Здесь вытянутыми прямоугольниками заменены числа, которые вводятся с клавиатуры, а многоточием после знака равенства—результат вычисления.

И последний пример. Дано натуральное  $n$  и целые  $a_1, \dots, a_n$ . Предполагается, что  $a_1, \dots, a_n$ —попарно различны. Требуется определить наибольший член  $a_k$  последовательности  $a_1, \dots, a_n$  вместе с его порядковым номером  $k$ . Программа:

```
program наиб(input, output);
  var i, k, a, M: integer;
  begin writeln('┐'); writeln('?');
    read(n); writeln('?');
    read(M); k:= 1; writeln('┐')
    for i:= 2 to n do
      begin writeln('?');
        read(a);
        if a > M then
          begin M:= a; k:= i end;
```



```

        writeln('┐')
    end;
    write('наибольшее =', M, '┐ номер =', k)
end.

```

Полезно сравнить эту программу с программой *наим* из настоящего параграфа.

В заключение этого параграфа приведем два правила, принятых в Паскале:

1) по окончании выполнения оператора цикла с параметром значение параметра цикла считается неопределенным;

2) выполнение оператора, являющегося телом оператора цикла с параметром, не должно приводить к изменению значения параметра цикла.

### ЗАДАЧИ \*)

136. Дано натуральное  $n$ . Вычислить:

- а)  $\frac{1}{1} + \frac{1}{2} + \dots + \frac{1}{n}$ , б)  $\frac{1}{1^5} + \frac{1}{2^5} + \dots + \frac{1}{n^5}$ ,  
 в)  $\frac{1}{3^2} + \frac{1}{5^2} + \dots + \frac{1}{(2n+1)^2}$ , г)  $-\frac{1}{3} + \frac{1}{5} - \dots + \frac{(-1)^n}{2n+1}$ ,  
 д)  $\frac{1}{1 \cdot 2} - \frac{1}{2 \cdot 3} + \dots + \frac{(-1)^{n+1}}{n(n+1)}$ , е)  $-\frac{2}{1!} + \frac{3}{2!} - \dots + \frac{(-1)^n(n+1)}{n!}$ ,  
 ж)  $\frac{1!}{1} + \frac{2!}{1 + \frac{1}{2}} + \dots + \frac{n!}{1 + \frac{1}{2} + \dots + \frac{1}{n}}$ .

137. Дано: натуральное  $n$ , действительное  $x$ . Вычислить:

- а)  $\frac{x^1}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!}$ ,  
 б)  $\left(\frac{1}{1!} + \sqrt{|x|}\right) + \left(\frac{1}{2!} + \sqrt{|x|}\right) + \dots + \left(\frac{1}{n!} + \sqrt{|x|}\right)$ ,  
 в)  $\left(1 + \frac{\sin x}{1!}\right) \left(1 + \frac{\sin 2x}{2!}\right) \dots \left(1 + \frac{\sin nx}{n!}\right)$ ,  
 г)  $\left(\frac{1}{2} - \cos |x|\right) \left(\frac{2}{3} - \cos^2 |x|\right) \dots \left(\frac{n}{n+1} - \cos^n |x|\right)$ .

138. Вычислить:

- а)  $\frac{1}{1^2} + \frac{1}{2^2} + \dots + \frac{1}{100^2}$ , б)  $\frac{1}{1^3} + \frac{1}{2^3} + \dots + \frac{1}{50^3}$ ,  
 в)  $\frac{1}{1!} - \frac{1}{2!} + \dots - \frac{1}{10!}$ , г)  $\frac{1}{2^2} + \frac{1}{4^2} + \frac{1}{8^2} + \dots + \frac{1}{128^2}$ ,  
 д)  $\frac{1^2}{1^2+3} \cdot \frac{2^2}{2^2+3} \cdot \dots \cdot \frac{52^2}{52^2+3}$ .

\*) Полезно дополнительно прорешать задачи 107—114, 116—118, используя в программах оператор цикла с параметром.

$$е) \left(2 + \frac{1}{1!}\right) \left(2 - \frac{1}{2!}\right) \left(2 + \frac{1}{3!}\right) \dots \left(2 - \frac{1}{8!}\right),$$

139. Дано натуральное  $n$ . Получить последовательность  $b_1, \dots, \dots, b_n$ , где при  $i=1, 2, \dots, n$  значение  $b_i$  равно:

а)  $i$ , б)  $i^3$ , в)  $i!$ , г)  $2^{i+1}$ , д)  $2^i + 3^{i+1}$ , е)  $\frac{2^i}{i!}$ ,

ж)  $1 + \frac{1}{2} + \dots + \frac{1}{i}$ , з)  $1 - \frac{1}{2} + \dots + \frac{(-1)^{i+1}}{i}$ ,

и)  $i \left( \frac{1}{1!} + \frac{1}{2!} + \dots + \frac{1}{i!} \right)$ .

140. Дано: натуральное  $n$ , действительные  $a_1, \dots, a_n$ . Получить:

а)  $a_1 a_2 \dots a_n$ ,

б) среднее арифметическое  $a_1, \dots, a_n$ ,

в)  $a_1 + 2a_2 + 2a_3 + \dots + 2a_{n-1} + a_n$ ,

г)  $a_1 a_2 + a_2 a_3 + \dots + a_{n-1} a_n$  ( $n > 1$ ),

д)  $a_1 a_2 \dots a_k$ , где  $k=n$ , если  $n$  нечетное, и  $k=n-1$ , если  $n$  четное,

е)  $|a_1 - a_n|$ ,

ж)  $n + a_n$ ,

з)  $a_1, a_1 a_2, \dots, a_1 a_2 \dots a_n$ .

141. а) Вывести таблицу значений функции  $y = \sin x$ . Таблица имеет две колонки, в  $i$ -ю строку таблицы заносятся значения  $x_i$  и  $y_i$ , где  $x_i = 0.1i$ ,  $y_i = \sin x_i$  ( $i=0, 1, \dots, 15$ ).

б) Дано: действительные  $a, b$  ( $a < b$ ), натуральное  $n$ . Вывести таблицу значений функции  $f(x) = \cos \sqrt[2]{x}$ . Таблица имеет две колонки, в  $i$ -ю строку таблицы заносятся значения  $x_i$  и  $y_i$ , где  $x_i = a + \frac{b-a}{n}i$ ,  $y_i = f(x_i)$  ( $i=0, 1, \dots, n$ ).

в) Дано: действительное  $h > 0$ , натуральное  $n$ . Вывести таблицу значений функции  $f(x) = \frac{x^2 - 3x + 2}{\sqrt{2x^3 - 1}}$ . Таблица имеет две колонки, в  $i$ -ю строку таблицы заносятся значения  $x_i$  и  $y_i$ , где  $x_i = 1 + ih$ ,  $y_i = f(x_i)$ ,  $i=0, 1, \dots, n$ .

142. Пусть  $n$  — натуральное число. Обозначим через  $n!!$  произведение  $1 \cdot 3 \cdot \dots \cdot n$  для нечетного  $n$  и  $2 \cdot 4 \cdot \dots \cdot n$  для четного  $n$ .

Дано натуральное  $n$ . Получить  $n!!$

143. Даны целые  $a_1, \dots, a_{50}$ . Получить сумму тех членов  $a_i$  данной последовательности, которые:

а) кратны 5;

б) нечетны и отрицательны;

в) удовлетворяют условию  $|a_i| < i^2$ .

144. Пусть  $a_1 = b_1 = 1$ ;  $a_k = \frac{1}{2} \left( \sqrt{b_{k-1}} + \frac{1}{2} a_{k-1} \right)$ ;  $b_k = 2a_{k-1}^2 + b_{k-1}$  ( $k=1, 2, \dots$ ). Дано натуральное  $n$ . Найти  $a_1 b_1 + a_2 b_2 + \dots + a_n b_n$ .

145. Пусть  $x_1 = x_2 = x_3 = 1$ ;  $x_i = x_{i-1} + x_{i-3}$  ( $i = 4, 5, \dots$ ). Дано натуральное  $n$ . Найти  $\frac{x_1}{2} + \frac{x_2}{2^2} + \dots + \frac{x_n}{2^n}$ .

146. Даны натуральные  $n, q_1, \dots, q_n$ . Найти те члены  $q_i$  последовательности  $q_1, \dots, q_n$ , которые:

- являются удвоенными нечетными;
- при делении на 7 дают остаток 1, 2 или 5;
- обладают тем свойством, что корни уравнения  $x^2 + 3q_i - 5 = 0$  действительны и положительны.

*Указание.* Каждый раз, когда обнаруживается, что введенное число удовлетворяет поставленному условию, на экран можно вывести текст *последнее число удовлетворяет поставленному условию*. Можно сделать сообщение более конкретным, например: *последнее число является удвоенным нечетным* и т. д.

147. Дано натуральное  $n$ . Вычислить  $n$  сомножителей произведения

$$\frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \dots$$

148. Дано: натуральное  $n$ , действительные  $y_1, \dots, y_n$ . Найти:

- $\max(|z_1|, \dots, |z_n|)$ , где  $z_i = \begin{cases} y_i & \text{при } |y_i| \leq 2, \\ 0.5 & \text{в противном случае;} \end{cases}$
- $\min(|z_1|, \dots, |z_n|)$ , где  $z_i = \begin{cases} y_i & \text{при } |y_i| > 1, \\ 2 & \text{в противном случае;} \end{cases}$
- $z_1^2 + \dots + z_n^2$ , где  $z_i = \begin{cases} y_i & \text{при } 0 < y_i < 10, \\ 1 & \text{в противном случае.} \end{cases}$

149. Дано: натуральное  $n$ , действительное  $x$ . Среди чисел  $x^2 \sin kx$  ( $k = 1, \dots, n$ ) найти ближайшее к какому-нибудь целому.

150. В некоторых видах спортивных состязаний выступление каждого спортсмена независимо оценивается несколькими судьями, ватем из всей совокупности оценок удаляется наиболее высокая и наиболее низкая, а для оставшихся оценок вычисляется среднее арифметическое, которое и идет в зачет спортсмену. Если наиболее высокую оценку выставили несколько судей, то из совокупности оценок удаляется только одна такая оценка, аналогично поступают с наиболее низкими оценками.

Дано: натуральное  $n > 2$ , действительные положительные  $a_1, \dots, a_n$ . Считая, что числа  $a_1, \dots, a_n$  — это оценки, выставленные судьями одному из участников соревнований, определить ту оценку, которая пойдет в зачет этому спортсмену.

*Указание.* С помощью одного оператора цикла с параметром задать вычисление наибольшего из  $a_1, \dots, a_n$ , наименьшего из  $a_1, \dots, a_n$  и суммы  $a_1, \dots, a_n$ .

## § 11. Простейшие графические возможности компьютера

На экране современного компьютера можно получать не только последовательности букв, цифр и других символов, но и разнообразные рисунки, схемы и т. д. Для этого в языки программирования включаются специальные средства—графические операторы. Стандартного набора графических операторов для языка Паскаль не существует, и в дальнейшем мы для определенности будем использовать набор графических операторов одного из вариантов языка Паскаль, называемого Турбо-Паскалем. Им можно пользоваться, например, на персональных компьютерах IBM PC/XT и ЕС—1841. В этих компьютерах предусмотрено два основных режима работы экрана—символьный и графический. В графическом режиме имеется возможность высветить отдельно любую из 320 точек каждой из 200 точечных строк экрана. Точка здесь понимается как миниатюрный прямоугольник, размером примерно 0.8 на 1 мм \*). Каждая точка определяется парой целых чисел: ее порядковым номером в строке (нумерация точек в строке идет слева направо, начиная с 0) и порядковым номером строки экрана (строки нумеруются сверху вниз, начиная с 0). Максимальный номер точки в строке равен 319, максимальный номер строки—199.

Обычно экран компьютера находится в символьном режиме. Для переключения экрана в графический режим можно воспользоваться оператором *graphiccolormode* \*\*). В результате выполнения этого оператора экран очищается и переходит в цветной графический режим. В этом режиме можно одновременно использовать четыре цвета, которые пронумерованы целыми числами от 0 до 3. В стандартной ситуации соответствие между номером и цветом следующее: 0—черный (цвет экрана), 1—зеленый, 2—красный, 3—коричневый.

Для того чтобы высветить на экране отдельную точку, следует воспользоваться оператором *plot* \*\*\*), указав в скобках за идентификатором *plot* через запятую три выражения со значениями типа *integer*: значение первого выражения—номер точки в строке, значение второго—номер строки, значение третьего—номер цвета точки. Например, выполнение оператора *plot* (0, 0, 1) приведет к высвечиванию точки в левом верхнем углу экрана, выполнение оператора *plot*(319, 199, 1)—к высвечиванию точки в правом нижнем углу экрана, выполнение оператора *plot*(11, 131, 1)—к высве-

---

\*) Принят также термин *пиксел* (от picture element—элемент изображения).

\*\*) *Graphic color mode*—цветной графический режим.

\*\*\*) *Plot*—чертить, вычерчивать.

чиванию точки с номером 11 в строке с номером 131. Все три точки окрашены в зеленый цвет.

В результате выполнения программы

```
program точки;  
  var i: integer;  
  begin graphcolormode;  
    for i:=0 to 319 do plot(1, 10, 1)  
  end.
```

на экране появится отрезок прямой, состоящей из всех точек, расположенных в строке с номером 10 и окрашенных в зеленый

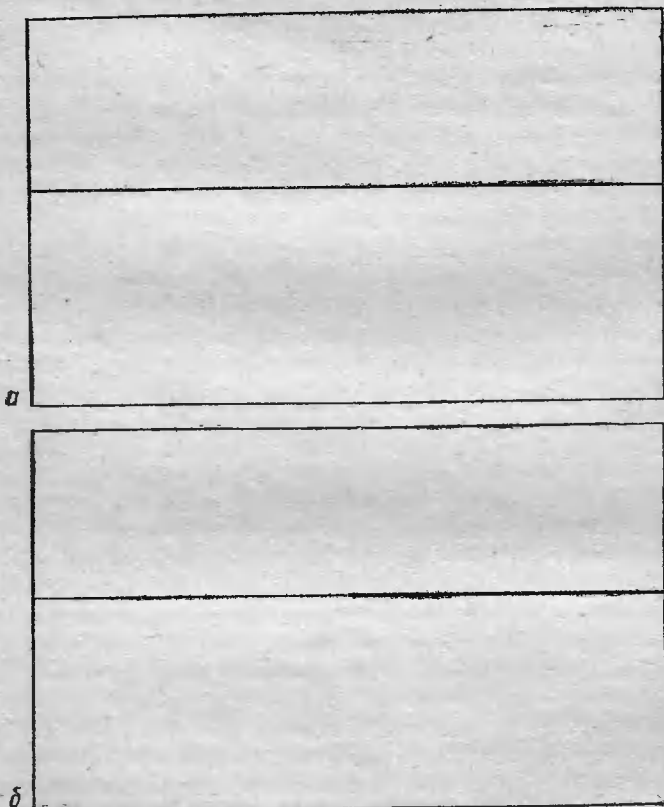


Рис. 17

цвет (рис. 17a). Если заменить оператор цикла этой программы на два оператора цикла

```
for i:=0 to 159 do plot(i, 10, 1);  
for i:=0 to 159 do plot(160+i, 10, 2)
```

то на экране будет высвечен тот же отрезок, левая половина которого окрашена в зеленый цвет, а правая — в красный (рис. 17б). Заметим, тот же результат можно получить с помощью одного оператора цикла

```
for i:=0 to 159 do  
begin plot(i, 10, 1); plot(160+i, 10, 2) end
```

В программе *точки* не встречаются операторы ввода и вывода. Поэтому после имени программы (идентификатора *точки*) нет скобок, в которые заключены идентификаторы *input* или *output*. Однако, если заменить заголовок программы на

```
program точки(output);
```

или на

```
program точки(input, output);
```

ошибки не будет.

В графическом режиме можно высвечивать сразу несколько точек экрана, лежащих на одной прямой. Для этого нужно воспользоваться оператором *draw*\*). Например, в результате выполнения оператора *draw*(0, 0, 10, 20, 2) на экране появится отрезок

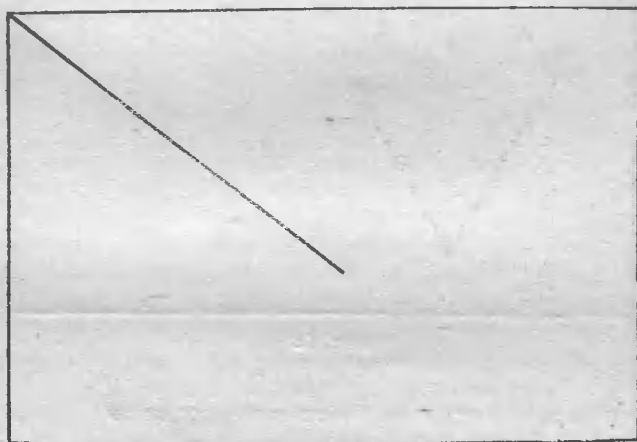


Рис. 18

прямой, соединяющий точки, задаваемые парами чисел 0,0 и 10, 20, и окрашенный в красный цвет (рис. 18). Общий вид оператора *draw* таков:

```
draw(x1, y1, x2, y2, c)
```

\*) Draw — проводить линию.

В скобках через запятую перечисляются выражения со значениями типа *integer*. Пары значений  $x_1$ ,  $y_1$  и  $x_2$ ,  $y_2$  задают положение концов отрезка на экране, значение  $c$  — цвет отрезка.

Пример. Выполнение программы

```
program ромб;  
  begin graphcolormode;  
    draw (100, 10, 50, 90, 1);  
    draw (50, 90, 100, 170, 1);  
    draw (100, 170, 150, 90, 1);  
    draw (150, 90, 100, 10, 1)  
  end.
```

приводит к высвечиванию ромба с вершинами, положение которых на экране определяется парами чисел 100, 10, 50, 90, 100, 170, и 150, 90 (рис. 19).

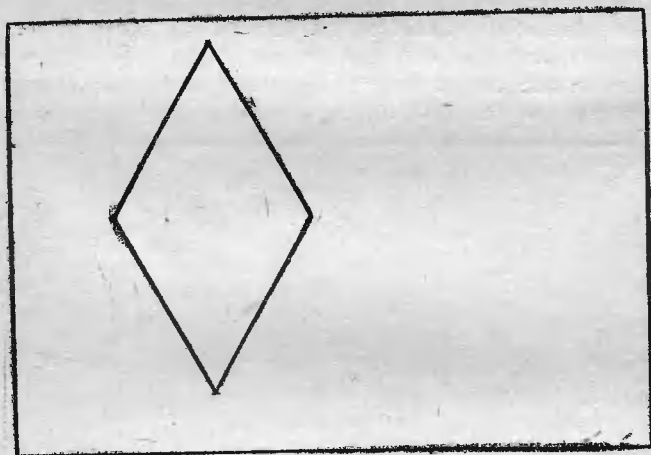


Рис. 19

Пример. В результате выполнения программы

```
program ступени;  
  var a, b, i: integer;  
  begin graphcolormode; a:=20; b:=20;  
    for i:=1 to 8 do  
      begin draw (a, b, a+20, b, 1);  
        draw (a+20, b, a+20, b+20, 1);  
        a:=a+20; b:=b+20  
      end  
    end.
```

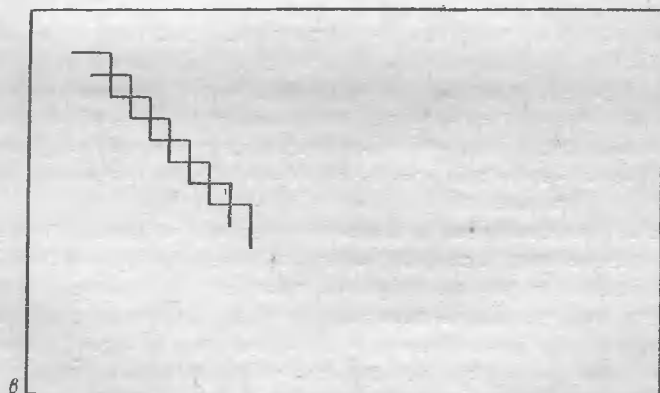
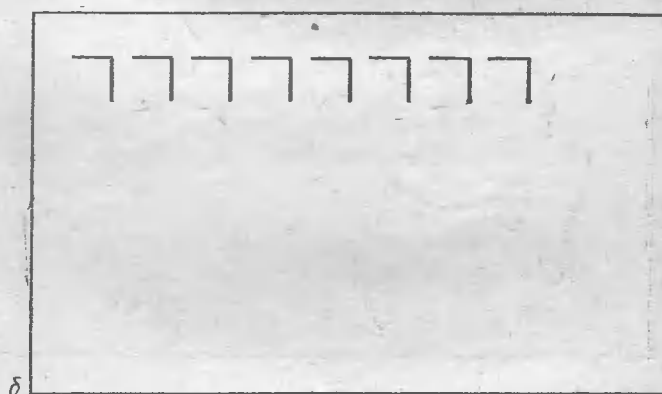
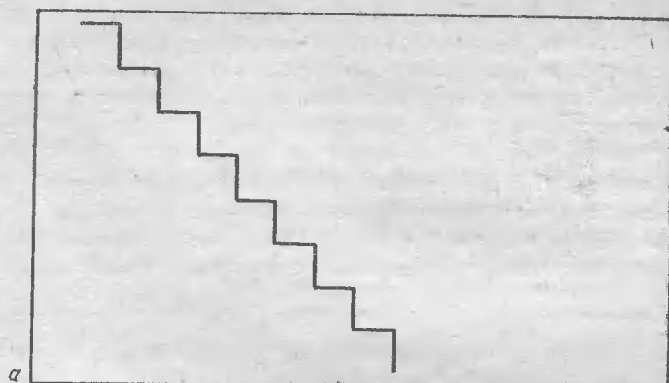


Рис. 20



на экране появится ломаная линия, состоящая из горизонтальных и вертикальных отрезков (рис. 20а). Построение линии выполняется в цикле, на каждом шаге цикла строится два отрезка—горизонтальный и вертикальный. При этом положение левого конца горизонтального отрезка на экране определяется парой значений  $a, b$ , положение правого конца—парой значений  $a+20, b$ ; положение верхнего конца вертикального отрезка определяется парой значений  $a+20, b$ , положение нижнего конца—парой значений  $a+20, b+20$ . После выполнения операторов, высвечивающих эти два отрезка, значения переменных  $a$  и  $b$  увеличиваются на 20.

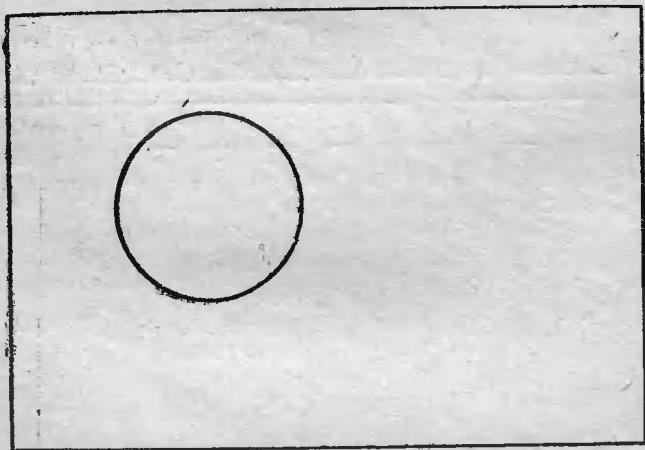


Рис. 21

Ясно, что от того, как меняются в теле цикла значения переменных  $a$  и  $b$ , будет зависеть изображение, получаемое на экране. Так, например, если заметить в рассмотренной программе операторы  $a:=a+20; b:=b+20$  на один оператор  $a:=a+25$ , то получим изображение, как на рис. 20б, если же ту же пару операторов заменить на операторы  $a:=a+10; b:=b+10$ , то получим изображение, как на рис. 20в.

В графическом режиме на экране можно получать не только изображения точек и прямых, но и изображения окружностей. Для высвечивания на экране окружности следует воспользоваться оператором *circle* \*), указав в скобках за идентификатором *circle* через запятую четыре выражения со значениями типа *integer*: значения первых двух выражений определяют положение центра окружности на экране, значение третьего выражения—радиус окружности

\*) Circle—круг, окружность.

(число точек экрана), значение четвертого выражения — цвет окружности. Например, выполнение оператора *circle* (100, 90, 45, 2) приводит к высвечиванию на экране окружности красного цвета, центр которой — точка с номером 100 в строке с номером 90, а радиус — 45 точек экрана (рис. 21).

В результате выполнения программы

```
program окр (input);  
  var i, a, b: integer;  
  begin read(a, b); graphcolormode;  
        for i:=1 to 10 do circle(a, b, 5*i, 3)  
  end.
```

на экране появится 10 концентрических окружностей коричневого цвета радиусов 5, 10, 15, ..., 50, положение центра которых определяется парой вводимых с клавиатуры значений *a*, *b*. Если при выполнении программы были введены числа 160, 100, то все изображение будет располагаться в середине экрана (рис. 22).

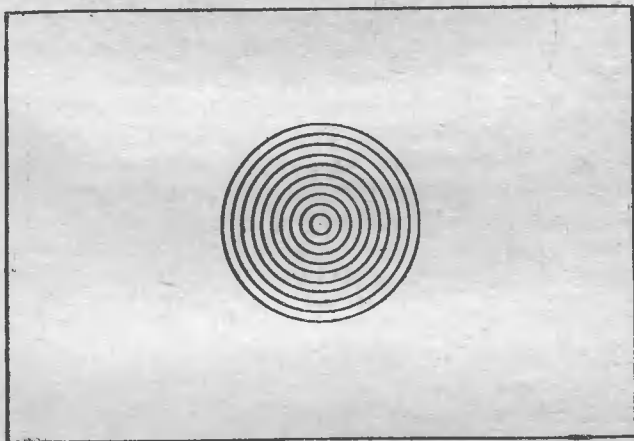


Рис. 22

Заметим, что в графическом режиме запрещено использование оператора ввода. Поэтому в рассмотренной программе ввод исходных данных выполняется до переключения экрана в графический режим.

Из простейших геометрических фигур, построенных с помощью операторов *plot*, *draw*, *circle*, можно составлять различные рисунки. Напишем программу, в результате выполнения которой на экране будет появляться рисунок, показанный на рис. 23. Размеры рисунка и местоположение его на экране будут определяться начальными данными.

```

program колобок(input);
  var x, y, r: integer;
      l, p, h, rm: integer;
  begin read(x, y, r); graphcolormode;
        circle(x, y, r, 1); l:=x-r div 2;
        p:=x+r div 2; h:=y-r div 2; rm:=r div 5;
        circle(l, h, rm, 1); plot(l, h, 2);
        circle(p, h, rm, 1); plot(p, h, 2);
        draw(x, y-r div 3, x, y+r div 3, 1);
        b:=y+2*(r div 3);
        draw(x-r div 3, h, x+r div 3, h, 1)
  end.

```

Рисунок строится из трех окружностей, двух точек и двух отрезков. Вводимые значения переменных  $x$  и  $y$  определяют положение центра большой окружности на экране.



Рис. 23

Значение переменной  $r$  — радиус большой окружности. Малые окружности располагаются внутри большой и имеют радиусы, равные частному от деления радиуса большой окружности на 5. Центры этих окружностей, определяющиеся парами значений  $l, h$  для левой окружности и  $p, h$  — для правой, высвечиваются красным цветом. Длины вертикального и горизон-

тального отрезков совпадают и равны  $2[r/3]$ . На рис. 24 показаны изображения, которые получаются на экране в результате выполнения программы *колобок* при вводе в качестве исходных данных чисел 160, 100, 85 (а), 40, 40, 25 (б), 300, 180, 5 (в).

## ЗАДАЧИ

### 151. Построить:

- треугольник, положение вершин которого на экране определяются парами чисел 50, 50; 100, 50; и 80, 190;
- четыреугольник, положение вершин которого на экране определяется парами чисел 70, 80; 170, 70; 170, 150 и 80, 140;
- шестиугольник, положение вершин которого на экране определяется парами чисел 120, 100; 140, 120; 140, 140; 120, 160; 100, 140 и 100, 120.

152. Построить квадрат со стороной  $30^*$ ), центр которого совмещен с центром экрана. Стороны квадрата должны быть параллельны сторонам экрана.

\*) Длины отрезков здесь и далее задаются числом точек.

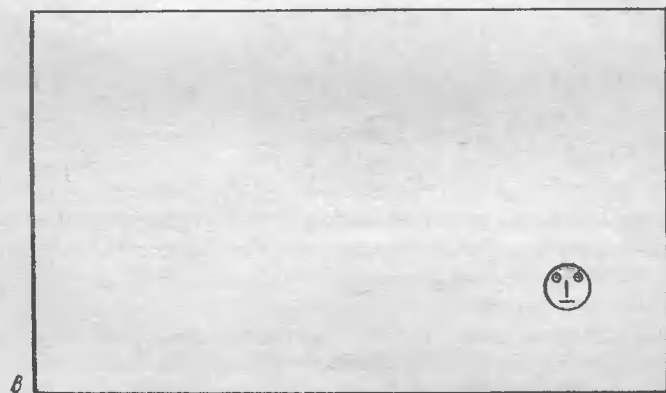
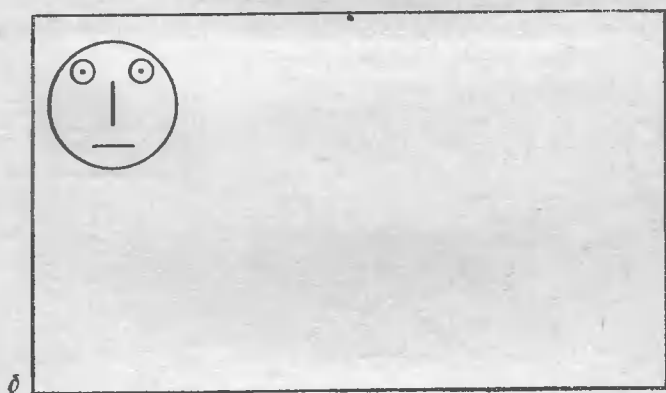
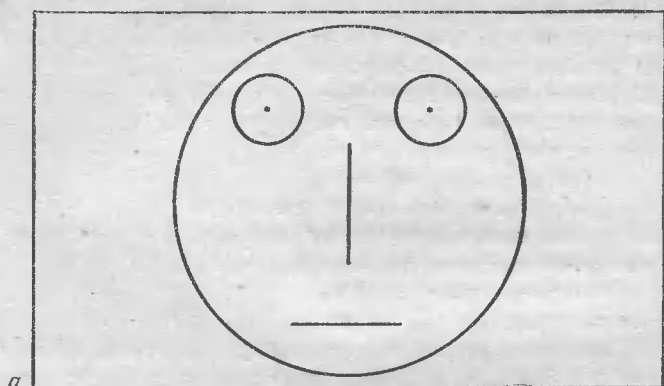


Рис. 24

153. Построить прямоугольник со сторонами 30 и 50, центр которого совмещен с центром экрана. Стороны прямоугольника должны быть параллельны сторонам экрана.

154. Какое изображение будет получено на экране, если в программе *ступени* заменить операторы  $a := a + 20$ ;  $b := b + 20$ :

а) на один оператор  $b := b + 20$ ;

б) на один оператор  $b := b + 5$ ;

в) на два оператора  $a := a + 20$ ;  $b := b + 15$ ?

155. Какое изображение будет получено на экране, если в программе *окр* заменить оператор  $circle(a, b, 5*i, 3)$ :

а) на оператор  $circle(a, b, 5, 3)$ ;

б) на оператор  $circle(a + 5*i, b, 10, 3)$ ?

156. Дано шесть целых чисел  $x_1, y_1, x_2, y_2, x_3, y_3$ . Каждая пара  $x_i, y_i$  ( $i = 1, 2, 3$ ) определяет положение одной из вершин треугольника на экране. Если данные числа определяют прямоугольный треугольник, высветить его на экране, в противном случае вывести сообщение *треугольник не прямоугольный*.

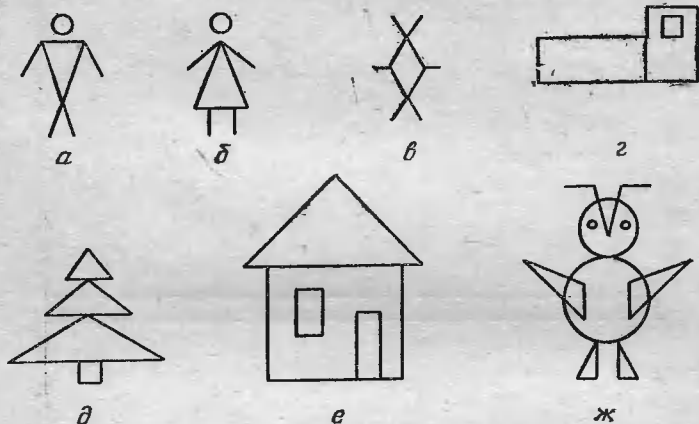


Рис. 25

157. Дано шесть целых чисел, определяющих положение вершин треугольника, расположенного в левой половине экрана. Построить на экране этот треугольник, а также треугольник, симметричный данному относительно вертикальной прямой, проходящей через середину экрана.

158. Дано три целых числа, определяющих положение центра окружности на экране и ее радиус. Если окружность не пересекает горизонтальную прямую, проходящую через середину экрана, то высветить данную окружность и окружность, симметричную данной относительно этой прямой.

159. Четыре целых числа задают положение концов отрезка на экране. Получить изображение этого отрезка и изображение отрезка, центрально-симметричного данному относительно точки, расположенной в центре экрана.

160. Рис. 25а — ж составлены из отрезков, окружностей и точек. Получить эти рисунки на экране.

161. Получить в центре экрана изображение, состоящее из 10 вложенных квадратов со сторонами 10, 20, 30, ..., 100 (рис. 26).

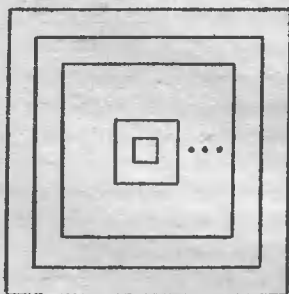


Рис. 26

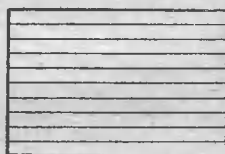
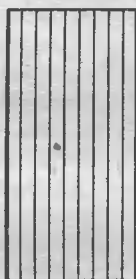


Рис. 27

162. Вывести на экран два прямоугольника. Один заштриховать вертикальными прямыми, другой — горизонтальными (см. рис. 27).

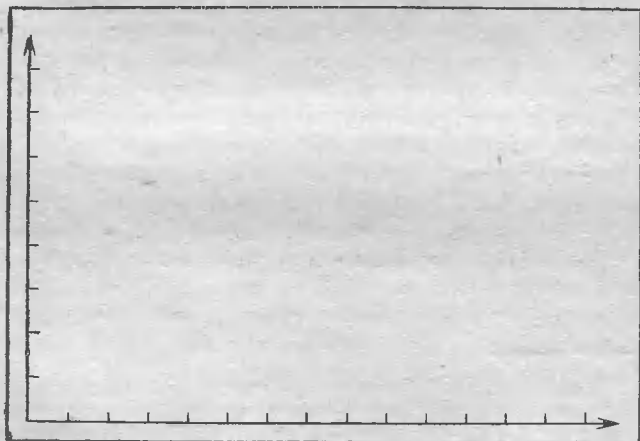


Рис. 28

163. Построить оси координат (рис. 28). Начало координат поместить вблизи левого нижнего угла экрана. Полуоси  $OX$  и  $OY$  разметить так, как показано на рис. 28.

164. Построить 9 концентрических окружностей, окрашенных поочередно в зеленый, красный и коричневый цвета.

165. Построить 10 вложенных квадратов (см. задачу 161), окрашенных поочередно в зеленый и красный цвета.

## § 12. Приближенное вычисление площади криволинейной трапеции

В этом параграфе мы будем рассматривать определенные на некотором отрезке  $[a, b]$  числовые функции, графики которых являются непрерывными линиями. Первоначально ограничимся слу-

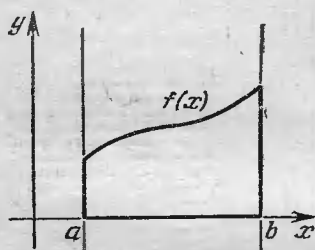


Рис. 29

чаем, когда функция неотрицательна на отрезке  $[a, b]$ . Нас будет интересовать площадь фигуры, ограниченной осью абсцисс, графиком функции  $f(x)$  и вертикальными прямыми  $x=a$ ,  $x=b$  (рис. 29). Такая фигура называется криволинейной трапецией. Алгоритм вычисления площадей криволинейных трапеций (и, в частности, алгоритмы приближенного вычисления) имеют большое значение

для решения многих задач. В качестве примера отметим задачи, связанные с движением тел \*). Если автомобиль, не изменяя направления своего движения, едет по шоссе с постоянной скоростью  $v_0$ , и если его движение продолжается от момента времени  $t=a$  до  $t=b$ , то график зависимости скорости автомобиля от времени есть отрезок прямой, параллельной оси  $t$ , а путь равен  $v_0(b-a)$ , т. е. площади прямоугольника, представленного на рис. 30а. Пусть движение не является равномерным, и пусть известна определенная на отрезке  $[a, b]$  функция  $v(t)$  такая, что ее значение при  $a \leq t \leq b$  — это скорость (показание спидометра) в момент времени  $t$  \*\*). Оказывается, что и в этом случае путь автомобиля равен площади фигуры, ограниченной осью  $t$ , графиком зависимости скорости от  $t$  и вертикальными прямыми  $t=a$ ,  $t=b$  (рис. 30б),

\*) Здесь и в дальнейшем при рассмотрении физических задач мы предполагаем (если не оговорено противное), что значения физических величин заданы в единицах международной системы СИ: время — в секундах, расстояние — в метрах и т. д.

\*\*) При изучении движения вдоль заданной линии (например, вдоль шоссе) можно ограничиться рассмотрением двух направлений движения — вперед (положительное направление) и назад (отрицательное направление). В одном случае скорость  $v(t)$  положительна, в другом — отрицательна.

т. е. равен площади криволинейной трапеции. Объяснение этому будет дано позже.

Рассмотрим простейший алгоритм приближенного вычисления площади криволинейных трапеций. Отрезок  $[a, b]$  разбивается на

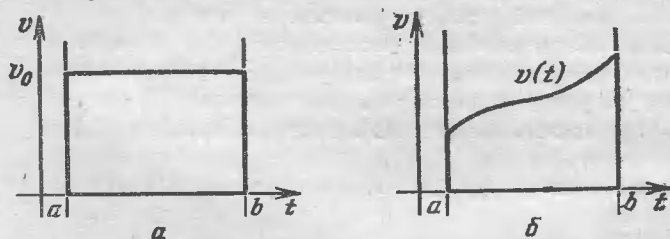


Рис. 30

несколько равных частей. Пусть число частей равно  $n$ , тогда каждая часть имеет длину  $h = (b-a)/n$ . Сама криволинейная трапеция при этом разбивается на  $n$  узких полос (рис. 31а). Площадь криволинейной трапеции, очевидно, равна сумме площадей всех полос,

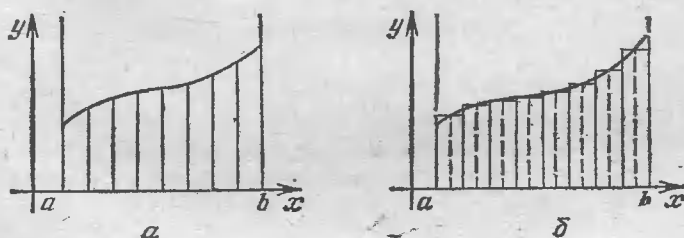


Рис. 31

а площадь каждой отдельной полосы может быть приближенно вычислена как  $hf(c_i)$ , где  $c_i$  — середина отрезка, лежащего в основании полосы. Фактически мы каждую полосу заменяем прямоугольником: ширина  $i$ -го прямоугольника для  $i=1, \dots, n$  равна  $h$ , а высота —  $f(c_i)$ , где  $c_i$  — середина основания  $i$ -й полосы (рис. 31б). Сумма площадей прямоугольников равна

$$hf(c_1) + \dots + hf(c_n) = h(f(c_1) + \dots + f(c_n)) = \\ = h\left(f\left(a + \frac{h}{2}\right) + f\left(a + \frac{h}{2} + h\right) + \dots + f\left(a + \frac{h}{2} + (n-1)h\right)\right).$$

На использовании последнего выражения основывается алгоритм приближенного вычисления площадей криволинейных трапеций, называемый алгоритмом прямоугольников.

Те рассуждения, которые привели нас к этому алгоритму, объясняют и связь задачи определения пути по зависимости скорости от времени с задачей определения площади. На протяжении



малого отрезка времени  $h$  скорость меняется незначительно, и движение можно считать равномерным со скоростью, измеренной в середине этого отрезка времени. Таким образом, весь путь приближенно равен площади ступенчатой фигуры, аналогичной изображенной на рис. 31б. Ясно, что уменьшение отрезков времени, на которых движение считается равномерным приводит к более точному описанию движения. Но при уменьшении  $h$  и площадь ступенчатой фигуры приближается к площади исходной фигуры. Это и позволяет сделать заключение о равенстве пути и соответствующей площади.

Напишем схему программы, реализующей алгоритм прямоугольников:

```

program площадь(input, output);
  var a, b, c, h, s ...: real;
      i, n ...: integer;
  begin ввести или явно определить a, b, n;
        h:=(b-a)/n; s:=0; c:=a-h/2;
        for i:= 1 to n do
          begin c:= c+h;
                добавить f(c) к значению s
          end;
        s = s*h; write ('площадь=',s)
  end.

```

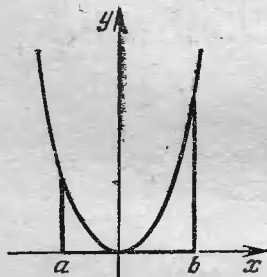


Рис. 32

Следует обратить внимание на значение, которое принимает переменная  $c$  перед выполнением оператора цикла:  $a - \frac{h}{2}$ . Благодаря этому на первом шаге цикла переменная  $c$  примет значение  $a - \frac{h}{2} + h$ , т. е.  $a + \frac{h}{2}$ , на втором  $-a + \frac{h}{2} + h$  и т. д.

Рассмотрим конкретный пример:  $f(x) = x^2$  (рис. 32). Предполагается,

что  $b > a$ . Программа:

```

program площадь (input, output);
  var a, b, c, h, s: real; i, n: integer;
  begin read(a, b, n);
        h:= (b-a)/n; s:= 0; c:= a-h/2;
        for i:= 1 to n do
          begin c:= c+h; s:= s+sqr(c) end;
        s = s*h; write ('площадь=',s)
  end.

```

Рассмотрим общий случай. Отказавшись от предположения, что функция  $f(x)$  неотрицательна на отрезке  $[a, b]$ , примем следующее соглашение: если график функции расположен под осью абсцисс, то соответствующая площадь считается отрицательной. На рис. 33 показаны знаки, которыми обладают площади отдельных частей криволинейной трапеции в случае знакопеременной функции  $f(x)$ . Выписанная выше формула приближенного вычисления площади, лежащая в основе алгоритма прямоугольников, охватывает и такие функции: если на некотором промежутке функция принимает отрицательные значения, то и значения площадей прямоугольников, равные  $hf(c_i)$ , будут отрицательными.

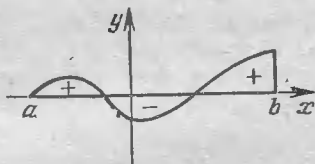


Рис. 33

Площадь криволинейной трапеции, определяемая с учетом знака функции так, как было объяснено выше с помощью рис. 33, называется интегралом функции  $f(x)$  на отрезке  $[a, b]$  и обозначается через  $\int_a^b f(x) dx$  \*). В последнем выражении символ  $\int$  — это

знак интеграла (который есть не что иное, как специальным образом написанная буква  $S$ , традиционно привлекаемая для обозначения площади),  $a$  и  $b$  — это, соответственно нижний и верхний пределы интегрирования,  $f(x)$  — подынтегральная функция; внесение множителя  $dx$  под знак интеграла для нас будет просто способом указания переменной интегрирования — для данного случая эта переменная есть  $x$ .

При определении пути по функции  $v(t)$  мы пользуемся интегралом  $\int_a^b v(t) dt$  и переменной интегрирования является  $t$ , но воп-

рос использования той или иной буквы не является принципиальным, и тот же самый путь получится при вычислении интеграла

$$\int_a^b v(x) dx.$$

Соглашение о знаках, проиллюстрированное с помощью рис. 33, естественно для задачи определения пути, если под путем понимать смещение автомобиля в фиксированном (положительном) направлении, потому что движение с отрицательной скоростью — это движение назад (движение в отрицательном направлении).

\*) Читается это выражение так: «интеграл от  $a$  до  $b$  эф от икс дэ икс».

Наряду с  $\int_a^b v(t) dt$  можно рассмотреть  $\int_a^b |v(t)| dt$ . Последний интеграл позволит вычислить тот путь, который можно оценить по имеющемуся в автомобиле счетчику пройденных километров.

Приближенную формулу, используемую в алгоритме прямоугольников, можно записать так:

$$\int_a^b f(x) dx \approx \\ \approx h \left( f\left(a + \frac{h}{2}\right) + f\left(a + \frac{h}{2} + h\right) + \dots + f\left(a + \frac{h}{2} + (n-1)h\right) \right).$$

Здесь  $h = \frac{b-a}{n}$ , величина  $h$  называется шагом приближенного интегрирования.

Понятие интеграла было введено в XVII веке И. Ньютоном и Г. Лейбницем. Это понятие и сегодня остается одним из важнейших понятий науки.

Исходя из строго математического определения интеграла, которое мы здесь не даем\*), построена теория, позволяющая,

в частности, находить для некоторых интегралов  $\int_a^b f(x) dx$  явные выражения через  $a$  и  $b$ : так, например, устанавливается, что

$$\int_a^b x^2 dx = \frac{b^3 - a^3}{3}, \quad \int_a^b \sin x dx = \cos a - \cos b$$

и т. д. В то же время доказано, что для многих функций невозможно построить явное выражение интеграла, в которое входили бы только элементарные («школьные») функции. Такого выражения

не существует, например, для интеграла  $\int_a^b \sin x^2 dx$ . Поэтому алгоритмы приближенного интегрирования играют важную роль в вычислительной практике.

## ЗАДАЧИ

136. Вычислить с помощью компьютера интегралы (справа от каждого интеграла дается его точное значение, которое можно использовать для проверки ответа). Использовать алгоритм прямоугольников, полагая  $n = 20$ .

\*) Строгое определение и доказательство свойств интеграла составляют предмет интегрального исчисления — раздела математического анализа.

$$\text{а) } \int_{-1}^2 \frac{x}{(x^2+1)^2} dx \quad 0.15,$$

$$\text{б) } \int_0^{1/2} 4 \cos^2 x dx \quad 1.84147098\dots,$$

$$\text{в) } \int_0^{\pi/3} \frac{dx}{\cos^2 x} \quad \sqrt{3} = 1.73205080\dots,$$

$$\text{г) } \int_4^9 \frac{x+1}{\sqrt{x}} dx \quad 14.6666666\dots$$

167. Определим среднее значение функции  $f(x)$  на отрезке  $[a, b]$  как  $\left( \int_a^b f(x) dx \right) / (b-a)$ . Например, среднее значение ско-

рости на некотором отрезке времени равно пути, пройденному за это время, деленному на продолжительность движения. Какие изменения надо внести в схему программы *площадь*, для того, чтобы программа обеспечивала приближенное вычисление среднего значения функции на отрезке  $[a, b]$ ?

168. Используя приближенное интегрирование, найти приближенное значение  $\pi$ . Для этого вычислить площадь четверти единичного круга (рис. 34), а затем увеличить ее в четыре раза. В алгоритме прямоугольников считать  $n=50$ .

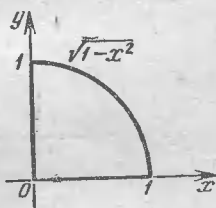


Рис. 34

169. Для получения  $\pi$  можно воспользоваться более удобным в вычислительном отношении интегралом, чем предложенный в предыдущей задаче, а именно интегралом

$$\int_0^1 \frac{dx}{1+x^2}$$

То, что последний интеграл равен  $\frac{\pi}{4}$ , доказывается в курсе математического анализа. Этот интеграл удобнее, так как его подынтегральная функция вычисляется проще. Получить  $\pi$  с помощью этого интеграла (снова взять  $n=50$ ).

170. Наряду с функцией  $f(x)$ , определенной на отрезке  $[a, b]$ , можно рассмотреть функцию  $F(x)$ , положив для любого  $c$  по определению  $F(c) = \int_a^c f(x) dx$ . Таким образом,  $F(c)$  — это площадь

«промежуточной» криволинейной трапеции, изображенной на рис. 35. Например,  $F(a)=0$ . Вывести таблицу значений функций  $f(x)$  и  $F(x)$ . Таблица имеет три колонки, в  $i$ -ю строку таблицы заносятся значения  $x_i$ ,  $f(x_i)$ ,  $F(x_i)$ , где  $x_i = a + \frac{b-a}{n}i$  ( $i=0, 1, \dots, n$ ).

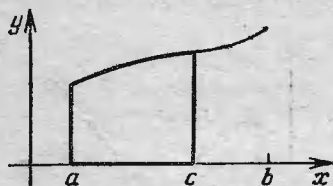


Рис. 35

В качестве  $f(x)$  рассмотреть подынтегральные функции из задачи 166, в качестве  $a$  и  $b$  — использованные там пределы интегрирования, в качестве  $n$  — данное число.

171. Очевидно, что алгоритм прямоугольников обеспечивает точное вычисление значения интеграла для любой постоянной функции. Указать еще бесконечное множество функций, для которых алгоритм прямоугольников дает точное значение интеграла.

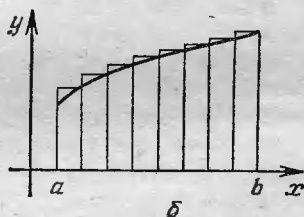
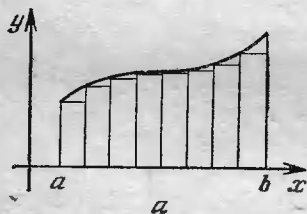


Рис. 36

172. Рассмотренный в тексте алгоритм часто называют алгоритмом средних прямоугольников, так как функция вычисляется в точке, являющейся серединой основания прямоугольника. Аналогичным образом можно рассмотреть алгоритм левых прямоугольников и алгоритм правых прямоугольников — функция вычисляется в левом конце основания и, соответственно, в правом конце (рис. 36а, б).

Написать программы, реализующие алгоритмы левых и правых прямоугольников, и вычислить по ним интегралы, указанные в задаче 166.

173. Пусть функция  $f(x)$  определена на отрезке  $[a, b]$ . Используя приближенное интегрирование, найти на отрезке  $[a, b]$  точку  $c$  такую, что «промежуточная» криволинейная трапеция (см. рис. 35) имеет наибольшую площадь. Найти также значения этой площади. (Задача представляет интерес для знакопеременной функции  $f(x)$ ; если же, например,  $f(x)$  положительна на  $[a, b]$ , то в качестве  $c$

надо, очевидно, взять  $b$ ). Рассмотреть  $f(x) = \sin x + \cos 5.6x$ . Положить  $a=0$ ,  $b=7$ ,  $n=60$ .

174. Дано действительное  $a$ . Пусть  $f(x)$  — такая функция, что  $f(a) > 0$  и  $f(x)$  обращается в 0 для некоторого  $x > a$ . Найти площадь криволинейной трапеции, изображенной на рис. 37. Вид  $f(x)$ , а также  $a$ ,  $b$  и  $n$  взять из условия предыдущей задачи.

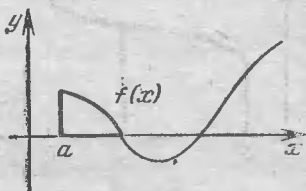


Рис. 37

175. Ракета была запущена вертикально вверх. Известна зависимость скорости полета от времени — функция  $v(t)$ . В начале полета скорость возрастала, а через некоторое время начала уменьшаться. Определить высоту ракеты над землей к тому моменту, когда скорость начала падать (рис. 38). Использовать для решения задачи алгоритм прямоугольников. Считать  $t_{\text{нач}}=0$ , в качестве  $v(t)$  рассмотреть  $(-t^2 + 20t - 90) \operatorname{tg}(t/15)$ ,  $h=0.2$ .

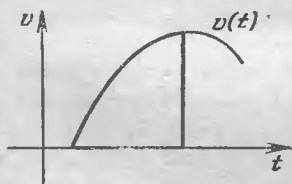


Рис. 38

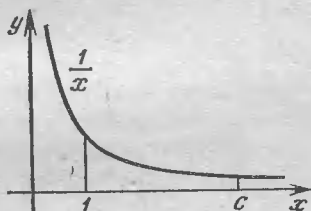


Рис. 39

176. Рассмотрим функцию  $\varphi(x)$ , определенную для любого числа  $c \geq 1$  следующим образом:  $\varphi(c)$  равно площади криволинейной трапеции, ограниченной осью абсцисс, графиком функции  $1/x$  и вертикальными прямыми  $x=1$ ,  $x=c$  (рис. 39).

В курсе математического анализа устанавливается, что при любых больших или равных единице числах  $a$  и  $b$  имеет место равенство  $\varphi(ab) = \varphi(a) + \varphi(b)$ .

Для данных целых  $a$  и  $b$  ( $1 < a \leq b < 10$ ) вычислить по алгоритму прямоугольников  $\varphi(a)$ ,  $\varphi(b)$ ,  $\varphi(ab)$  и  $\varphi(a) + \varphi(b) - \varphi(ab)$ . В качестве шага приближенного интегрирования взять 0.1.

177. В тексте параграфа были приведены соображения в пользу того, что площадь криволинейной трапеции, ограниченной графиком функции, осью абсцисс и двумя вертикальными прямыми равна пути, пройденному вдоль прямой некоторым телом, для которого зависимость скорости от времени описывается данной функцией. Привести аналогичные соображения в пользу того, что площадь такого рода криволинейной трапеции равна работе силы

по перемещению тела вдоль координатной прямой из положения  $x=a$  в положение  $x=b$ , если зависимость силы от координаты точки есть данная функция. То есть работа силы  $F(x)$  есть

$$\int_a^b F(x) dx.$$

178. Шар радиуса  $r$  плавает на поверхности воды, погруженный в нее на глубину  $\alpha r$  ( $\alpha < 2$ ). Требуется вычислить работу, которую надо совершить для того, чтобы погрузить шар под воду целиком. Написать выражение для этой работы в виде интеграла. Написать программу вычисления этой работы по данным  $\alpha$  и  $r$  (см. предыдущую задачу и указание к задаче 100).

179. Задача аналогична предыдущей, но требуется вычислить работу, которую надо совершить для того, чтобы вытащить шар из воды.

180. Решить задачу 178, считая, что дана плотность вещества  $\rho$  ( $\rho < 1$ ), из которого изготовлен шар и радиус шара  $r$  (значение  $\alpha$  не известно).

### § 13. Оператор перехода. Пустой оператор

Каждый оператор может быть помечен меткой — любым целым числом без знака, содержащим не более 4 цифр. Метка располагается перед оператором и отделяется от него двоеточием. Оператор `write(x)`, помеченный меткой 21, запишется так:

21: `write(x)`

Метка не влияет на выполнение оператора.

В рассмотренных программах операторы выполнялись в том порядке, в каком были написаны. Изменить этот порядок можно с помощью *оператора перехода*. Оператор перехода состоит из служебного слова `goto` \*), за которым следует метка, например:

`goto 17`

`goto 0`

Оператор перехода прерывает естественную последовательность выполнения операторов: следом за ним выполняется оператор, помеченный указанной меткой.

Пусть программа содержит последовательность операторов:

`x:=2; a:=b; goto 99;`

`14: a:=0; x:=b; 99: y:=x; write(x)`

В этом случае сначала выполняются операторы `x:=2` и `a:=b`, затем следует переход к оператору, помеченному меткой 99, т. е.

---

\*) Go to — перейти к.

к оператору  $y := x$ . После оператора  $y := x$  будет выполнен оператор  $write(x)$ .

Все метки должны быть описаны. Описание меток состоит из служебного слова *label* \*) и следующего за ним списка меток, который завершается точкой с запятой. Например,

*label* 17, 0, 14;

Описание меток располагается до совокупности всех описаний переменных. Описанной меткой должен быть помечен ровно один оператор программы.

О пустом операторе. Пустой оператор не предписывает никаких действий. По определению он представляет собой пустую совокупность символов. Как и все операторы, пустой оператор может быть помечен меткой.

Рассмотрим конец некоторой программы или составного оператора:

... 10: *end*. или ... 10: *end*

Здесь перед *end* расположен помеченный меткой 10 пустой оператор. Основное назначение пустого оператора — дать возможность выхода из середины программы или составного оператора.

В результате выполнения следующей программы выясняется, имеются или не имеются среди чисел  $\cos(i^3) \sin(in)$  ( $i=1, \dots, n$ ) меньшие 0.0001. Если имеются, то выводится *есть*, если нет — *нет*.

```
program П1 (input, output);  
  label 1;  
  var i, n: integer;  
  begin read(n);  
    for i:=1 to n do  
      if  $\cos(i*i*i)*\sin(i*n) < 0.0001$  then  
        begin  
          write('есть'); goto 1  
        end;  
      write('нет');  
    1: end.
```

Если оказывается, что некоторое число меньше 0.0001, то следующие числа уже не рассматриваются. Программу можно было бы написать с двумя операторами перехода:

```
program П2 (input, output);  
  label 1, 2;  
  var i, n: integer;  
  begin read(n);
```

---

\*) Label — метка.



```

for i:=1 to n do
    if  $\cos(i*i)*\sin(i*n) < 0.0001$  then goto 1;
write('нет'); goto 2;
1: write('есть');
2: end.

```

Программа, в результате выполнения которой выясняется, имеется ли среди 20 целых данных чисел хотя бы одно положительное, кратное 5. Если да, то выводится первое из этих чисел, если нет — сообщение *не найдено*.

```

program ПЗ(input, output);
label 13, 12;
var k, a: integer;
begin for k:=1 to 20 do
    begin writeln('?'); read(a);
        if  $a \leq 0$  then goto 12;
        if  $a \bmod 5 = 0$  then
            begin write(a); goto 13 end;
    12: end;
    write('не найдено');
13: end.

```

С помощью оператора перехода, расположенного вне условного оператора или оператора цикла, нельзя перейти внутрь этого условного оператора или оператора цикла.

## ЗАДАЧИ

181. Может ли в программе встретиться оператор перехода:

- а) `goto 0`, б) `goto 99999`,  
 в) `goto x:=x+1`, г) `goto -1001`?

182. Может ли программа содержать в качестве описания меток:

- а) `label 1; 2; 3;` б) `label 2, 1;` в) `label 1;`  
 г) `label 2, -3, 4;` д) `label x;` е) `label 0, 1.2, 3;`

183. Записать оператор, предписывающий переход к оператору, помеченному меткой 17, если  $x > y$  и переход к оператору, помеченному меткой 6 в противном случае.

184. Можно ли указать программу, выполнение которой завершается и конец которой имеет вид:

- а) `; goto 1 end.` б) `goto 1 end.`  
 в) `goto 1; 1: end.` г) `goto 1; 2: end.?`

185. Даны натуральные  $n, a, b$ . Выяснить, есть ли среди чисел  $i^3 - 17in^2 + n^3$  ( $i=1, \dots, n$ ) хотя бы одно, которое кратно  $a$  и не кратно  $b$ .

186. Дано: натуральное  $n$ , целые  $a_1, \dots, a_n$ . Выяснить, имеется ли в последовательности  $a_1, \dots, a_n$  хотя бы одно нечетное отрицательное число.

187. Выяснить, есть или нет среди 15 данных целых чисел полные квадраты. Если есть, то должно быть выведено одно такое число, если нет, то должен быть выведен текст *квадратов нет*.

188. Дано: натуральное  $n$ , действительные  $x_1, y_1, \dots, x_n, y_n$ . Выяснить, верно ли, что среди точек  $(x_i, y_i)$ ,  $i=1, \dots, n$  есть хотя бы одна, принадлежащая квадрату, стороны которого параллельны координатным осям, центр совпадает с началом координат, а длина стороны равна 1.

189. Даны действительные  $x_1, y_1, x_2, y_2, \dots, x_{10}, y_{10}$ . Выяснить, верно ли, что все точки  $(x_i, y_i)$  ( $i=1, \dots, 10$ ) принадлежат кругу радиуса 2 с центром в точке  $(1, 1)$ .

190. Дано: натуральное  $n$ , целые  $a_1, \dots, a_n$ . Выяснить, какая из трех ситуаций имеет место: все числа  $a_1, \dots, a_n$  равны нулю, в последовательности  $a_1, \dots, a_n$  первое ненулевое число — положительное, первое ненулевое число — отрицательное.

191. Дано: натуральное  $n$ , действительные  $a_1, \dots, a_n$ . Выяснить, верно ли, что для всех чисел  $a_1, \dots, a_n$  выполнены неравенства  $i+1 < a_i < i!$

192. Даны целые  $a, n, x_1, \dots, x_n$  ( $n > 0$ ). Определить, каким по счету идет в последовательности  $x_1, \dots, x_n$  член равный  $a$ . Если такого члена нет, то ответом должно быть число 0.

193. Дано: натуральное  $n$ , действительные  $a_1, \dots, a_n$ .

а) Выяснить, верно ли, что  $a_1 < a_2 < \dots < a_n$ .

б) Получить число, равное  $n+1$ , коль скоро  $a_1 < a_2 < \dots < a_n$ , и равное значению  $m$ , для которого  $a_1 < \dots < a_m$  и  $a_m \geq a_{m+1}$  в противном случае.

194. Дано: натуральное  $n$ , действительные  $a_1, \dots, a_n$ . Выяснить, верно ли, что наибольший член последовательности  $a_1, \dots, \dots, a_n$  по модулю больше единицы.

195. Даны целые  $c_1, \dots, c_{25}$ . Выяснить, имеются ли в последовательности  $c_1, \dots, c_{25}$  два идущих подряд нулевых члена.

## § 14. Вложенные операторы цикла

Интересна конструкция, в которой оператор, расположенный после *do*, сам является оператором цикла или содержит в себе оператор цикла.

Пусть дано натуральное  $n$  и требуется вычислить сумму степеней  $\left(\frac{1}{1}\right)^n + \left(\frac{1}{2}\right)^n + \dots + \left(\frac{1}{n}\right)^n$ . Схема программы может быть взята такой:

```

program суммa(input, output);
  var n, i ...: integer; a, s, ...: real;
  begin read(n); s := 0;
    for i:= 1 to n do
      begin a:= 1/i;
        вычислить  $a^n$  и результат прибавить к s
      end;
    write('сумма степеней=', s)
  end.

```

Вычисление  $a^n$  и прибавление результата к  $s$  может быть описано, например, так:

```

p:= a; for j:= 2 to n do p:= p*a; s:= s+p

```

Подставив эти три оператора в приведенную выше схему и включив переменную  $p$  в описание переменных типа *real*, а переменную  $j$  — в описание переменных типа *integer*, получим программу

```

program суммa(input, output);
  var n, i, j: integer; a, s, p: real;
  begin read(n); s := 0;
    for i:= 1 to n do
      begin a := 1/i; p := a;
        for j:= 2 to n do p := p*a; s := s+p
      end;
    write ('сумма степеней =', s)
  end.

```

Напишем теперь программу получения всех совершенных чисел меньших данного  $n$  (натуральное число называется совершенным, если оно равно сумме всех своих делителей, исключая себя самого; так число 6 — совершенное, поскольку его делителями служат 1, 2, 3, 6, и при этом  $1+2+3=6$  \*). Воспользуемся следующей схемой:

```

program совершенные(input, output);
  var n, i, m, ...: integer;
  begin read(n); writeln('┐');
    for i:= 2 to n do
      begin вычислить m-сумму всех делителей числа i из
        диапазона от 1 до i-1;
      if m=i then writeln(i)
      end
    end.

```

---

\*) Предположение о бесконечности множества всех совершенных чисел до сих пор не доказано и не опровергнуто. К настоящему времени найдено двадцать четыре совершенных числа, все эти числа — четные. Существуют ли нечетные совершенные числа — неизвестно.

и займемся подробным описанием процесса вычисления суммы делителей. Сразу можно дать следующее решение:

```
m := 0; for j := 1 to i-1 do
    if i mod j = 0 then m := m + j
```

Однако  $i$  заведомо делится на 1, поэтому первая проверка делимости лишняя, и здесь лучше написать так:

```
m := 1; for j := 2 to i-1 do
    if i mod j = 0 then m := m + j
```

Можно и еще сократить количество проверок на делимость, воспользовавшись тем, что среди чисел, больших  $[i/2]$  и меньших  $i-1$ , заведомо нет делителей  $i$  (например, у числа 100 нет делителей  $j$  таких, что  $51 \leq j \leq 99$ ):

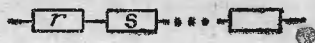
```
m := 1; for j := 2 to i div 2 do
    if i mod j = 0 then m := m + j
```

Подставив эти два оператора в выписанную ранее схему и включив переменную  $j$  в описание переменных, получим программу

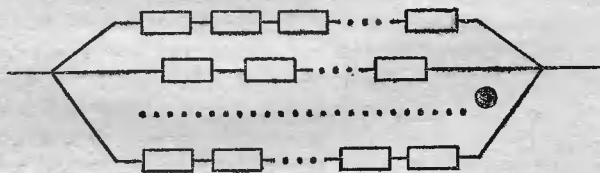
```
program совершенные(input, output);
var n, i, m, j: integer;
begin read(n); writeln('┐');
    for i := 2 to n do
        begin m := 1;
            for j := 2 to i div 2 do
                if i mod j = 0 then m := m + j;
            if m = i then writeln(i)
        end
    end.
```

Приведем пример вложенности операторов цикла, имеющих вид *while B do...* (операторов цикла с условием). Рассмотрим задачу из электротехники. Группа последовательно соединенных сопротивлений, изображенная на рис. 40а, задается числами  $r, s, \dots$  — величинами сопротивлений. Параллельное соединение ряда таких групп, показанное на рис. 40б, задается так: сначала идут величины сопротивлений, входящих в первую группу, затем — некоторое отрицательное число, затем — величины сопротивлений, входящих во вторую группу, затем — некоторое отрицательное число и т. д. После величины последнего сопротивления последней группы идут два отрицательных числа. Требуется рассчитать сопротивление соединения. Здесь естественно использовать такой оператор цикла, в ходе выполнения которого одна из переменных программы, скажем, переменная  $r$ , поочередно принимает значения величин первых сопротивлений различных групп: вна-

чае это значение — самое первое число, затем — число, следующее за первым отрицательным числом, затем — число, следующее за вторым отрицательным числом и т. д. Выполнение оператора цикла завершится, как только это значение само окажется отрицательным числом. Для каждого значения переменной  $r$  с помощью другого оператора цикла выполняются следующие действия: вводятся



$\alpha$



$\delta$

Рис. 40

все числа, начиная с того, которое расположено вслед за  $r$ , и кончая следующим очередным отрицательным числом; попутно вычисляется сумма всех этих чисел, исключая отрицательное число. Более подробно это записывается в виде схемы

```

program conp(input, output);
  var r, s, u, v...: real;
  begin u := 0; read(r);
    while r > 0 do
      begin
        поочередно ввести все числа вплоть до первого
        отрицательного, попутно вычислить v — сумму r
        и всех этих чисел, кроме отрицательного;
        u := u + 1/v; read(r)
      end;
      write('сопротивление = ', 1/u)
    end.
  
```

Тот участок схемы программы *conp*, который записан словами, допускает, очевидно, следующую детализацию в виде операторов Паскаля:

```

  v := r; read(s);
  while s > 0 do
    begin v := v + s; read(s) end
  
```

Теперь можно написать программу в окончательном виде:

```

program comp(input, output);
  var r, s, u, v: real;
  begin u := 0; read(r);
    while r > 0 do
      begin v := r; read(s);
        while s > 0 do
          begin v := v + s; read(s) end;
          u := u + 1/v; read(r)
        end;
      ~write('compотивление = ', 1/u)
    end.
end.

```

В этой программе мы не использовали вывод приглашений и другие способы облегчения работы с программой. Добавление многочисленных операторов вывода затруднило бы понимание самой программы, структура которой достаточно сложна. Мы и впредь, как правило, будем опускать эти добавочные операторы вывода. Их расстановка предоставляется читателю в качестве самостоятельного упражнения.

Следует отметить, что иногда в пределах одной программы встречаются операторы цикла разных видов — с условием и с параметром, и при этом один оператор вложен в другой. Разберем пример. Будем рассматривать равенство  $y^3 + y^2 - x^2 = 0$  как уравнение, зависящее от параметра, и будем считать  $x$  параметром, а  $y$  — неизвестной. Более подробно: если  $x$  имеет некоторое значение, то, подставляя это значение в  $y^3 + y^2 - x^2 = 0$ , получим конкретное уравнение относительно  $y$ . Пусть, например,  $x = 1$ . Тогда получится  $y^3 + y^2 - 1 = 0$ . Пусть, далее,  $x = 1.1$ , тогда получится  $y^3 + y^2 - 1.21 = 0$  и т. д. Пусть для значений  $x$ , равных 1, 1.1, 1.2, ..., 2 требуется найти значения  $y$ , удовлетворяющие тем уравнениям, которые — получаются в результате подстановки в  $y^3 + y^2 - x^2 = 0$  значения  $x$ . Каждое такое уравнение имеет единственный корень, этот корень принадлежит отрезку  $[0, 2]$  и может быть найден алгоритмом деления отрезка пополам (доказательство мы опускаем). Напишем программу получения последовательности корней. Каждый корень будет вычисляться с точностью 0.001. Вначале составим схему программы. Для последовательного перебора значений 1, 1.1, 1.2, ..., 2 будем привлекать оператор цикла

```
for i := 0 to 10 do ...
```

в теле которого в качестве очередного значения  $x$  используется значение выражения  $1 + 0.1 * i$ :

```

program nk(output);
  var i, ...: integer; x, u, a, ...: real;

```

```

begin
  for i := 0 to 10 do
    begin x := 1 + 0.1*i; u := sgr(x);
      найти a-значение корня уравнения  $y^3 + y^2 - u = 0$ ,
      такое, что  $0 \leq a \leq 2$ ;
      writeln(a)
    end
  end.

```

Для детализации этой схемы воспользуемся схемой программы *делл* из § 7. Включим необходимые переменные в описания переменных типа *real*, и попутно исключим из рассмотрения переменную *x*, так как сразу можно положить  $u := \text{sgr}(1 + 0.1*i)$ :

```

program nk(output);
  var i: integer; u, a, b, c, fa, fc: real;
  begin
    for i := 0 to 10 do
      begin u := sgr(1 + 0.1*i);
        a := 0; b := 2; fa := u;
        while b - a >= 0.0001 do
          begin c := (a + b)/2;
            fc := sgr(c)*(c + 1) - u;
            if fa*fc <= 0 then b := c
            else begin a := c; fa := fc end
          end;
          writeln(a)
        end
      end
    end.

```

Итак, значениям *x*, принадлежащим отрезку  $[1, 2]$ , мы сопоставим значения *y*, определив, таким образом, *y* как функцию *x* на отрезке  $[1, 2]$ . Приведенная программа позволяет вычислить значения этой функции для ряда значений аргумента *x*. Отметим, что программа не содержит ни одного оператора ввода, поэтому ее заголовок не содержит идентификатора *input*.

## ЗАДАЧИ

196. Изменить программу *сумст* так, чтобы вычислялась:

- а) последовательность  $\left(\frac{1}{1}\right)^n, \left(\frac{1}{2}\right)^n, \dots, \left(\frac{1}{n}\right)^n$ ,
- б) последовательность  $\left(\frac{1}{1}\right)^1, \left(\frac{1}{2}\right)^2, \dots, \left(\frac{1}{n}\right)^n$ ,
- в) сумма  $\left(\frac{1}{1}\right)^1 + \left(\frac{1}{2}\right)^2 + \dots + \left(\frac{1}{n}\right)^n$ ,

г) сумма  $\left(\frac{1}{1}\right)^n + \left(\frac{1}{2}\right)^{n-1} + \dots + \left(\frac{1}{n}\right)^1$ .

197. Дано натуральное  $n$ . Получить:

а)  $\left(1 + \frac{1}{1^n}\right) \left(1 + \frac{1}{2^n}\right) \dots \left(1 + \frac{1}{n^n}\right)$ ,

б)  $\left(1 + \frac{1}{1^1}\right) \left(1 + \frac{1}{2^2}\right) \dots \left(1 + \frac{1}{n^n}\right)$ ,

в)  $\left(1 + \frac{1}{1}\right)^n \left(1 + \frac{1}{2}\right)^n \dots \left(1 + \frac{1}{n}\right)^n$ ,

г)  $\left(1 + \frac{1}{1}\right)^1 \left(1 + \frac{1}{2}\right)^2 \dots \left(1 + \frac{1}{n}\right)^n$ ,

д)  $\left(1 + \frac{1}{1}\right)^n \left(1 + \frac{1}{2}\right)^{n-1} \dots \left(1 + \frac{1}{n}\right)^1$ .

198. Дано натуральное  $n$ . Получить  $f_0 f_1 \dots f_n$ , где  $f_i =$

$$= \frac{1}{i^2+1} + \frac{1}{i^2+2} + \dots + \frac{1}{i^2+i+1}, \quad i=0, 1, \dots, n.$$

199. Дано: натуральное  $n$ , действительные  $a_1, \dots, a_n$ . Вычислить:

а)  $a_1^n + a_2^n + \dots + a_n^n$ ,

б)  $a_1^1 + a_2^2 + \dots + a_n^n$ ,

в)  $a_1 - a_2^2 + \dots + (-1)^{n+1} a_n^n$ ,

г)  $a_1 + a_2(a_2-1) + \dots + a_n(a_n-1) \dots (a_n-n+1)$ .

200. Рассмотрим последовательность  $e_1, e_2, e_3, \dots$ , образованную по следующему закону:  $e_k = \left(1 + \frac{1}{k}\right)^k$  ( $k=1, 2, \dots$ ). Дано действительное  $\varepsilon$ . Найти первый член  $e_n$  последовательности  $e_1, e_2, \dots$ , для которого  $n \geq 2$  и  $|e_n - e_{n-1}| < \varepsilon$ . (Существование  $e_n$  гарантируется одной из теорем математического анализа; чем меньше  $\varepsilon$ , тем ближе  $e_n$  к числу  $e=2.718281828\dots$ )

201. Для каждого положительного  $a$  найдется натуральное  $m$  такое, что  $1 + \frac{1}{2} + \dots + \frac{1}{m} > a$  (см. задачу 82). Определим последовательность  $k_1, k_2, \dots$  так:  $k_i$  — это наименьшее натуральное число  $i$  такое, что  $1 + \frac{1}{2} + \dots + \frac{1}{i} > i$  ( $i=1, 2, \dots$ ). Непосредственно проверяется, что  $k_1=2$  (так как  $i=1$  и  $1 + \frac{1}{2} > 1$ ) и  $k_2=4$  (так как  $1 + \frac{1}{2} + \frac{1}{3} < 2$  и  $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} > 2$ ). Дано натуральное  $n$ . Вывести таблицу значений  $k_1, k_2, \dots$ . Таблица имеет две колонки, в  $i$ -ю строку таблицы заносятся значения  $i$  и  $k_i$ ,  $i=1, 2, \dots, n$ .

202. Дано: натуральное  $n$ , действительное неотрицательное  $u$ . Используя последовательность  $x_1, x_2, x_3, \dots$ , из задачи 87, § 6,



найти приближенное значение  $\sqrt[n]{u}$ . Нужное приближение  $x_k$  считается полученным, если  $k \geq 2$  и  $|x_k - x_{k-1}| < 0.0001$ .

203. Даны натуральные  $n, a_1, \dots, a_n$ . Вычислить НОД  $(a_1, a_2, \dots, a_n)$ . Воспользуемся алгоритмом Евклида (программа E из § 9) и следующим свойством наибольшего общего делителя: если  $\text{НОД}(a_1, \dots, a_{k-1}) = d$ , то  $\text{НОД}(a_1, \dots, a_k) = \text{НОД}(d, a_k)$ .

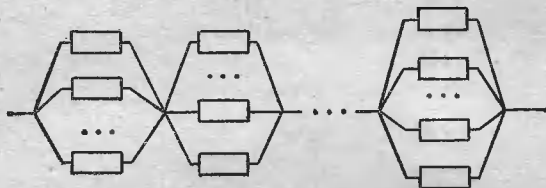


Рис. 41

204. Рассчитать сопротивление последовательного соединения параллельных групп сопротивлений (рис. 41). Считая, что каждая группа задается рядом величин сопротивлений, за которым идет отрицательное число, а вслед за величиной последнего сопротивления последней группы идут два отрицательных числа.

205. Прямоугольное хоккейное поле размера  $a \times b$  освещено  $n$  рядами ламп по  $m$  ламп в ряду, расположенными на высоте  $h$  от поверхности льда. Расстояние между рядами ламп равно  $\frac{a}{n-1}$ ,

расстояние между лампами в ряду  $\frac{b}{m-1}$ . Определить освещенность хоккейного поля в точке, расстояния от которой до бортов соответственно равны  $a_1, b_1$  ( $a_1 \leq a, b_1 \leq b$ ). Мощность ламп—200 Вт, КПД ламп—1%.

206. Дано: натуральные  $k, n$ , действительные  $a_1, \dots, a_{kn}$ . Получить:

а) последовательность  $a_1 + \dots + a_k, a_{k+1} + \dots + a_{2k}, \dots, a_{k(n-1)+1} + \dots + a_{kn}$ ;

б) последовательность  $\max(a_1, \dots, a_k), \max(a_{k+1}, \dots, a_{2k}), \dots, \max(a_{k(n-1)+1}, \dots, a_{kn})$ ;

в)  $\min(a_1, \dots, a_k) + \min(a_{k+1}, \dots, a_{2k}) + \dots + \min(a_{k(n-1)+1}, \dots, a_{kn})$ .

Указание. В заданиях а) и б) предполагается, что при выполнении программы в нужные моменты будут выводиться сообщения  
сумма последних  $k$  чисел равна ...

и

наибольшее из  $k$  последних чисел равно ...

с конкретными числовыми значениями вместо многоточий.

207. Даны натуральные  $n, m$ . Получить все меньшие  $n$  натуральные числа, квадрат суммы цифр которых равен  $m$ .

208. Дано натуральное  $n$ . Можно ли  $n$  представить в виде суммы трех квадратов натуральных чисел? Если можно, то:

а) указать тройку  $x, y, z$  таких натуральных чисел, что  $x^2 + y^2 + z^2 = n$ ;

б) указать все тройки  $x, y, z$  таких натуральных чисел, что  $x^2 + y^2 + z^2 = n$ .

209. Дано натуральное  $n \geq 2$ . Разложить число  $n$  на простые множители. Простой множитель  $p$  должен быть выведен  $\alpha$  раз, где  $\alpha$  — натуральное число такое, что  $n$  делится на  $p^\alpha$  и не делится на  $p^{\alpha+1}$ .

Указание. Воспользоваться тем, что если  $n$  не делится на 2, 3, ...,  $k-1$ , но делится на  $k$ , то  $k$  — простое число (если бы  $k$  было составным, то оно делилось бы на какое-нибудь число из 2, 3, ...,  $k-1$ , но тогда бы и  $n$  делилось на это число). Использовать операторы следующего вида:

```
while p < n do
  begin
    while n mod p = 0 do begin ... end;
    p := p + 1
  end
  . . . . .
```

210. В условия предыдущей задачи вносится изменение: каждый простой множитель данного числа  $n$  должен быть выведен ровно один раз.

## § 15. Составные условия

В условных операторах

```
if B then P else Q
if B then P
```

и в операторе цикла

```
while B do P
```

в качестве условия  $B$  мы до сих пор использовали только отношения равенства и неравенства:  $a = b + 1$ ,  $n \geq 0$  и т. д. Из простых условий в Паскале разрешается строить более сложные. Для этих построений существуют определенные правила. Прежде всего, требуется, чтобы все отношения, которые используются при построении, заключались в скобки:  $(a = b + 1)$ ,  $(n \geq 0)$ ,  $(s > t)$ ,  $(t = s + 12)$  и т. д. Далее, пусть  $A$  и  $B$  — некоторые условия, тогда

$A \text{ and } B^*)$  — это новое условие, оно соблюдается тогда и только тогда, когда соблюдаются оба условия  $A$  и  $B$ ; в свою очередь  $A \text{ or } B^{**})$  — это новое условие, которое соблюдается тогда и только тогда, когда соблюдается хотя бы одно из условий  $A$  и  $B$ . Если

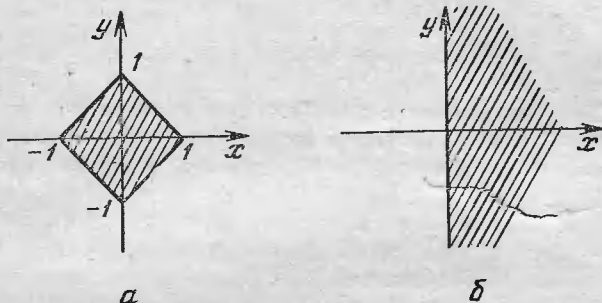


Рис. 42

$C$  — условие, то  $\text{not } C^{***})$  — это новое условие, которое соблюдается тогда и только тогда, когда не соблюдается  $C$ .

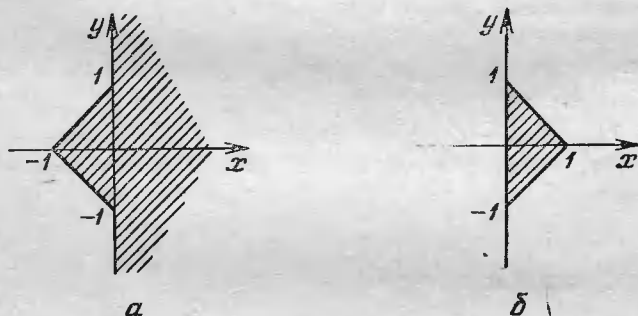


Рис. 43

Операции над условиями *and*, *or*, *not* называются логическими операциями.

Рассмотрим условия

$$(\text{abs}(x) + \text{abs}(y) \leq 1) \text{ or } (x > 0)$$

и

$$(\text{abs}(x) + \text{abs}(y) \leq 1) \text{ and } (x > 0)$$

Первое из этих условий соблюдается, если, например,  $x=0, y=0$ ; это же условие не соблюдается, если  $x=-2, y=3$ . Второе из этих

\*) And — и.

\*\*) Or — или.

\*\*\*) Not — не.

условий соблюдается, если, например,  $x=0.5$ ,  $y=0.25$ ; это же условие не соблюдается, если например,  $x=3$ ,  $y=3$ . На рис. 42 в заштрихованные области попали все точки, значения координат которых таковы, что соблюдаются условия  $abs(x) + abs(y) \leq 1$  и  $x \geq 0$  соответственно. Далее, на рис. 43а, б, в заштрихованные области попали все точки, координаты которых таковы, что соблюдаются условия  $(abs(x) + abs(y) \leq 1) \text{ or } (x \geq 0)$  и  $(abs(x) + abs(y) \leq 1) \text{ and } (x \geq 0)$  соответственно. Наконец, на рис. 44 в заштрихованную область попали все точки, для координат которых соблюдается условие  $not ((abs(x) + abs(y) \leq 1) \text{ and } (x \geq 0))$ , граница в последнем случае не входит в область.

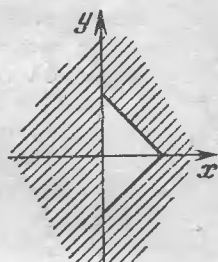


Рис. 44

При определении того, соблюдается условие или нет, действуют следующие правила старшинства логических операций: самая старшая операция — это *not*, следующая — *and*, потом — *or*. Первой из двух одинаковых операций выполняется та, знак которой в условии встречается раньше. Скобки могут изменить этот порядок. Для условия

$$(y > z) \text{ and } not ((x > 0) \text{ or } (z > x)) \text{ or } (x > y)$$

устанавливается следующий порядок выполнения логических операций:

$$(y > z) \text{ and } not ((x > 0) \text{ or } (z > x)) \text{ or } (x > y)$$

При  $x=-1$ ,  $z=-2$ ,  $y=1$  имеем

$$\underbrace{(y > z)}_{\text{соблюдается}} \text{ and } not \left( \underbrace{((x > 0) \text{ or } (z > x))}_{\text{не соблюдается}} \right) \text{ or } \underbrace{(x > y)}_{\text{не соблюдается}}$$

$\underbrace{\hspace{10em}}_{\text{не соблюдается}}$   
 $\underbrace{\hspace{10em}}_{\text{соблюдается}}$   
 $\underbrace{\hspace{10em}}_{\text{соблюдается}}$   
 $\underbrace{\hspace{10em}}_{\text{соблюдается}}$

Такого рода условия используются в условных операторах и операторах цикла. Рассмотрим программу, в результате выполнения которой выясняется, принадлежит ли число  $x$  отрезку  $[a, b]$ . Первый вариант:

```
program отрезок1(input, output);
  var x, a, b: real;
  begin read(x, a, b);
```

```

if ( $x \geq a$ ) and ( $x \leq b$ )
then write('принадлежим')
else write('не принадлежим')

```

end.

Второй вариант:

```

program отрезок2(input, output);
var x, a, b: real;
begin read(x, a, b);
if ( $x < a$ ) or ( $x > b$ ) then write('не ');
write('принадлежим')
end.

```

Если условие содержит хотя бы одну логическую операцию, то оно называется составным условием (в отличие от условий, являющихся отношениями). В последних двух программах составные условия позволили избежать громоздких вложений одного условного оператора в другой.

Пусть теперь  $x_0 = y_0 = 10$  и  $x_i = 0.1y_{i-1}$ ,  $y_i = 0.1x_i - 0.12y_i$  ( $i = 1, 2, \dots$ ). Пусть  $\varepsilon > 0$ . Требуется найти наименьшее  $i$  такое, что  $|x_i| < \varepsilon$ ,  $|y_i| < \varepsilon$ .

Программа:

```

program I (input, output);
var x, y, xx, eps: real; i: integer;
begin read(eps); x := 10; y := 10; i := 0;
while (abs(x) >= eps) or (abs(y) >= eps) do
begin
xx := 0.1 * y; y := 0.1 * x - 0.12 * y;
x := xx; i := i + 1
end;
write(i)
end.

```

Вернемся теперь к задаче 70. Напомним ее условие. Даны положительные  $a, b, c, d$ . Выяснить, можно ли прямоугольник со сторонами  $a, b$  уместить внутри прямоугольника со сторонами  $c, d$  так, чтобы каждая из сторон одного прямоугольника была параллельна или перпендикулярна каждой из сторон второго прямоугольника.

Программа:

```

program Пр (input, output);
var a, b, c, d: real;
begin read(a, b, c, d);
if ( $a \leq c$ ) and ( $b \leq d$ ) or ( $a \leq d$ ) and ( $b \leq c$ )
then write('можно')
else write('нельзя')
end.

```

В этой программе использовано составное условие:

$$(a \leq c) \text{ and } (b \leq d) \text{ or } (a \leq d) \text{ and } (b \leq c)$$

Если это облегчает проверку программы, то при записи такого рода условий можно употреблять дополнительные скобки. В последней программе можно было бы написать

$$\text{if } ((a \leq c) \text{ and } (b \leq d)) \text{ or } ((a \leq d) \text{ and } (b \leq c)) \text{ then } \dots$$

## ЗАДАЧИ

211. Можно ли между *if* и *then* или между *while* и *do* разместить условие, записанное следующим образом:

- а)  $(x < y) \text{ and } (y < z)$ ,
- б)  $x < y$ ,
- в)  $x < z \text{ or } z > y$ ,
- г)  $\text{not } (x > 1) \text{ or } (x \leq y) \text{ and } (x < z)$ ,
- д)  $\text{not } ((x > 1) \text{ or } (x \leq y)) \text{ and } (x < z)$ ,
- е)  $\text{not } (x > 1) \text{ or } (x \leq y) \text{ and } (x < z)$ ,
- ж)  $\text{not } ((x > 1) \text{ or } (x \leq y) \text{ and } (x < z))$ ?

В случае утвердительного ответа указать порядок выполнения логических операций.

212. Для каких значений переменной  $x$  типа *real* соблюдается условие:

- а)  $(\text{abs}(x) > 2) \text{ and } (\text{abs}(x) < 5)$ ,
- б)  $(\text{abs}(x) > 2) \text{ or } (\text{abs}(x) < 5)$ .

213. Сколько раз будет выполнен оператор  $n := n + 1$  в ходе выполнения оператора цикла *while B do*  $n := n + 1$ , если  $n$  — это переменная типа *integer*, а  $B$  — это:

- а)  $(\text{sqr}(n) > 0) \text{ or } (n = 0)$ ,
- б)  $(n > 0) \text{ and } (n < 0)$ ,
- в)  $(n < 0) \text{ or } (n = 0) \text{ or } (n > 0)$ .

214. Дано действительное  $x$ . Вычислить

$$y = \begin{cases} x, & \text{если } x \geq 1 \text{ или } x < -3, \\ x^2 + 2x - 2 & \text{в противном случае.} \end{cases}$$

215. Даны действительные  $x, y$ . Определить, принадлежит ли точка с координатами  $x, y$  части плоскости, изображенной на рис. 45а—г.

216. Дано натуральное  $n$ . Получить сумму тех из чисел вида  $i^3 - 3in^2 + n$  ( $i = 1, \dots, n$ ), которые являются утроенными нечетными.

217. Дано: натуральное  $n$ , целые  $a_1, \dots, a_n$ . Найти количество и сумму тех членов последовательности  $a_1, \dots, a_n$ , которые делятся на 5 и не делятся на 7.

218. Дано: натуральное  $n$ , действительные  $a_1, \dots, a_n$ . Получить число отрицательных членов последовательности  $a_1, \dots, a_n$  и число членов, принадлежащих отрезку  $[1, 2]$ .

219. Рассматривается последовательность  $a_1, \dots, a_{1000}$ . Определить, сколько из членов последовательности с номерами 1, 2, 4, 8, 16, ... имеют значения меньше, чем 0.25. При этом считать, что:

а)  $a_i = \sin^2(3i + 5) - \cos(i^2 - 15)$  ( $i = 1, 2, \dots, 1000$ ),

б)  $a_1, \dots, a_{1000}$  — данные действительные числа,

в)  $a_1 = 0.01$ ;  $a_i = \sin(i + a_{i-1})$  ( $i = 2, \dots, 1000$ ).

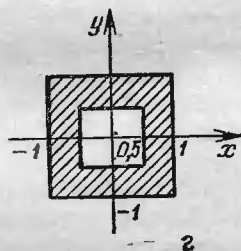
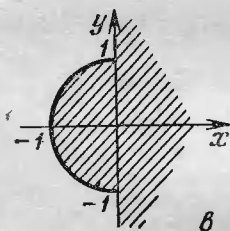
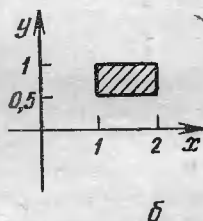
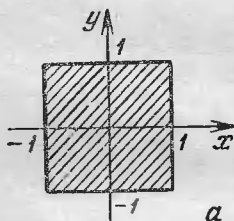


Рис. 45

220. Дано: натуральное  $n$ , действительные  $a_1, \dots, a_n$ . Найти в последовательности  $a_1, \dots, a_n$  все пары  $a_i, a_{i+1}$ ,  $1 \leq i \leq n-1$ , такие, что  $a_i \leq 3$  и  $a_{i+1}$  либо больше 4.312, либо меньше 0. Среди значений первых элементов всех таких пар найти наибольшее.

*Указание.* В нужные моменты должен выводиться текст *последняя пара удовлетворяет поставленному условию*.

221. Даны целые  $n, a_1, \dots, a_n$  ( $n > 0$ ). Найти в последовательности  $a_1, \dots, a_n$  все тройки  $a_i, a_{i+1}, a_{i+2}$ ,  $1 \leq i \leq n-2$  такие, что  $a_i = 0$ ,  $a_{i+1}$  кратно 2,  $a_{i+2}$  кратно 7. Вычислить количество таких троек.

*Указание.* В нужные моменты должен выводиться текст *в последней тройке первое число равно 0, второе кратно 2, третье кратно 7*.

222. Даны действительные  $a, b$  ( $a < b$ ). Найти первый член последовательности  $a_i = (-1)^i \left( 1 + \frac{1}{2} + \dots + \frac{1}{i} \right)$  ( $i = 1, 2, \dots$ ), который не принадлежит отрезку  $[a, b]$ .

223. Дано: натуральное  $n$ , действительные  $a_1, \dots, a_n$ . Выяснить, образуют ли возрастающую последовательность числа:

а)  $a_1, \dots, a_n, 2a_1, 3a_2, \dots, (n+1)a_n$ ,

б)  $a_1, \dots, a_n, a_n+1, a_{n-1}+2, \dots, a_1+n$ ,

в)  $a_1, \dots, a_n, n(a_{n-1}+1), (n-1)(a_{n-2}+2), \dots, 2(a_1+n-1)$ .

224. Пусть  $A$  и  $B$  — некоторые условия. Доказать, что условие  $\text{not } (A \text{ and } B)$  соблюдается тогда и только тогда, когда соблюдается условие  $\text{not } A \text{ or not } B$ , а условие  $\text{not } (A \text{ or } B)$  соблюдается тогда и только тогда, когда соблюдается условие  $\text{not } A \text{ and not } B$ .

225. Пусть  $P$  — условие, которое соблюдается при любых значениях входящих в него переменных, например  $x+1 > x, \text{ sqr}(x) + \text{sqr}(y) > 0, 2*2=4$  и т. д. Пусть, далее,  $Q$  — условие, которое не соблюдается ни при каких значениях входящих в него переменных, например  $\text{abs}(x) < 0, y+2 < y, 1=0$  и т. д. Доказать, что для любого условия  $R$  составные условия  $P \text{ and } R$  и  $Q \text{ or } R$  соблюдаются тогда и только тогда, когда соблюдается  $R$ . Что можно сказать о составных условиях  $P \text{ or } R$  и  $Q \text{ and } R$ ?

226. Дано шесть натуральных чисел  $a, b, l, h, x, y$ . Числа  $a, b, l, h$  определяют положение левой нижней вершины прямоугольника, ширину и высоту. Стороны прямоугольника параллельны сторонам экрана. Значения  $x, y$  определяют положение точки на экране. Если точка лежит внутри прямоугольника, то высветить прямоугольник и точку. В противном случае высветить данный прямоугольник и прямоугольник, получающийся из данного поворотом на  $90^\circ$  по часовой стрелке вокруг точки  $x, y$ .

## § 16. Тип *char*

До сих пор рассматривались только программы, предназначенные для обработки числовых данных. Обработка символьных (иначе: знаковых, литерных) данных становится возможной благодаря привлечению значений и переменных типа *char* \*). Значениями типа *char* служат все те символы, которые могут быть высвечены на экране: буквы, цифры, знаки операций, скобки, пробел и т. д. Исключение составляет штрих ', имеющий специальное назначение.

Если в программе имеется описание

$u, v: \text{char}$

то возможны, например, операторы присваивания  $u := 'a', u := v$ ,

\*) *Char* — сокращение от *character* — символ.



$v := '*'$  и т. д. Штрих ' — принятая в Паскале форма кавычки — употребляется всякий раз, когда значение типа *char* явно указывается в программе. Выполнение операторов

$u := 'b'; \text{write}(u)$

приводит к высвечиванию на экране символа *b*.

Пусть по-прежнему  $u$  и  $v$  — переменные типа *char*. В качестве условий после служебных слов *if* и *while* могут употребляться отношения ' $a' = u, u = v, v < 'b'$ ' и т. д., а также построенные на их основе составные условия.

Пример. Даны символы  $s_1, s_2, \dots$ . Известно, что символ  $s_1$  отличен от символа / и что среди  $s_2, s_3, \dots$  есть хотя бы один символ /. Пусть  $s_1, \dots, s_n$  — символы данной последовательности, предшествующие первому символу / ( $n$  заранее не известно). Программа подсчета восклицательных знаков среди  $s_1, \dots, s_n$ .

```
program воскл (input, output);
  var c: char; n: integer;
  begin read(c); n := 0;
    while c <> '/' do
      begin if c = '!' then n := n + 1;
        read(c)
      end;
    write(n)
  end.
```

Символы, подготовленные для ввода, не отделяются друг от друга никакими знаками. Если три символа будут вводиться с помощью *read(a, b, c)*, то можно будет набрать на клавиатуре

*чищц*

тогда значениями переменных  $a, b$  и  $c$  станут, соответственно символы *ч, ш* и *ц*.

При построении условий, располагающихся после *if* и *while*, можно использовать разнообразные отношения в множестве символов. Здесь возможен не только знак  $=$  и комбинация  $<>$  (аналог знака  $\neq$ ), но также и  $>, >=, <, <=$ , так как все множество символов считается упорядоченным. Упорядоченность получается так: из всех символов составлен список, и из двух символов меньше тот, который встречается в списке раньше. Оговорим, что малые латинские буквы идут друг за другом, не перемешиваясь с другими символами, в алфавитном порядке, т. е. список содержит фрагмент

$\dots a, b, \dots, z, \dots$

точно так же большие латинские буквы образуют фрагмент

$\dots A, B, \dots, Z, \dots$

и арабские цифры идут друг за другом, как обычно:

...0, 1, ..., 9...

Поэтому, например,  $'a' < 'c'$ ,  $'Y' > 'X'$ ,  $'3' > '1'$ . В остальном вопрос порядка решается по-разному на разных компьютерах.

Упорядоченность открывает возможность использования в программах операторов цикла с параметром, имеющим тип *char*. Параметр цикла, имеющий тип *char*, пробегает ряд символов в указанных границах. Выполнение оператора цикла

```
for c := 'a' to 'z' do write (c)
```

где *c* — переменная типа *char*, приводит к высвечиванию на экране всех малых букв латинского алфавита:

abcdefghijklmnopqrstuvwxyz

Выполнение оператора цикла

```
for c := 'z' downto 'a' do write (c)
```

приведет к высвечиванию этих же букв в обратном порядке:

zyxwvutsrqponmlkjihgfedcba

**Пример.** Программа вывода последовательностей букв:

a

ab

aba

. . . .

abc ... xyz

```
program aab (output);
```

```
var c, d: char;
```

```
begin
```

```
  for c := 'a' to 'z' do
```

```
    begin
```

```
      for d := 'a' to c do write(d);
```

```
      writeln(' ');
```

```
    end
```

```
end.
```

Программа не содержит оператор ввода, и в заголовке программы идентификатор *input* опущен.

Рассмотрим еще пример, в котором использована упорядоченность значений типа *char*. Программа, в результате выполнения которой выясняется, имеется ли хотя бы одна малая латинская буква среди символов, предшествующих первому символу / в последовательности данных символов:

```

program быка(input, output);
  label 1;
  var c: char;
  begin read(c);
    while c <> '!' do
      if ('a' <= c) and (c <= 'z') then
        begin write('ecmb'); go to 1 end
      else read(c);
    write('nem');
  1: end.

```

Переменные типа *char* удобны для написания диалоговых программ. В качестве простой иллюстрации представим себе, что человеку, имеющему на столе компьютер, необходимо довольно часто получать значения некоторой функции, скажем функции  $\sin \sqrt{x}$ , для разных значений  $x$ .

Пусть заранее неизвестно, сколько раз и для каких именно значений  $x$  потребуется вычислить значения функции. Можно составить такую программу вычисления значений функции для вводимых с клавиатуры значений  $x$ , которая после каждого вычисления будет готова к выводу нового значения  $x$  и, соответственно, к вычислению значения функции \*). Один из вариантов — это зацикленная программа:

```

program синкоп(input, output);
  label 1;
  var x: real;
  begin
    1: write('x='); read(x);
    writeln(' ', f(x)=' ', sin(sqrt(x))); goto 1
  end.

```

Когда эта программа выполняется, на экране формируется последовательность строк:

```

x=..., f(x)=...
x=..., f(x)=...
x=..., f(x)=...
. . . . .

```

---

\*) Можно было бы привести более интересный пример: историк работает с некоторыми материалами и ему при этом довольно часто требуется переводить даты по одному из древних календарей в даты по современному календарю. Мы рассматриваем пример с вычислением  $\sin \sqrt{x}$  как более простой в вычислительном отношении. Проблемы, связанные с организацией диалога, в обеих задачах одинаковы.

где после каждой пары символов  $x=$  размещается число, введенное с клавиатуры, а после каждой группы символов  $f(x)=$  размещается число, вычисленное компьютером. Для того чтобы выполнение программы прекратилось, потребуется предпринять специальные меры — например выключить компьютер или воспользоваться какими-нибудь средствами прерывания выполнения программ.

Другой вариант — программа, в ходе выполнения которой на экране регулярно возникает текст, содержащий вопрос о том, следует ли продолжать работу или надо закончить выполнение программы.

```

program синкоп1(input, output);
  var x: real; s: char;
  begin s:='0';
    while s='0' do
      begin write('x='); read(x);
        writeln(' f(x)=', sin(sqrt(x)));
        writeln('продолжить? (0 или n)');
        read(s); writeln(' ');
      end
    end.

```

Когда выполняется эта программа, на экране возникает последовательность строк

```

x=..., f(x)=...
продолжить? (0 или n)
0
x=..., f(x)=...
продолжить? (0 или n)
0
. . . . .
x=..., f(x)=...
продолжить? (0 или n)
n

```

С вводом буквы  $n$  выполнение программы заканчивается. Возможность прочтения буквенных ответов человека (0 — да и  $n$  — нет) обеспечивается использованием в программе переменной типа *char*.

## ЗАДАЧИ

227. Дано: натуральное  $n$ , символы  $s_1, \dots, s_n$ . Подсчитать:

а) сколько раз среди данных символов встречается символ  $+$  и сколько раз — символ  $*$ ;

б) общее число вхождений символов  $+$ ,  $-$ ,  $*$  в последовательность  $s_1, \dots, s_n$ .

228. Дано натуральное  $n$ , символы  $s_1, \dots, s_n$ . Выяснить, встречается ли в данной последовательности символов группа из трех стоящих рядом точек.

229. Дано: натуральное  $n$ , символы  $s_1, \dots, s_n$ . Выяснить, имеются ли в последовательности  $s_1, \dots, s_n$  члены  $s_i, s_{i+1}$  такие, что  $s_i$  — это запятая, а  $s_{i+1}$  — тире.

230. Дано: натуральное  $n$ , символы  $s_1, \dots, s_n$ . Получить первое натуральное  $i$ , для которого символы  $s_i$  и  $s_{i+1}$  совпадают с буквой  $a$ . Если такой пары символов в последовательности  $s_1, \dots, s_n$  нет, то ответом должно быть число 0.

231. Дано: натуральное  $n$ , символы  $s_1, \dots, s_n$ . Известно, что среди  $s_1, \dots, s_n$  есть по крайней мере одна запятая. Найти натуральное  $i$  такое, что:

а)  $s_i$  — первая по порядку запятая;

б)  $s_i$  — последняя по порядку запятая.

232. Даны символы  $s_1, s_2, \dots$ . Известно, что символ  $s_1$  отличен от символа  $/$  и что среди  $s_2, s_3, \dots$  есть по крайней мере один символ  $/$ . Пусть  $s_1, \dots, s_n$  — символы данной последовательности, предшествующие первому символу  $/$  ( $n$  заранее не известно). Выполнить задания, сформулированные выше в задачах 227—231, предполагая, что  $n$  не входит в данные этих задач.

233. Даны символы  $s_1, s_2, \dots$ . Известно, что символ  $s_1$  отличен от восклицательного знака и что среди  $s_2, s_3, \dots$  есть по крайней мере один восклицательный знак. Пусть  $s_1, \dots, s_n$  — символы данной последовательности, предшествующие первому восклицательному знаку ( $n$  заранее не известно).

а) Выяснить, верно ли, что среди  $s_1, \dots, s_n$  имеются все буквы, входящие в слово *шина*.

б) Выяснить, имеется ли среди  $s_1, \dots, s_n$  пара соседствующих букв *но* или *он*.

в) Выяснить, имеется ли среди  $s_1, \dots, s_n$  пара соседствующих одинаковых символов.

234. Дано: натуральное  $n$ , символы  $s_1, \dots, s_n$ . Выяснить, каких символов среди  $s_1, \dots, s_n$  больше — запятых или точек с запятой? (Не исключается и случай равенства).

235. Дано: натуральное  $n$ , символы  $s_1, \dots, s_n$  ( $n > 1$ ). Выяснить, где больше вопросительных знаков — среди  $s_1, \dots, s_{[n/2]}$  или среди  $s_{[n/2]+1}, \dots, s_n$ . (Не исключается и случай равенства).

236. Дано: натуральное  $n$ , символы  $s_1, \dots, s_n$ . Подсчитать наибольшее число букв  $a$ , идущих подряд в данной последовательности символов.

237. Дано: натуральное  $n$ , символы  $s_1, \dots, s_n$ . Выяснить, встречается ли в данной последовательности символов такая группа из трех стоящих рядом точек, которой непосредственно не предшествует точка и за которой не следует точка (ср. с задачей 228).

238. Дано: натуральное  $n$ , символы  $s_1, \dots, s_n$ . Определить общее количество латинских букв (малых и больших), входящих в данную последовательность символов.

239. Вывести последовательности символов:

а)  $ABVCCC \dots ZZ \dots Z$ ;

б)  $ZYYXXX \dots AA \dots A$ ;

в)  $ABC \dots ZBC \dots Z \dots Z$ .

240. Взяв за образец программы *синкор* и *синкор1* составить программы, позволяющие определять день недели по календарным датам. Каждая дата — это тройка целых чисел: число, месяц, год (см. задачу 127).

241. Преобразовать программу *синкор*: удалить метку и оператор перехода, используя вместо них оператор цикла *while*  
 $l = 1$  *do* ....

## ГЛАВА III

### ВЫЧИСЛЕНИЯ С ХРАНЕНИЕМ ПОСЛЕДОВАТЕЛЬНОСТИ ЗНАЧЕНИЙ. МОДЕЛИРОВАНИЕ. СОРТИРОВКА

#### § 17. Массивы. Пример математической модели в биологии

Тип в программировании — это множество, для которого оговорен некоторый набор операций над элементами. Сами элементы множества называются объектами или значениями данного типа.

Типы *real* и *integer* — это числовые множества. Вместе с ними рассматривают соответствующие арифметические операции и операции сравнения. Тип *char* — это множество символов. Вместе с ним рассматривают операции сравнения.

Эти три типа — стандартные типы Паскаля. В Паскале имеются средства, позволяющие определять, исходя из имеющихся типов, новые нестандартные типы. Мы остановимся сейчас на одном из этих средств.

В математике наряду с некоторым множеством часто рассматривают множества упорядоченных пар элементов данного множества, троек и т. д. \*). Упорядоченные пары, тройки и т. д. в Паскале удобно задавать в виде массивов длины 2, 3 и т. д. Массив — это набор объектов одного типа, у каждого из которых есть индекс (номер). Элементы массива длины 20 могут иметь, например, индексы 1, 2, ..., 20 или 0, 1, ..., 19 и т. д. Способ индексации, тип элементов, длина массива фиксируются в описании того типа, к которому принадлежит массив. Рассмотрим конкретный

---

\*) Например, координаты точек плоскости образуют упорядоченные пары действительных чисел, координаты точек пространства — упорядоченные тройки.

пример. Описание, имеющее вид

```
t=array [1..20] of real *)
```

это описание типа, имя которого *t*. Объектами типа *t* будут упорядоченные наборы по 20 элементов, имеющих тип *real*; диапазон изменения значения индекса—от 1 до 20. Это описание типа, предваренное служебным словом *type* \*\*), помещается в программу перед совокупностью описаний переменных. Пусть переменная *a* описана в программе как переменная типа *t*:

```
var a: t;...
```

Тогда при выполнении программы значениями переменной *a* будут массивы длины 20, элементы которых имеют тип *real*. Для того чтобы рассматривать эти элементы по отдельности, для них применяются обозначения *a*[1], *a*[2], ..., *a*[20].

Переменная *a*—это переменная типа *t*, переменные *a*[1], *a*[2], ..., *a*[20]—это переменные типа *real*. Переменные *a*[1], *a*[2], ..., *a*[20] могут использоваться в операторах программы наравне с обычными переменными типа *real* (т. е. с переменными—идентификаторами *x*, *y*, *z* и т. д.). Заключение в квадратные скобки индекс—это не обязательно целое число, им может быть произвольное выражение со значением типа *integer*. Например, переменные *a*[*i*], *a*[2\**i*], *a*[2\**i*—1] удобно использовать для поочередного рассмотрения в цикле всех элементов массива, элементов, стоящих на четных и нечетных местах; здесь проявляется преимущество обозначений *a*[1], *a*[2], ..., *a*[20] перед обозначениями *a*<sub>1</sub>, *a*<sub>2</sub>, ..., *a*<sub>20</sub>. Значение индекса обязано лежать в указанном в описании типа диапазоне, в данном случае—в диапазоне от 1 до 20.

Операции над объектами типа *t*—это доступ к отдельным элементам массивов с помощью индексов и изменения отдельных элементов массивов с помощью операций, связанных с типом *real*.

Пример. Пусть *a*[1], ..., *a*[20]—количество осадков в миллиметрах, выпадавшее в Москве в течение первых 20 лет нашего столетия. Надо вычислить среднее количество осадков и отклонение от среднего для каждого года. Программа:

```
program осадки(input, output);  
type t=array[1..20] of real;  
var a: t; i: integer; s: real;  
begin s:= 0;
```

---

\*) Аггау—массив, порядок, строй; of—из (точнее, здесь of—предлог, служащий для образования родительного падежа—массив действительных чисел).

\*\*) Type—тип.



```

for i := 1 to 20 do
    begin read(a[i]); s := s + a[i] end;
s := s/20; writeln(s);
for i := 1 to 20 do writeln (s - a[i])
end.

```

В определении типа  $t$  мы указали диапазон изменения индекса от 1 до 20. Можно было бы взять и диапазон от 0 до 19. Вообще, можно было в качестве границ взять любые целые числа, разность между которыми равна 19. Естественным был бы диапазон от 1901 до 1920 — описание типа  $t$  выглядело бы тогда так:

```
t = array[1901..1920] of real
```

Соответствующие изменения потребовалось бы внести и в операторы цикла: *for i := 1901 to 1920 do...*

**Пример.** Дана последовательность символов (объектов типа *char*)  $s_1, s_2, \dots, s_{30}$ . Требуется определить, совпадает ли начальная часть  $s_1, \dots, s_{15}$  последовательности с ее концевой частью  $s_{16}, \dots, s_{30}$ .

Программа:

```

program дубль(input, output);
label 1;
type t = array[1..15] of char;
var x:t; y: char; r:integer;
begin
    for i := 1 to 15 do read(x[i]);
    for i := 1 to 15 do
        begin read(y);
            if x[i] < > y then
                begin write('не_');
                    goto 1
                end
            end;
        1: write('совпадают')
    end.

```

Программа не требует одновременного удерживания в памяти обеих частей данной последовательности символов.

Резюмируем и дополним сказанное. В программе на Паскале допускается использование нестандартных типов, которые описываются в самой программе. Разновидностью нестандартных типов являются типы, описываемые с помощью конструкции *array...of...* Такие нестандартные типы называются регулярными типами. Пусть описание регулярного типа  $u$  имеет вид

```
u = array[n1..n2] of r
```

где  $n_1$  и  $n_2$  — некоторые конкретные целые числа,  $n_2 \geq n_1$ , а  $r$  — это *real*, *integer* или *char*. Тогда тип  $r$  называется базовым типом по отношению к типу  $u$ . Объекты любого регулярного типа называются массивами. Если тип  $u$  описан так, как показано выше, а переменная  $a$  — это переменная типа  $u$ , то  $a[n_1], \dots, a[n_2]$  — это переменные типа  $r$ . В квадратных скобках в качестве индекса можно помещать выражение со значением типа *integer*, например, можно писать  $a[2*i+1]$ . Такие конструкции тоже являются переменными типа  $r$ . Значение индекса не должно выходить из диапазона от  $n_1$  до  $n_2$ . Операции над массивами, являющимися объектами типа  $u$  — это доступ к отдельным элементам массивов с помощью индексов и изменение отдельных элементов с помощью операций, связанных с базовым типом  $r$ . Если в программе необходимо описать несколько типов, то совокупность всех описаний типов должна быть оформлена в виде

*type*  $T_1; T_2; \dots; T_m;$  .

где  $T_1, T_2, \dots, T_m$  — описания отдельных типов. Например, такая совокупность может иметь вид

*type*  $u = \text{array}[0..10] \text{ of } \textit{real};$   
            $v = \text{array}[1..50] \text{ of } \textit{char};$

Совокупность описаний типов (в частности, совокупность, *type*  $T$ ; в которую входит одно — единственное описание  $T$ ) располагается непосредственно перед совокупностью описаний переменных. Если в программе имеется также описание меток, то оно будет располагаться перед совокупностью описаний типов:

*label* 0, 1;  
       *type*  $u = \text{array}[0..10] \text{ of } \textit{real};$   
            $v = \text{array}[1..50] \text{ of } \textit{char};$   
       *var*  $x, y: u; z: v; i, j: \textit{integer};$

и т. д.

Сделаем общее замечание об использовании массивов в программах. Массивы нужны тогда, когда для решения задачи необходимо хранение последовательности значений. Но, например, в программе вычисления суммы данных чисел  $a_1, \dots, a_{1000}$  массив не нужен: можно вводить числа по одному и накапливать их сумму. Исходные данные — это всегда последовательность значений стандартных типов (*real*, *integer*, *char*), которые могут быть прочитаны одно за другим с помощью оператора ввода *read*. Каким именно переменным — с индексами или без индексов — будут присвоены эти значения, целиком зависит от того, как составлена программа.

В заключение параграфа рассмотрим дополнительный пример, интересный в том отношении, что некоторый биологический процесс описывается в нем средствами математики и, таким образом,

дается математическая модель биологического явления. Эта модель была предложена американским биологом П. Лесли в 1948 г. Составление программы, основывающейся на этой модели, потребует привлечения массивов.

Для описания происходящего со временем изменения численности популяции \*) удобно рассматривать распределение популяции по возрастным группам (в первую группу попадают все особи в возрасте до года, во вторую — все те не попавшие в первую группу особи, возраст которых меньше двух лет и т. д.). Пусть в первую возрастную группу входит  $p_1$  особей, во вторую —  $p_2$  особей и т. д. Смертность весьма упрощенно можно учесть, положив, что все особи умирают, находясь в  $n$ -й возрастной группе, где  $n$  — некоторое фиксированное натуральное число. Ясно, что общее число особей в популяции есть  $p_1 + \dots + p_n$ . Для каждой возрастной группы имеется свой коэффициент рождаемости: если коэффициенты рождаемости для рассматриваемых возрастных групп соответственно равны  $b_1, \dots, b_n$ , то годовой приплод, обусловленный присутствием в популяции  $i$ -й возрастной группы с численностью  $p_i$ , есть  $b_i p_i$ . Таким образом, годовой приплод по всей популяции есть  $b_1 p_1 + \dots + b_n p_n$ . В то же время, особи, принадлежащие  $i$ -й возрастной группе через год перейдут в  $(i+1)$ -ю возрастную группу ( $i=1, \dots, n-1$ ), а особи, принадлежащие  $n$ -й группе, вымрут в течение года. Поэтому через год распределение популяции по возрастным группам будет следующим:  $b_1 p_1 + \dots + b_n p_n, p_1, \dots, p_{n-1}$ . Модель становится значительно более точной в результате введения кроме коэффициентов рождаемости еще и коэффициентов выживания  $s_1, \dots, s_n$  ( $0 \leq s < 1, i=1, \dots, n$ ); считается, что если в текущем году  $i$ -я возрастная группа имеет численность  $p_i$ , то в будущем году  $(i+1)$ -я возрастная группа будет иметь численность, равную  $s_i p_i$  ( $i=1, \dots, n-1$ ). При этом  $s_n=0$  (мы по-прежнему считаем, что возраст особи не превосходит  $n$  лет). Годовой приплод за счет  $i$ -й группы в этом варианте модели по-прежнему считается равным  $b_i p_i$  ( $i=1, \dots, n$ ), а распределение популяции по возрастным группам через год — имеющим вид  $b_1 p_1 + \dots + b_n p_n, s_1 p_1, \dots, s_{n-1} p_{n-1}$ . Коэффициенты выживания, так же как и коэффициенты рождаемости, выводятся на основе многолетних наблюдений за популяциями данного вида \*\*).

\*) Популяция — совокупность особей одного вида, которые длительное время (в течение многих поколений) населяют определенное пространство.

\*\*) Отмечено, например, что для ящериц характерна слабая зависимость  $s_i$  от  $i$ , т. е. выполняются приближенные равенства  $s_1 = s_2 = \dots$ ; убывание  $s_i$  с возрастанием  $i$  характерно для слонов, китов; максимум  $s_i$  для срединных значений  $i$  отмечаются у пингвинов; крайне малое значение  $s_i$  для начальных значений  $i$  наблюдается у некоторых видов рыб и т. д.

Пусть известно  $p_1, \dots, p_n$  — первоначально зарегистрированное распределение популяции по возрастным группам, а также  $b_1, \dots, b_n$  — коэффициенты рождаемости и  $s_1, \dots, s_n$  — коэффициенты выживания. Пусть дано натуральное число  $m$ . Требуется определить общую численность популяции через  $m$  лет. Будем считать, что  $n=70$  (такова, например, граница продолжительности жизни филина). Напишем вначале схему программы. Как в этой схеме, так и в тех, которые придется рассматривать в дальнейшем, мы будем позволять себе писать  $p_1, p_2, \dots$  вместо  $p[1], p[2], \dots, b_1, b_2, \dots$  вместо  $b[1], b[2], \dots$  и т. д.:

```

program популяция(input, output);
  описания;
  begin read(m);
    ввести значения элементов массивов p, b и s;
    for i:=1 to m do
      получить на месте  $p_1, p_2, \dots$  числа
         $b_1 p_1 + b_2 p_2 + \dots, s_1 p_1, s_2 p_2, \dots$ ;
      вычислить  $u = p_1 + p_2 + \dots$ ;
      write ('численность = ', u)
    end.
  
```

Описывая переход от  $p_1, p_2, \dots$  к  $b_1 p_1 + b_2 p_2 + \dots, s_1 p_1, s_2 p_2, \dots$ , надо учесть, что число  $s_{j-1} p_{j-1}$  может быть помещено на место  $p_j$  только после того, как значение  $p_j$  станет уже ненужным. Поэтому будем заменять элементы массива  $p_j$  в порядке убывания индекса: сначала заменим  $p_n$ , затем заменим  $p_{n-1}$  и т. д. Попутно с перемещением  $s_{j-1} p_{j-1}$  на место  $p_j$  будем включать  $b_j p_j$  в накапливаемую сумму  $b_n p_n + b_{n-1} p_{n-1} + \dots$ .

В соответствии с принятым ранее соглашением считаем, что  $n=70$ .

Операторы, описывающие переход от  $p_1, \dots, p_{70}$  к  $b_1 p_1 + \dots + b_{70} p_{70}, s_1 p_1, \dots, s_{69} p_{69}$  можно взять такими:

```

u:=0;
for j:=70 downto 2 do
  begin u:=u+p[j]*b[j];
    p[j]:=p[j-1]*s[j-1]
  end;
p[1]:=u+p[1]*b[1]
  
```

Пусть исходные данные расположены в следующем порядке:  $m, p_1, b_1, s_1, p_2, b_2, s_2, \dots, p_{70}, b_{70}, s_{70}$ . Тогда всю программу целиком можно написать так:

```

program популяция (input, output);
  type t=array[1..70] of real;
  var p, b, s: t; m, i, j: integer; u: real;
  
```

```

begin read(m);
  for j := 1 to 70 do read(p[j], b[j], s[j]);
  for i := 1 to m do
    begin u := 0;
      for j := 70 downto 2 do
        begin u := u + p[j]*b[j];
          p[j] := p[j-1]*s[j-1]
        end;
        p[1] := u + p[1]*b[1]
      end;
      u := p[1];
      for i := 2 to 70 do u := u + p[i];
      write('численность = ', u)
    end.
end.

```

## ЗАДАЧИ

242. Даны символы  $s_1, \dots, s_{30}$ . Получить символы данной последовательности в обратном порядке:  $s_{30}, \dots, s_1$ .

243. Даны действительные  $a_1, \dots, a_{15}, b_1, \dots, b_{15}$ . Вычислить  $(a_1 + b_{15})(a_2 + b_{14}) \dots (a_{15} + b_1)$ .

244. Система 10 материальных точек на плоскости задана с помощью действительных чисел  $x_1, y_1, m_1, \dots, x_{10}, y_{10}, m_{10}$ , где  $x_i, y_i$  — координаты  $i$ -й точки, а  $m_i$  — ее масса,  $i = 1, \dots, 10$ . Получить координаты центра масс (центра тяжести) системы, а также расстояния от центра масс до каждой из точек системы. Как изменится программа, если исходные данные будут расположены в следующем порядке:  $x_1, \dots, x_{10}, y_1, \dots, y_{10}, m_1, \dots, m_{10}$ ?

Указание. Координаты центра масс могут быть вычислены по формулам

$$x_{ц.м.} = \frac{x_1 m_1 + x_2 m_2 + \dots}{m_1 + m_2 + \dots}; \quad y_{ц.м.} = \frac{y_1 m_1 + y_2 m_2 + \dots}{m_1 + m_2 + \dots}.$$

245. Даны действительные  $a_1, \dots, a_{18}$ . Получить:

а)  $a_1, a_{10}, a_2, a_{11}, \dots, a_9, a_{18}$ ;

б)  $a_1, a_{18}, a_2, a_{17}, \dots, a_9, a_{10}$ ;

в)  $a_1 + a_{18}, a_2 + a_{17}, \dots, a_9 + a_{10}$ .

246. Даны символы  $s_1, \dots, s_{17}$ . Получить:

а)  $s_{17}, s_1, s_2, \dots, s_{16}$ ;

б)  $s_{11}, s_{12}, \dots, s_{17}, s_1, \dots, s_{10}$ ;

в)  $s_{12}, s_{13}, \dots, s_{17}, s_{11}, s_{10}, \dots, s_1$ .

247. Даны целые  $x_0, \dots, x_{15}$ . Получить последовательность чисел  $x_0 - x_{15}, x_1 - x_{15}, \dots, x_{14} - x_{15}$ .

248. Даны символы  $s_1, \dots, s_{80}$ . Определить количество неверных равенств среди:

а)  $s_1 = s_{41}, s_2 = s_{42}, \dots, s_{40} = s_{80}$ ;

б)  $s_1 = s_{80}, s_2 = s_{79}, \dots, s_{40} = s_{41}$ .

249. Даны действительные  $a_1, \dots, a_{16}$ . Получить:

а)  $\max(a_1 + a_{16}, a_2 + a_{15}, \dots, a_8 + a_9)$ ;

б)  $\min(a_1 a_9, a_2 a_{10}, \dots, a_8 a_{16})$ .

250. Даны действительные  $x_1, \dots, x_{11}, y_1, \dots, y_{11}$ . Получить действительные  $x'_1, \dots, x'_{11}, y'_1, \dots, y'_{11}$ , преобразовав для получения  $x'_i, y'_i$  члены  $x_i, y_i$  по правилу: если они оба отрицательны, то каждый из них увеличить на 0.5; если отрицательно только одно число, то отрицательное число заменить его квадратом; если оба числа неотрицательны, то каждое из них заменить на среднее арифметическое исходных значений.

251. Даны целые  $a_1, \dots, a_{12}$ . Если последовательность  $a_1, \dots, a_{12}$  упорядочена по неубыванию (т. е. если  $a_1 \leq a_2 \leq \dots \leq a_{12}$ ), то оставить ее без изменения. Иначе получить последовательность  $a_{12}, \dots, a_1$ .

252. Даны символы  $r_1, \dots, r_{20}$ . Получить последовательность символов:

а)  $r_1, \dots, r_{20}, r_1, \dots, r_{20}$ ;

б)  $r_1, \dots, r_{20}, r_{20}, \dots, r_1$ ;

в)  $r_{20}, \dots, r_1, r_1, \dots, r_{20}$ .

253. Дано натуральное  $n$ . Сколько различных цифр встречается в десятичной записи  $n$ ?

254. Взаимное влияние некоторых двух конкурирующих видов на размер  $x_n, y_n$  их популяций в  $n$ -м году описывается системой

$$\begin{aligned} x_n &= 2x_{n-1} - y_{n-1} \\ y_n &= -x_{n-1} + 2y_{n-1}. \end{aligned}$$

Пусть  $x_0 = a, y_0 = b$  ( $a \neq b$ ), где  $a$  и  $b$  — данные числа. Найти численность обеих популяций за все годы, предшествующие полному вымиранию одной из них. Нужны ли для решения этой задачи массивы?

255. Внести в программу *популяция*, приведенную в тексте параграфа, изменения, считая, что исходные данные задачи расположены в следующем порядке:  $p_1, \dots, p_{70}, b_1, \dots, b_{70}, s_1, \dots, s_{70}, m$ .

256. Дополняя модель изменения численности популяции, описанную в тексте параграфа, предположим, что когда численность популяции оказывается к концу года большей, чем некоторое число  $v$ , то из-за большой скученности начинается эпидемия, приводящая к понижению вдвое коэффициентов выживания в следующем году (если к концу следующего года численность не будет превосходить  $v$ , то коэффициенты выживания восстановятся). Считая, что даны  $v, m, p_1, b_1, s_1, p_2, b_2, s_2, \dots$ , вновь рассмотреть задачу расчета численности популяции через  $m$  лет.

## § 18. Раскрашивание фигур. Графические построения с использованием массивов

В графическом режиме можно не только высвечивать на экране отдельные точки, отрезки прямых и окружности, но и раскрашивать участки экрана. Для этого участок должен быть ограничен замкнутым контуром, состоящим из отрезков прямых и дуг окружностей одного и того же цвета\*). Если это условие выполнено, можно воспользоваться оператором *fillshape*\*\*), указав в скобках за идентификатором *fillshape* через запятую четыре выражения со

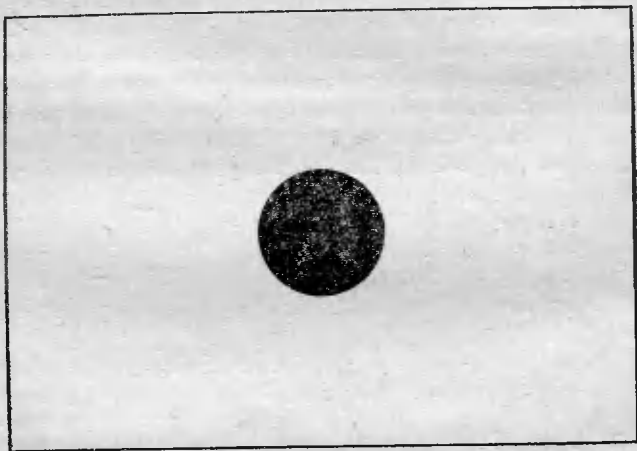


Рис. 46

значениями типа *integer*. Значения первых двух выражений определяют положение точки, принадлежащей закрашиваемому участку (с этой точки начинается закрашивание), значение третьего выражения — номер цвета, которым закрашивается участок, значение четвертого выражения — номер цвета, в который окрашен замкнутый контур, ограничивающий этот участок. Например, в результате выполнения операторов

```
circle(160, 100, 30, 1);  
fillshape(160, 100, 2, 1) *
```

в центре экрана появляется окружность зеленого цвета и внутренность этой окружности закрашивается красным цветом (рис. 46).

---

\*) В рисунках для изображения разных цветов мы будем пользоваться черным цветом и штриховкой.

\*\*) Fill — заполнять, shape — контур, очертание.

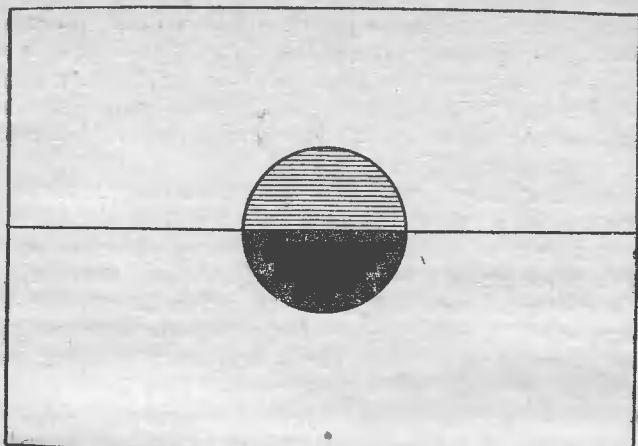


Рис. 47

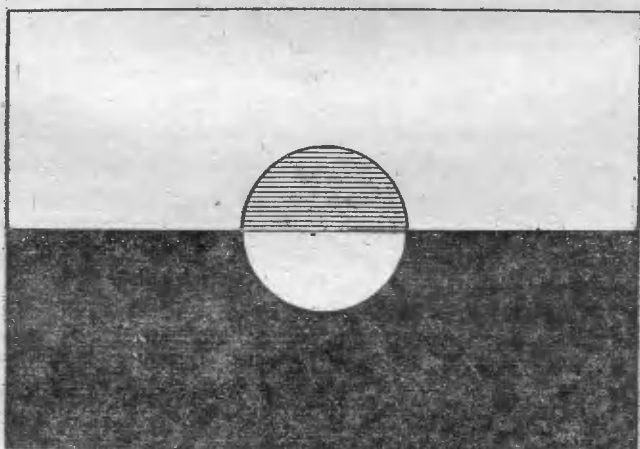


Рис. 48

В результате выполнения операторов

```
circle(160, 100, 40, 2); draw(0, 100, 319, 100, 2);
fillshape(160, 90, 1, 2); fillshape(160, 110, 3, 2)
```

на экране высвечивается окружность красного цвета и отрезок горизонтальной прямой красного цвета, проходящий через центр окружности. Затем верхняя половина круга закрашивается зеленым цветом, а нижняя — коричневым (рис. 47). Заметим, что если



бы вместо последнего оператора был выполнен оператор *fillshape* (160, 150, 3, 2), то коричневым цветом закрасилась бы вся нижняя половина экрана, кроме участка, ограниченного окружностью (рис. 48). Действительно, в этом случае точка, с которой начинается закрашивание (ее положение на экране определяется парой чисел 160, 150) не лежит внутри окружности.

Отметим, наконец, что если ограничивающий закрашиваемый участок контур окажется разомкнутым хотя бы в одной точке, то при выполнении оператора *fillshape* краска «прольется» и заполнит весь экран или его часть. Например, в результате выполнения операторов *circle*(160, 100, 30, 1); *draw*(125, 100, 135, 100, 3); *fillshape*(160, 100, 2, 1) весь экран будет «залит» красным цветом, так как окружность зеленого цвета пересечена отрезком прямой коричневого цвета.

Напишем программу, в результате выполнения которой на экране появляется шесть концентрических колец, окрашенных поочередно в красный и зеленый цвет. Положение центра окружностей определяется данными значениями переменных *a* и *b*.

```
program кольца (input),
  var i, a, b: integer;
  begin read(a, b); graphcolormode;
    for i:=1 to 7 do circle(a, b, 5*i, 3);
    for i:=1 to 6 do fillshape(a-2-5*i, b, i mod 2+1, 3)
  end.
```

Вначале выполняется построение семи концентрических окружностей с радиусами, равными 5, 10, 15, ..., 35. Затем выполняется закрашка шести колец поочередно зеленым и красным цветом. При выполнении оператора цикла *for i:=1 to 6 do ...* выражение *i mod 2+1* принимает значения 2, 1, 2, 1, 2, 1 (таким образом внутреннее кольцо закрашивается красным цветом). Выражение же *a-2-5\*i* принимает значения *a-7*, *a-12*, *a-17*, ..., *a-32* и пара значений *a-2-5\*i*, *b* определяет положение точки внутри кольца с номером *i*, с которой начинается закрашивание (рис. 49).

До сих пор мы использовали графические операторы только для создания изображений на экране. Однако те же операторы можно использовать и для удаления изображения или его части. Для этого достаточно при высвечивании точки, отрезка или окружности указать цвет, совпадающий с цветом экрана (черный, фоновый цвет, имеющий номер 0). Например, в результате выполнения операторов

```
draw(0, 100, 319, 100, 2); draw(0, 100, 159, 100, 0)
```

вначале высвечивается горизонтальный отрезок, проходящий через середину экрана, а затем левая половина этого отрезка удаляется

(высвечивается цветом с номером 0). Пользуясь возможностью удаления элементов изображения можно составлять программы, в ходе выполнения которых изображение перемещается по экрану.

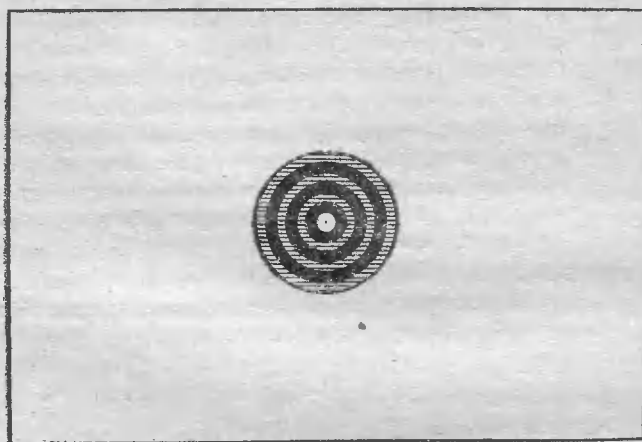


Рис. 49

Напишем программу при выполнении которой красный прямоугольник, появившись у левого края экрана, постепенно перемещается вправо и останавливается у правого края экрана.

```

program прямоугог;
  var i: integer;
  begin graphcolormode;
    draw(1, 90, 1, 110, 2); draw(1, 90, 10, 90, 2);
    draw(10, 90, 10, 110, 2); draw(10, 110, 1, 110, 2);
    fillshape(5, 100, 2, 2);
    for i:=11 to 300 do
      begin draw(i-10, 90, i-10, 110, 0);
        draw(i, 90, i, 110, 2)
      end
    end.

```

Здесь вначале строится прямоугольник из отрезков четырех прямых и закрашивается красным цветом. Затем при выполнении оператора цикла этот прямоугольник перемещается по экрану. Эффект движения возникает из-за того, что на каждом шаге цикла крайний слева вертикальный отрезок, принадлежащий прямоугольнику, удаляется с экрана и вплотную справа к прямоугольнику пририсовывается новый вертикальный отрезок красного цвета. На рис. 50 показаны отдельные фазы этого процесса: а—перед выполнением тела цикла,

б—после выполнения первого оператора тела цикла, в—после выполнения второго оператора тела цикла.

В графическом режиме запрещено выполнение оператора ввода. Для того, чтобы оператор ввода мог быть выполнен, необходимо переключить экран в символьный режим, что приводит к уничтожению полученного изображения. Поэтому обычно в программах, использующих графический режим, вначале вводятся все исходные данные, определяющие положение элементов изображения на экране, а затем уже выполняется построение изображений с использованием введенных значений. Для хранения исходных данных удобно использовать массивы.

Рис. 50

Пример. Дано 40 натуральных чисел  $x_1, y_1, x_2, y_2, \dots, x_{20}, y_{20}$ . Каждая пара чисел  $x_i, y_i$  определяет положение вершины ломаной линии на экране. Построить эту ломаную.

Программа:

```
program ломаная(input);
  type mac=array[1..20] of integer;
  var x, y: mac; i: integer;
  begin for i:=1 to 20 do read(x[i], y[i]);
    graphcolormode;
    for i:=1 to 19 do
      draw(x[i], y[i], x[i+1], y[i+1], 1)
    end.
```

Задача решается в два этапа: вначале вводится вся данная последовательность, затем выполняется построение ломаной.

Еще один пример. Дана последовательность, состоящая из 50 натуральных чисел. Первые 30 чисел определяют положение центров и радиусы десяти окружностей, следующие 20 чисел—положение на экране десяти точек. Высветить все точки и окружности зеленым цветом. Круги, не содержащие ни одной из данных точек, закрасить в красный цвет. Программа:

```
program om (input);
  label 1;
  type mac=array [1..10] of integer;
  var a, b, r, x, y: mac; i, j: integer;
  begin for i:=1 to 10 do read(a[i], b[i], r[i]);
    for i:=1 to 10 do read(x[i], y[i]);
    graphcolormode;
    for i:=1 to 10 do plot(x[i], y[i], 1);
    for i:=1 to 10 do
      begin circle(a[i], b[i], r[i], 1);
```

```

for j:=1 to 10 do
  if sqrt(sqr(a[i]-x[j])+
    sqr(b[i]-y[j]))<r[i]
  then goto 1;
fillshape(a[i], b[i], 2, 1);
1: end
end.

```

Здесь после ввода исходных данных высвечиваются зеленым цветом все точки. Затем (при  $i=1, 2, \dots, 10$ ) выполняется десять однотипных шагов: высвечивается на экране окружность, определяемая значениями  $a[i]$ ,  $b[i]$ ,  $r[i]$ , и с помощью цикла с параметром  $j$  выясняется, верно ли, что эта окружность содержит внутри себя какую-нибудь из данных точек. Условие  $\text{sqrt}(\text{sqr}(a[i]-x[j])+\text{sqr}(b[i]-y[j]))<r[i]$  соблюдено, если окружность, определяемая значениями  $a[i]$ ,  $b[i]$ ,  $r[i]$ , содержит внутри себя точку, определяемую значениями  $x[j]$ ,  $y[j]$ . Если это условие не соблюдается для всех десяти значений переменной  $j$  ( $j=1, 2, \dots, 10$ ), то внутренность окружности закрашивается красным цветом.

## ЗАДАЧИ

257. Какое изображение будет получено на экране в результате выполнения в графическом режиме операторов:

- $\text{draw}(10, 100, 110, 100, 1)$ ;  $\text{draw}(10, 100, 60, 50, 1)$ ;  
 $\text{draw}(60, 50, 110, 100, 1)$ ;  $\text{fillshape}(60, 90, 3, 1)$ ;
- $\text{draw}(10, 100, 110, 100, 1)$ ;  $\text{draw}(10, 100, 60, 50, 1)$ ;  
 $\text{draw}(60, 50, 110, 100, 1)$ ;  $\text{fillshape}(60, 40, 3, 1)$ ;
- $\text{draw}(10, 100, 110, 100, 1)$ ;  $\text{draw}(10, 100, 60, 50, 1)$ ;  
 $\text{draw}(60, 50, 110, 100, 2)$ ;  $\text{fillshape}(60, 90, 3, 1)$ ;
- $\text{draw}(10, 100, 110, 100, 1)$ ;  $\text{draw}(10, 100, 60, 50, 1)$ ;  
 $\text{draw}(60, 50, 120, 100, 1)$ ;  $\text{fillshape}(60, 90, 3, 1)$ ?

258. С помощью рассмотренных графических операторов можно получать на экране изображения, состоящие не целиком из окружностей, а из их частей. Например, получить изображение красного полумесяца можно так (см. рис. 51): построить две пересекающиеся окружности (а); закрасить полумесяц (б); удалить с экрана окружности (в). Написать программу, выполняющую построение красного полумесяца, а также программы, выполняющие построение фигур, показанных на рис. 52а—в (фигуры закрашиваются зеленым цветом).

259. Получить в центре экрана изображение, состоящее из девяти вложенных квадратов и раскрасить его тремя цветами (рис. 53).

260. Дано восемь натуральных чисел  $x_1, y_1, l_1, h_1$  и  $x_2, y_2, l_2, h_2$ . Каждая четверка чисел задает положение на экране прямоугольника со сторонами, параллельными сторонам экрана. Значения

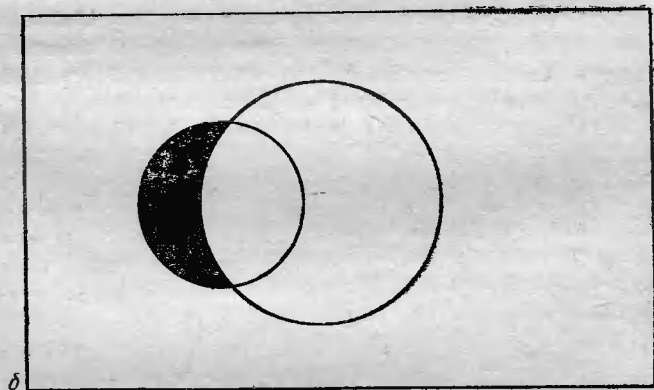
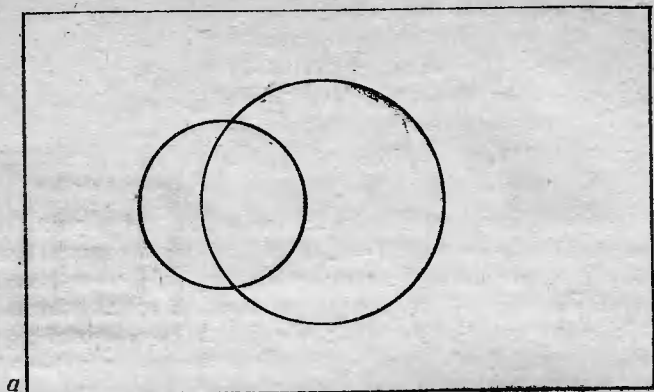


Рис. 51

$x_i$ ,  $y_i$  определяет положение левого нижнего угла прямоугольника с номером  $i$  ( $i=1, 2$ ),  $l_i$ —длину основания,  $h_i$ —высоту. Построить прямоугольники и закрасить первый зеленым цветом, второй — красным. Если прямоугольники пересекаются, то их общую часть закрасить коричневым цветом.

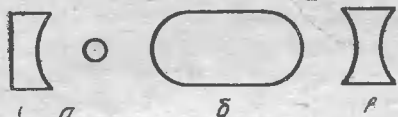


Рис. 52

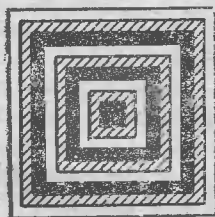


Рис. 53

261. Написать программу, в ходе выполнения которой круг зеленого цвета, появившись в центре экрана и постепенно расширяясь, увеличивается в размерах в три раза, а затем сжимается до начальных размеров.

262. Модифицировать программу *колобок* из § 11 так, чтобы после появления изображения на экране, точки, высвечиваемые внутри малых окружностей, перемещались одновременно несколько раз влево—вправо (т. е. чтобы колобок двигал глазами).

263. Пропеллер состоит из двух закрасенных треугольников. Получить на экране вращающийся пропеллер.

264. Дано два натуральных числа. Написать программу, в ходе выполнения которой отрезок, появившись в левом верхнем углу экрана, передвинется по экрану так, что его левый конец совместится с точкой, положение которой определяется данными числами. Весь путь отрезка должен состоять из двух участков—горизонтального и вертикального.

265. Написать программу, в ходе выполнения которой зеленый квадрат, появившись в левом верхнем углу экрана, перемещается вправо вниз по диагонали.

266. Дана последовательность, состоящая из 40 натуральных чисел  $x_1, y_1, l_1, h_1, \dots, x_{10}, y_{10}, l_{10}, h_{10}$ . Каждая четверка чисел  $x_i, y_i, l_i, h_i$  задает положение прямоугольника на экране. Высветить все прямоугольники и закрасить их коричневым цветом.

267. Дана последовательность, состоящая из 70 натуральных чисел. Первые тридцать чисел определяют положение на экране 10 окружностей. Остальные 40 чисел—положение на экране 20 точек. Высветить:

а) все окружности и те из данных точек, которые лежат внутри не более чем одной окружности;

б) только те окружности, внутри которых содержится хотя бы одна из данных точек;

в) все точки, не попадающие внутрь окружностей и все окружности, не содержащие ни одной из данных точек.

268. Даны натуральные  $x_1, y_1, \dots, x_{20}, y_{20}$ . Построить на экране точки, задаваемые парами значений  $x_i, y_i$  ( $i=1, \dots, 20$ ) и соединить пары точек, наиболее удаленные друг от друга.

269. Даны натуральные  $x_1, y_1, \dots, x_{20}, y_{20}$ . Построить на экране точки, задаваемые парами значений  $x_i, y_i$  ( $i=1, \dots, 20$ ) и соединить отрезками прямых:

а) каждую из точек со всеми остальными;

б) точки с номерами одной четности;

в) точки с номерами разной четности.

270. Даны натуральные  $x_1, y_1, \dots, x_{40}, y_{40}$ . Построить на экране точки, задаваемые парами значений  $x_i, y_i$  ( $i=1, \dots, 40$ ) и прямоугольник со сторонами, параллельными сторонам экрана, содержащий все эти точки.

271. Даны натуральные  $x_1, y_1, r_1, c_1, \dots, x_{10}, y_{10}, r_{10}, c_{10}$ . Каждая четверка значений  $x_i, y_i, r_i, c_i$  ( $i=1, \dots, 10$ ) определяет положение окружности на экране и ее цвет. Высветить окружности:

а) цвет которых совпадает с цветом первой окружности;

б) цвет которых не совпадает с цветом предпоследней окружности;

в) цвет которых встречается чаще всего.

## § 19. Константы

Нетрудно заметить одно существенное неудобство, связанное с использованием регулярных типов в программах на Паскале. Оно состоит в том, что приходится сразу же жестко фиксировать число элементов массива. Мы не можем рассматривать массивы, число элементов которых равно  $n$ , где  $n$  — одно из исходных данных программы. Применение программы *дубль* из § 17 возможно только тогда, когда дано именно 30 символов, хотя, в принципе, эта задача может возникнуть и для 100 символов, и для 18, и т. д. Для того чтобы от приведенной программы *дубль* перейти к такому ее варианту, в котором рассматривается 200 символов (и в результате выполнения которого выясняется, верно ли, что  $s_1 = s_{101}$ ,  $s_2 = s_{102}, \dots, s_{100} = s_{200}$ ), потребуется в имеющийся вариант внести исправления. Выпишем новый вариант, рассчитанный на 200 символов, и подчеркнем все места, в которых потребовались исправления:

```
program дубль(input, output);  
  label 1;
```

```

type t=array[1..100] of char;
var x; t; y: char; i: integer;
begin for i:=1 to 100 do read (x[i]);
      for i:=1 to 100 do
        begin read(y);
          if x[i] <> y then
            begin write('не');
              goto 1
            end
          end;
        end;
      1: write('совпадают')
    end.

```

Помимо того, что надо вносить значительное число исправлений, требуется предварительно найти все места, нуждающиеся в исправлении. Если программа велика, то велика и вероятность того, что какие-то места будут пропущены.

В Паскале есть средство, позволяющее уменьшить трудности, связанные с такими исправлениями программ — речь идет о *константах*. Константа в программе на Паскале — это идентификатор, являющийся обозначением конкретного числа, которое называется значением константы; отличие же константы от переменной в том, что ее значение нельзя изменять с помощью операторов программы\*), а также в том, что значение константы закрепляется за ней еще до выполнения операторов, в ее описании.

Константы не могут встречаться в операторах присваивания слева от комбинации символов  $:=$ , но их можно использовать, как и переменные, в качестве компонент выражения. Важнее всего, однако, то, что константу с целым значением можно использовать в описании регулярного типа в качестве границы изменения индекса. Но прежде этого каждая константа сама должна быть описана в программе. Приведем пример описания:

```
const n=100;
```

Такого рода описание может охватывать и несколько констант. Пример.

```
const n=100; m=5; k=12;
```

Описание констант предшествует описанию типов (таким образом, сначала описываются метки, затем — константы, затем — типы, затем — переменные; если в программе нет меток или констант и т. д., то и соответствующий вид описания отсутствует). Рассмотрим новый вариант программы *дубль*:

\*) Напомним, что слово константа происходит от латинского constants — постоянный.



```

program дыбль (input, output);
  label 1;
  const n=15;
  type t=array[1..n] of char;
  var x: t; y: char; i: integer;
  begin for i:=1 to n do read(x[i]);
        for i:=1 to n do
          begin read(y);
                if x[i] <> y then
                  begin write('не_');
                        goto 1
                  end
                end;
          i: write('совпадают');
        end;
end;

```

Эта программа, как и программа из § 17 рассчитана на 30 символов, но теперь для перехода к любому другому четному числу символов достаточно внести в программу одно исправление: надо изменить описание константы  $n$  (подчеркнем, что  $n$  — это количество данных символов, деленное на два;  $n=15$ , когда рассматривается 30 символов;  $n=9$ , когда рассматривается 18 символов и т. д.).

Рассмотрим еще примеры программ, в которых используются константы. Даны целые  $a_1, \dots, a_n$ , где  $n$  — некоторая константа. Требуется получить все различные числа, входящие в последовательность  $a_1, \dots, a_n$  (другими словами, из каждой группы равных чисел должно быть выведено только одно: если данные числа — это 1, 0, 1, 2, 2, 0, 3, то должны быть выведены 1, 0, 2, 3). Мы не можем написать программу, не зная значения константы  $n$ . В этом и подобных случаях будем придавать константам некоторые конкретные осмысленные значения (например, в данной задаче положим  $n=20$ ), имея в виду, что при необходимости мы можем перейти к любому другому значению с помощью одного исправления программы. Возвращаясь к нашей задаче, видим, что она допускает следующее простое решение: числа вводятся поочередно, и если после ввода  $a_i$  обнаруживается, что для него нет равного среди  $a_1, \dots, a_{i-1}$ , то  $a_i$  сразу же выводится:

```

program разные(input, output);
  label 1;
  const n=20;
  type t=array[1..n] of integer;
  var a: t; l, j: integer;
  begin
    for i:=1 to n do
      begin read(a[i]);

```

```

    for j:=1 to i-1 do
      if a[i]=a[j] then goto 1;
    writeln(a[i]);
  1: end
end.

```

Можно сократить число сравнений вида  $a[i]=a[j]$ , если введенное число заносится в массив  $a[1], a[2], \dots$  только в случае отсутствия для него равного среди уже имеющихся элементов массива. Напишем программу, реализующую этот подход. Переменная  $k$  будет указывать, сколько различных чисел уже выведено:

```

program разные1(input, output);
  label 1;
  const n=20;
  type t=array[1..n] of integer;
  var a: t; x, i, j, k: integer;
  begin k:=0;
    for i:= 1 to n do
      begin read(x);
        for j:= 1 to k do
          if x=a[j] then goto 1;
        writeln(x); k:=k+1;
        a[k]:=x;
      1: end
    end.

```

Видно, что те значения, которые принимают переменные  $k$  и  $i$ , на каждом этапе выполнения программы связаны неравенством  $k \leq i$ . Если к моменту окончания выполнения программы выполнено  $k < n$ , то это означает, что переменные  $a[k+1], a[k+2], \dots, a[n]$ , не получили в ходе выполнения программы никаких значений.

Иногда в задачах, подобных последней, оказывается полезным рассмотрение дополнительного целочисленного массива  $b[1], b[2], \dots$ ; число  $b[i]$  может, например, показывать, сколько раз среди исходных данных встречается значение, равное  $a[i]$ . Пусть даны символы  $a_1, \dots, a_n$ , где  $n$  — некоторая константа, и пусть требуется вывести каждый символ по одному разу, указав при этом, сколько раз символ встречается среди  $a_1, \dots, a_n$ . Будем исходить из полученного решения предыдущей задачи, но при этом введем в рассмотрение дополнительный массив  $b[1], \dots, b[n]$ . Приходим к следующей схеме:

```

program сколько(input, output);
  описания;
  begin k:=0;
    for i:=1 to n do
      begin read(x);

```

если  $x$  совпадает с некоторым  $a[j]$ ,  $1 \leq j \leq k$ , то  
увеличить  $b[j]$  на 1 и перейти к метке 1;

$k := k + 1$ ;  $a[k] := x$ ;  $b[k] := 1$ ;

1: end;

вывести  $a[1]$ ,  $b[1]$ ,  $a[2]$ ,  $b[2]$ , ...,  $a[k]$ ,  $b[k]$

end.

Необходимая детализация проводится несложно:

```
program сколько(input, output);
  label 1;
  const n=20;
  type t=array[1..n] of char;
  u=array[1..n] of integer;
  var a: t; b: u; i, j, k: integer; x: char;
  begin k:=0;
    for i:=1 to n do
      begin read(x);
        for j:=1 to k do
          if x=a[j] then
            begin b[j]:=b[j]+1;
              goto 1
            end;
          k:=k+1; a[k]:=x; b[k]:=1;
        1: end;
      for i:=1 to k do writeln (a[i], ' ', b[i])
    end.
```

## ЗАДАЧИ\*)

272. Внести изменения в программу *разные1* для того, чтобы результатом выполнения новой программы было число различных чисел в последовательности  $\{a_1, \dots, a_n\}$ . Например, если данные числа—это 1, 0, 1, 2, 2, 0, 3, то ответом должно быть число 4.

273. Выяснить, имеются ли среди данных символов  $s_1, \dots, s_n$  совпадающие.

274. Даны целые  $a_1, \dots, a_n$ . Определить количество целых чисел, входящих в последовательность  $a_1, \dots, a_n$  по одному разу.

275. Даны целые  $a_1, \dots, a_n$ . Из модулей членов данной последовательности выбрать наибольший. Получить новую последовательность из  $n$  целых чисел (нулей и единиц), заменяя  $a_i$  нулем, если  $|a_i|$  не совпадает с выбранным значением, и заменяя  $a_i$  единицей, если совпадает.

---

\*) Предполагается, что число элементов массивов определяется в программе константой  $n$ . Имея в виду это предположение, полезно еще раз обратиться к задачам § 17.

276. Даны целые  $a_1, \dots, a_n$ . Получить новую последовательность, выбросив из исходной все члены с наибольшим значением (если все члены последовательности равны, то ничего выводить не нужно).

277. Даны целые  $a_1, \dots, a_n$ , действительные  $b_1, \dots, b_n$ . Преобразовать последовательность  $b_1, \dots, b_n$  по правилу: если  $a_i$  делится на 10, то  $b_i$  увеличить в 10 раз, иначе  $b_i$  заменить нулем.

278. Даны целые  $a_1, \dots, a_n$ . Все члены последовательности  $a_1, \dots, a_n$ , предшествующие первому по порядку наименьшему члену, домножить на этот наименьший член.

279. Даны действительные  $a_1, \dots, a_n$ . Требуется найти  $b$ , равное среднему арифметическому чисел  $a_1, \dots, a_n$ , и наибольшее отклонение от среднего, т. е.  $\max(|a_1 - b|, |a_2 - b|, \dots, |a_n - b|)$ .

280. Даны целые попарно различные числа  $m_1, \dots, m_n$ . Получить  $m_k, m_{k+1}, \dots, m_n$ , где  $m_k$  — член последовательности  $m_1, \dots, m_n$  с наибольшим значением.

281. Даны символы  $s_1, \dots, s_n$ . Получить последовательность  $s_m, \dots, s_1$ , где  $s_m$  — последняя малая латинская буква последовательности  $s_1, \dots, s_n$  (если малых латинских букв в последовательности нет, то ничего выводить не нужно).

282. Даны символы  $s_1, \dots, s_n$ . Получить в обратном порядке все цифры, входящие в последовательность  $s_1, \dots, s_n$ .

283. Даны действительные  $a_0, a_1, \dots, a_n$ . Вычислить  $a_0 a_n + a_1 a_{n-1} + \dots + a_n a_0$  (обратить внимание на наличие совпадающих слагаемых в выписанной сумме).

284. Если в данной последовательности действительных чисел  $a_1, \dots, a_n$  есть хотя бы один член, меньший, чем  $-2$ , то все отрицательные члены заменить их квадратами, оставив остальные без изменений; в противном случае домножить все члены на 0.1.

285. Многочлен  $p(x) = p_n x^n + p_{n-1} x^{n-1} + \dots + p_0$  задан последовательностью своих действительных коэффициентов  $p_n, p_{n-1}, \dots, p_0$ , аналогично многочлен  $q(x) = q_m x^m + q_{m-1} x^{m-1} + \dots + q_0$  — последовательностью  $q_m, q_{m-1}, \dots, q_0$ . Получить последовательность  $r_{n+m}, r_{n+m-1}, \dots, r_0$  коэффициентов многочлена  $r(x) = p(x)q(x)$ .

286. Найти все меньшие  $n$  простые числа, используя решето Эратосфена. Решетом Эратосфена называется следующий алгоритм. Выпишем подряд все целые числа от 2 до  $n$ . Первое простое число 2. Подчеркнем его, а все большие числа кратные 2 зачеркнем. Первое из оставшихся чисел 3. Подчеркнем его как простое, а все большие числа кратные 3 зачеркнем. Первое число из оставшихся теперь 5, так как 4 уже зачеркнуто. Подчеркнем его как простое, а все большие числа кратные 5 зачеркнем и т. д.:

2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, 10, ...

## § 20. Таблицы значений функций и линейная интерполяция

Пусть некоторая функциональная зависимость  $y = f(x)$  описана не полностью: числовые значения функции  $f(x)$  известны только для нескольких значений аргумента — чисел  $x_1, x_2, \dots, x_n$  ( $x_1 <$

$x$	$f(x)$
$x_1$	$y_1$
$x_2$	$y_2$
$\vdots$	$\vdots$
$x_n$	$y_n$

Рис. 54

$< x_2 < \dots < x_n$ ). Иными словами, все сведения, которыми мы располагаем относительно функции  $f(x)$  — это таблица ее значений (рис. 54). Пусть требуется определить значения функции  $f(x)$  для ряда значений аргумента. Некоторые из этих значений могут совпадать с какими-то из чисел  $x_1, \dots, x_n$ . С такими значениями не возникает никаких сложностей. Однако, могут потребоваться и значения функции для не попавших в таблицу промежуточных значений аргумента. Довольно часто

таблицы составляются на основе некоторого экспериментального материала — например, результатов измерения скорости распространения звука в воздухе при значе-

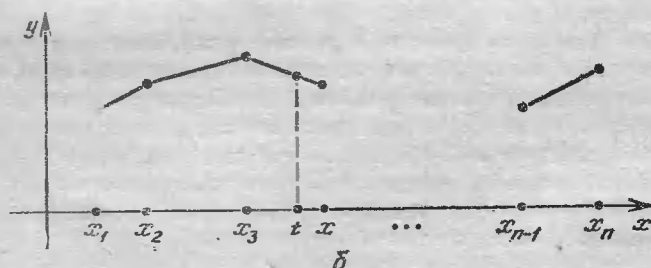
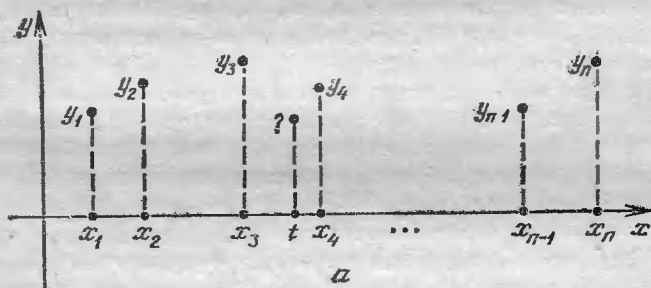


Рис. 55

ниях температуры воздуха  $x_1, \dots, x_n$ , результатов нескольких замеров глубины моря на определенной географической широте и т. д. В подобных случаях приходится исходить только из той информации о функции  $f(x)$ , которая содержится в таблице.

Ситуацию можно представить себе так: известно несколько точек, принадлежащих графику функции, и надо приблизительно определить значение функции для промежуточных значений аргумента (рис. 55а). Самый простой путь — это замена неизвестных дуг графика прямолинейными отрезками (рис. 55б). Такая замена дает

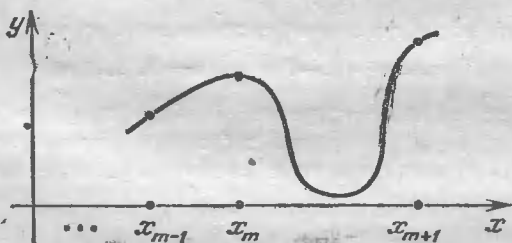


Рис. 56

хорошие результаты, если график функции является достаточно плавной линией, без внезапных выпадов, подобных изображенному на рис. 56.

Для того чтобы получить возможность детального описания получающегося алгоритма, выясним, прежде всего, вид уравнения прямой, проходящей через точки с заданными координатами  $(a, v)$ ,  $(b, w)$  (рис. 57) при  $a \neq b$ . Рассмотрим линейное уравнение

$$y = \frac{w-v}{b-a}(x-a) + v.$$

Подставляя  $x=a$ , получаем  $y=v$ ; подставляя  $x=b$ , получаем  $y=w-v+v=w$ . Таким образом, приведенное уравнение удовлетворяет требуемым условиям (его можно переписать в стандартном виде  $y=kx+b$ , но и вид, указанный выше, в данном случае вполне удобен). Итак, если значение аргумента  $t$  лежит между  $x_m$  и  $x_{m+1}$  ( $1 \leq m < n$ ), то в качестве приближенного значения функции можно взять

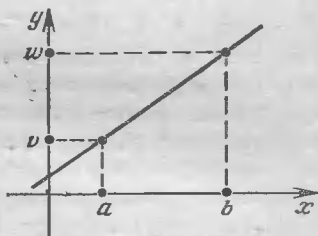


Рис. 57

$$\frac{y_{m+1} - y_m}{x_{m+1} - x_m}(t - x_m) + y_m.$$

Этот алгоритм получения отсутствующего в таблице значения  $y$  называется алгоритмом линейной интерполяции. Слово «интерполяция» имеет латинское происхождение и означает поиск промежуточных значений; прилагательное «линейная» подчеркивает факт привлечения линейных функций.

Как правило таблицы составляют так, что разности между известными соседними значениями аргумента равны одному и тому же числу  $h$ , называемому *шагом таблицы*:

$$x_m - x_{m-1} = h \quad (m = 2, 3, \dots, n).$$

Если  $x_1 = a$ , то из последнего соотношения получается, что

$$x_m = a + (m-1)h \quad (m = 1, \dots, n).$$

Если теперь значение аргумента  $t$  лежит между  $a + (m-1)h$  и  $a + mh$ , т. е. между соседними табличными значениями аргумента  $x_m$  и  $x_{m+1}$  ( $1 \leq m < n$ ), то в качестве приближенного значения функции можно взять

$$\frac{y_{m+1} - y_m}{h} (t - a - (m-1)h) + y_m.$$

Задача поиска соответствующего натурального  $m$  по данному действительному  $t$  решается несложно (предполагая, что  $a \leq t \leq a + (n-1)h$ ). В самом деле, вычислим значение  $r = (t - a)/h$ . Имеем  $t = a + rh$ , но  $r$  здесь может быть и нецелым числом. Если взять целую часть числа  $r$ , т. е.  $[r]$ , то будем, очевидно, иметь  $t \geq a + [r]h$ , или более подробно,

$$a + [r]h \leq t < a + ([r] + 1)h.$$

Взяв  $m = [r] + 1$ , мы получаем

$$a + (m-1)h \leq t < a + mh.$$

Если  $m < n$ , то задача поиска  $m$  решена. Равенство  $m = n$  возможно в единственном случае, когда  $t = a + (n-1)h = x_n$ . В этом единственном случае следует определить  $m$  иначе, взяв  $m = n-1$ . При неотрицательном  $r$  целая часть  $r$  может быть вычислена как значение функции  $\text{trunc}(r)$ , поэтому значение  $m$  можно вычислить так:

$$m := \text{trunc}((t - a)/h) + 1; \text{ if } m = n \text{ then } m := n - 1.$$

Напишем теперь всю программу, реализующую алгоритм линейной интерполяции. Будем рассматривать задачу в следующей постановке. Таблица значений функции задана числами  $a, h, y_1, \dots, y_n$  ( $n$  — некоторая константа). Далее,  $k$  — натуральное,  $t_1, \dots, t_k$  — действительные числа ( $a \leq t_i \leq a + (n-1)h$ ,  $i = 1, \dots, k$ ). Требуется получить значения функции для значений аргумента, равных  $t_1, \dots, t_k$ .

Примем, что значение константы  $n$  равно 10. Программа:

```
program линм(input, output);  
  const n=10;  
  type tab=array[1..n] of real;  
  var y: tab; a, h, t: real; i, k, m: integer;  
  begin read(a, h);  
    for i:=1 to n do read(y[i]);  
    read(t);  
    for i:=1 to k do  
      begin read(t); m:=trunc((t-a)/h)+1;  
        if m=n then m:=n-1;  
        writeln((y[m+1]-y[m])/h*  
          (t-a-(m-1)*h)+y[m])  
      end  
    end.  
  end.
```

Подчеркнем, что если какое-то значение  $t$  равно табличному значению аргумента  $x_l$  ( $1 \leq l \leq n$ ), то выполнение программы дает правильный ответ  $y_l$  — табличное значение функции.

Большинство таблиц, как уже говорилось, являются таблицами с постоянным шагом. Однако встречаются и таблицы, в которых разности между соседними известными значениями аргумента  $x_1, \dots, x_n$  не равны между собой: измерения некоторой величины (скорости, температуры, глубины, численности населения и т. д.) могли носить нерегулярный характер. Такую таблицу естественно представлять в программе двумя массивами:

$$x_1, \dots, x_n \text{ и } y_1, \dots, y_n \quad (x_1 < x_2 < \dots < x_n).$$

Пусть  $x_1 \leq t \leq x_n$ ; поиск  $m$  такого, что  $t$  лежит между  $x_m$  и  $x_{m+1}$  можно задать, например, следующими операторами:

$m := 1$ ; while  $t > x[m+1]$  do  $m := m + 1$

В заключение отметим, что идея замены графика функции прямой линией (или дуги графика прямолинейным отрезком, т. е. хордой) положена в основу целого ряда алгоритмов приближенных вычислений. По поводу некоторых из них см. задачи 295—302.

### ЗАДАЧИ \*)

287. Написать уравнение прямой, проходящей через точки с координатами (1, 1) и (1.5, 1.8).

---

\*) Задачи 287—289, 299 решаются без компьютера, остальные задачи предполагают написание и использование программ.



288. Имеются следующие данные измерения температуры воздуха в продолжении дня:

8 часов — 7°;      14 часов — 15°;  
10 часов — 10°;    17 часов — 9°;  
13 часов — 15°;    20 часов — 8°.

С помощью алгоритма линейной интерполяции определить, какой была температура воздуха в 9 часов, в 12 часов 30 минут, в 15 часов 15 минут. Начертить график изменения температуры в виде ломаной линии и проверить результаты вычислений. (Использование графика, имеющего вид ломаной линии — это тоже применение алгоритма линейной интерполяции.)

289. Используя сведения о температуре, приведенных в предыдущей задаче, определить, в какое время температура была равна 9°, 12°, 14,5°. (Начертить график изменения температуры в виде ломаной линии и решить задачу с помощью этого графика.)

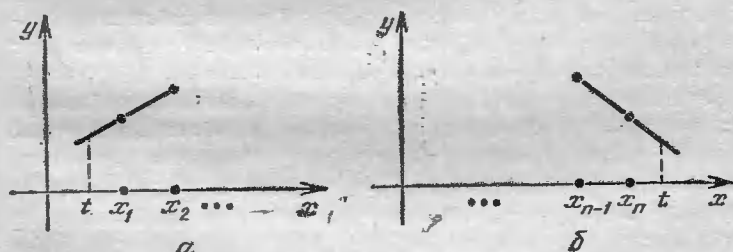


Рис. 58

290. Изменить программу *линт* таким образом, чтобы для значений  $t$ , выходящих за границы отрезка  $[x_1, x_n]$ , тоже получался бы осмысленный ответ: если  $t < x_1$ , то для определения ответа может использоваться прямая, проходящая через  $[x_1, y_1]$  и  $[x_2, y_2]$  (рис. 58а), а если  $t > x_n$ , то — прямая, проходящая через точки  $(x_{n-1}, y_{n-1})$ ,  $(x_n, y_n)$  (рис. 58б). Продумать также соответствующие изменения, предназначенные для случая непостоянного шага (см. выписанные в конце настоящего параграфа операторы поиска  $x_m$  и  $x_{m+1}$ ).

291. Таблица значений некоторой функции имеет постоянный шаг и задана числами  $a, b, y_1, \dots, y_{50}$ . С помощью алгоритма линейной интерполяции получить такую таблицу значений этой же функции, которая имеет шаг  $h/3$ . (Сколько значений функции будет содержать новая таблица?)

292. Написать варианты программы *линт*, не предполагая, что количество значений  $t_1, t_2, \dots, t_k$  известно. В качестве образца использовать программы *синкор* и *синкор1* из § 16.

293. Написать вариант программы *линт* для таблицы, шаг которой не является постоянным (см. обсуждение связанных с этим вопросом в конце текста настоящего параграфа).

а) Рассмотреть случай, когда количество  $k$  значений аргумента известно.

б) Рассмотреть случай, когда количество  $k$  значений аргумента не известно (принять за образец программы *синкор* и *синкор1* из § 16).

294. Дана таблица некоторой функции. Шаг таблицы не является постоянным (см. обсуждение связанных с этим вопросов в конце текста настоящего параграфа). Используя алгоритм линейной интерполяции, получить для этой функции таблицу с постоянным шагом. Число значений функции в новой таблице задано.

295. Применить алгоритм линейной интерполяции к таблице какой-нибудь известной функции, например функции  $\sin x$  или  $x^2$ . Сравнить результаты интерполяции со значениями функции.

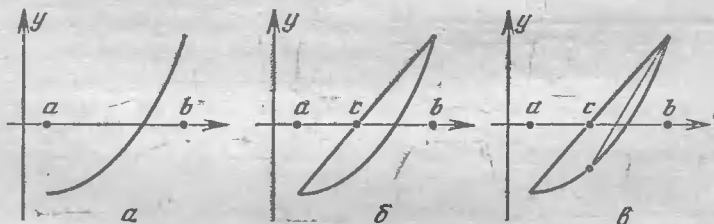


Рис. 59

296. Пусть функция  $y = f(x)$ , график которой является непрерывной линией принимает при  $x=a$  и  $x=b$  значения разных знаков:  $f(a)f(b) < 0$  (рис. 59а). Корню уравнения  $f(x)=0$  соответствует точка пересечения графика функции  $y=f(x)$  с осью абсцисс. В качестве приближения к корню, принадлежащему отрезку  $[a, b]$ , можно взять точку  $c$  пересечения прямой проходящей через точки с координатами  $(a, f(a))$ ,  $(b, f(b))$ , с осью абсцисс (рис. 59б). Далее можно перейти к поиску корня на отрезке  $[a, c]$  или на отрезке  $[c, b]$  в зависимости от знака  $f(c)$  (в ситуации, изображенной на рис. 59б следует перейти к отрезку  $[c, b]$  и т. д.). В отличие от алгоритма деления отрезка пополам этот процесс не всегда приводит к неограниченному уменьшению отрезка (рис. 59а). Поэтому в качестве условия, при выполнении которого завершается процесс вычисления корня, здесь обычно берется  $|f(c)| < \varepsilon$ , где  $\varepsilon$  — данное малое положительное число (см. задачу 98). Уравнением прямой, проходящей через точки с координатами  $(a, f(a))$ ,  $(b, f(b))$  будет  $y = \frac{f(b)-f(a)}{b-a}(x-a) + f(a)$ . Решением уравнения  $\frac{f(b)-f(a)}{b-a}(x-a) +$

$+f(a)=0$  будет  $a = \frac{b-a}{f(b)-f(a)} f(a)$ , именно это значение соответствует точке  $c$  на рис. 596. Этот алгоритм приближенного решения уравнения называется алгоритмом хорд. Довольно часто этот алгоритм требует меньших вычислений, чем алгоритм деления отрезка пополам.

Вернуться к задаче 91 и решить выписанные там уравнения с помощью программ, реализующих алгоритм хорд.

297. Один из алгоритмов приближенного интегрирования (по поводу приближенного интегрирования см. § 12), называемый алгоритмом трапеций, заключается в следующем.

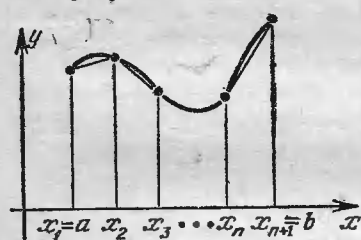


Рис. 60

Отрезок  $[a, b]$  разбивается на  $n$  равных частей точками  $x_1, x_2, \dots, x_{n+1}$  так, что  $x_1=a, x_{n+1}=b$  (рис. 60). Площадь каждой полосы между прямыми  $x=x_k$  и  $x=x_{k+1}$  ( $k=1, \dots, n$ ) приближенно вычисляется как площадь трапеции, ограниченной этими прямыми, осью абсцисс и хордой, соединяющей точки с координатами  $(x_k, f(x_k)), (x_{k+1}, f(x_{k+1}))$ .

Такая трапеция имеет площадь  $(x_{k+1}-x_k) \frac{f(x_k)+f(x_{k+1})}{2} = \frac{b-a}{n} \times \frac{f(x_k)+f(x_{k+1})}{2}$ . Сумма этих площадей равна

$$\begin{aligned} \frac{b-a}{n} \left( \frac{f(x_1)+f(x_2)}{2} + \frac{f(x_2)+f(x_3)}{2} + \dots + \frac{f(x_n)+f(x_{n+1})}{2} \right) = \\ = \frac{b-a}{n} \left( \frac{f(x_1)}{2} + f(x_2) + \dots + f(x_n) + \frac{f(x_{n+1})}{2} \right). \end{aligned}$$

Вводя обозначение  $h = \frac{b-a}{n}$ , получаем формулу приближенного интегрирования:

$$\begin{aligned} \int_a^b f(x) dx \approx \\ \approx h \left( \frac{f(a)}{2} + f(a+h) + f(a+2h) + \dots + f(a+(n-1)h) + \frac{f(a+nh)}{2} \right). \end{aligned}$$

Вернуться к задаче 166 и вычислить выписанные там интегралы с помощью программ, реализующих алгоритм трапеций.

298. Иногда функция  $y=f(x)$  задается на некотором отрезке с помощью уравнения вида  $F(x, y)=0$ —см. последний пример § 14. Вернуться к функции, рассмотренной в этом примере, и найти приближенное значение ее интеграла на отрезке  $[1, 2]$ . В качестве шага приближенного интегрирования  $h$  взять 0.01. Воспользоваться

каким-нибудь алгоритмом приближенного решения уравнений (например алгоритмом хорд) и каким-нибудь алгоритмом приближенного интегрирования (например, алгоритмом трапеций). Аналогичным образом рассмотреть  $y^3 + x^3 - 2xy = 0$  и проинтегрировать  $y$  как функцию  $x$  на отрезке  $[-1, 0]$ ; принять во внимание, что значение  $y$  в этом случае неотрицательно и меньше, чем 1.

299. Часто пытаются экспериментальные или статистические данные «уложить» в простую формулу. Пусть, например, испытания триода показали, что при значениях напряжения между сеткой и катодом  $x_1, x_2, \dots, x_n$  анодный ток имеет величину  $y_1, y_2, \dots, y_n$ . Допустим, что по этим результатам видно, что зависимость  $y$  от  $x$  близка к линейной зависимости вида  $y = kx + b$ . Ввиду погрешности эксперимента и других причин нельзя рассчитывать на то, что для каких-либо конкретных  $k$  и  $b$  будут с абсолютной точностью выполнены все  $n$  равенств  $y_i = kx_i + b$  ( $i=1, \dots, n$ ). Обычно берут  $k$  и  $b$  такими, что величина  $(y_1 - kx_1 - b)^2 + (y_2 - kx_2 - b)^2 + \dots + (y_n - kx_n - b)^2$  имеет наименьшее из возможных значений (рис. 61)\*. Как показал Гаусс, выписанная сумма квадратов имеет наименьшее значение при следующих  $k$  и  $b$ :

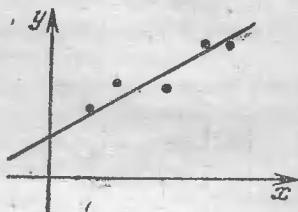


Рис. 61

$$k = \frac{nD - AC}{nB - A^2}, \quad b = \frac{BC - AD}{nB - A^2},$$

где

$$\begin{aligned} A &= x_1 + \dots + x_n, \\ B &= x_1^2 + \dots + x_n^2, \\ C &= y_1 + \dots + y_n, \\ D &= x_1 y_1 + \dots + x_n y_n. \end{aligned}$$

Этот алгоритм получения коэффициентов линейной зависимости  $y = kx + b$  называется алгоритмом наименьших квадратов.

Пусть при экспериментальном исследовании зависимости величины  $y$  от величины  $x$  установлено, что для  $x=1, 2, 3, 4, 5$  значения  $y$  соответственно равны 4,5, 7, 9, 12, 13. На основе этого экспериментального материала надо вывести линейный закон зависимости  $y$  от  $x$ . Предлагается сделать это двумя способами: с помощью алгоритма наименьших квадратов и путем подбора: на

\*) Отметим, что точное выполнение всех равенств  $y_i = kx_i + b$  ( $i=1, \dots, n$ ) эквивалентно одному равенству  $(y_1 - kx_1 - b)^2 + (y_2 - kx_2 - b)^2 + \dots + (y_n - kx_n - b)^2 = 0$ ; если нет возможности подбором  $k$  и  $b$  добиться равенства нулю этой суммы квадратов, то естественно попытаться минимизировать эту сумму.

миллиметровке или клетчатой бумаге строятся координатные оси, а затем отмечаются точки с координатами  $x_i, y_i$  ( $i=1, 2, \dots$ ); после этого передвижением линейки (желательно прозрачной) по бумаге подбирается такое ее положение, которое дает наиболее удачную прямую; проводится прямая, а потом выписываются соответствующие  $k$  и  $b$ . Сравнить результаты.

Обратим внимание на следующее. Алгоритм наименьших квадратов (в том варианте, который здесь изложен) позволяет приближенно заменить одной линейной функцией всю таблично заданную функцию, а алгоритм линейной интерполяции осуществляет такую замену только на интервале от одного табличного значения аргумента до другого; на следующем интервале линейная функция будет уже иной и т. д.

300. Написать программу получения  $k$  и  $b$ , исходя из данных  $x_1, y_1, \dots, x_n, y_n$  по алгоритму наименьших квадратов (см. предыдущую задачу). Применить эту программу к результатам эксперимента (описанного Д. И. Менделеевым) по исследованию растворимости  $\text{NaNO}_3$ : в 100 частях воды при температуре  $0^\circ, 4^\circ, 10^\circ, 15^\circ, 21^\circ, 29^\circ, 36^\circ, 51^\circ, 68^\circ$  смогло раствориться, соответственно, 66.7, 71, 76.3, 80.6, 85.7, 92.9, 99.4, 116.6, 125.1 частей  $\text{NaNO}_3$ . Получить коэффициенты линейной зависимости также с помощью миллиметровки или клетчатой бумаги и линейки (см. предыдущую задачу); сравнить результаты.

301. Написать программу, в результате выполнения которой по данным  $x_1, \dots, x_n, y_1, \dots, y_n$  вычисляются, в соответствии с алгоритмом наименьших квадратов, значения  $k$  и  $b$  (см. задачу 229); после этого вычисления рассматриваются значения аргумента, поочередно вводимые с клавиатуры, и для каждого из них вычисляется значение линейной функции (этот раздел программы должен быть устроен по образцу программы *синкор* или *синкор1* из § 16).

302. Алгоритм наименьших квадратов (см. задачу 299) может применяться для поиска линейной зависимости вида  $y=kx$ , где  $k$  — некоторое число (коэффициент пропорциональности). Пусть  $x_1,$

$\dots, x_n, y_1, \dots, y_n$  — данные числа и требуется определить  $k$  такое, что  $(y_1 - kx_1)^2 + (y_2 - kx_2)^2 + \dots + (y_n - kx_n)^2$  имеет наименьшее из возможных значений (рис. 62). Оказывается, что для этого достаточно взять

$$k = \frac{x_1 y_1 + \dots + x_n y_n}{x_1^2 + \dots + x_n^2}.$$

Задача поиска  $k$  возникает при вычислении по закону Ома сопротивления проводника, вычислении толщины одного листа бумаги

по результатам изменения толщины стопки листов (толщина стопки пропорциональна числу листов), вычислении толщины нити по результатам измерения длины участка карандаша, спирально обмотанного нитью (длина пропорциональна числу витков) и т. д.

Вернуться к задаче 301, предполагая, что зависимость между  $x$  и  $y$  имеет вид  $y=kx$ .

## § 21. Поиск элемента в упорядоченном массиве

При подготовке таблиц статистических сводок, справочных изданий, словарей как правило производят упорядочивание данных. Это делает работу с таблицей, справочником и т. д. более удобной. Например, выигравшие номера упорядочены в лотерейных таблицах по возрастанию. Упорядоченность облегчает проверку билета: если бы номера не были упорядочены, то проверка могла бы потребовать сравнения номера билета со всеми номерами, содержащимися в таблице. Точно так же словарный порядок облегчает поиск слова в словаре или справочнике.

Поиск сведений в разнообразных таблицах и сводках может быть возложен на компьютер. В этом параграфе мы будем заниматься алгоритмом быстрого поиска элемента в упорядоченной совокупности  $a_1, \dots, a_n$ . При этом будем считать, что  $a_1, \dots, a_n$  — целочисленный массив,  $b$  — некоторое целое число. Работая со словарем, мы имеем дело не с целыми числами, а со словами, но это не играет принципиальной роли: меняется только техника каждого сравнения, а не число сравнений.

Пусть  $a_1 < \dots < a_n$ ; рассмотрим задачу: выяснить, входит ли данное число  $b$  в массив  $a_1, \dots, a_n$ , и если входит, то каково значение  $p$ , для которого  $a_p = b$ ? Эту задачу мы назовем задачей поиска элемента. Тривиальный алгоритм решения этой задачи основывается на последовательных сравнениях  $b$  с элементами  $a_1, \dots, a_n$ : сравнить  $b$  с  $a_1$ , если они равны, то  $p=1$ , иначе сравнить  $b$  с  $a_2$  и т. д. Среднее число требуемых здесь сравнений можно считать равным  $n/2$ . Известен алгоритм, который требует гораздо меньших затрат.

Предположим, что в массиве  $a_1, \dots, a_n$  имеется элемент, равный  $b$ , т. е. существует такое  $p$ , что  $a_p = b$ . По результату любого сравнения  $a_s < b$  ( $1 \leq s \leq n$ ) мы сразу определяем, лежит ли  $p$  в диапазоне от 1 до  $s$  или же в диапазоне от  $s+1$  до  $n$ : второе будет иметь место, если неравенство  $a_s < b$  справедливо, а первое — если не справедливо. Если само  $s$  находится примерно посередине между 1 и  $n$ , то сравнение  $a_s < b$  сужает диапазон поиска примерно вдвое. Этот прием можно использовать многократно. Получается алгоритм, называемый алгоритмом деления пополам. В соответствии с этим алгоритмом надо взять первоначально 1

и  $n$  в качестве границ поиска индекса элемента; далее, до тех пор, пока границы не совпадут, шаг за шагом сдвигать эти границы следующим образом: сравнить  $b$  с  $a_s$ , где  $s$  — целая часть среднего арифметического границ \*); если  $a_s < b$ , то заменить прежнюю нижнюю границу на  $s+1$ , оставив верхнюю границу без изменения, иначе оставить без изменения нижнюю границу, а верхнюю заменить на  $s$ . Поиск закончится, когда границы совпадут.

Сказанное можно записать в виде последовательности операторов Паскаля ( $p$  и  $q$  обозначают упомянутые верхнюю и нижнюю границы; когда  $p$  и  $q$  совпадут, выполнение алгоритма закончится, и  $p$  дает результат выполнения):

```
p:=1; q:=n;
while p < q do
  begin s:=(p+q) div 2
    if a[s] < b then p:=s+1
    else q:=s
  end
```

Заметим сразу следующее. Мы исходим из предположения, что среди элементов  $a_1, \dots, a_n$  имеется такой, который равен  $b$ . Если заранее неизвестно, имеется или нет такой элемент, то, получив  $p$ , надо дополнительно проверить, действительно ли  $a_p = b$ . И если обнаружится, что равенство не справедливо, то из этого будет следовать, что среди  $a_1, \dots, a_n$  нет элемента, равного  $b$ .

Продemonстрируем применение этого алгоритма на конкретном примере. Пусть  $a_1, \dots, a_{10}$  соответственно равны 2, 3, 5, 7, 11, 13, 17, 19, 23, 29. Тогда для  $b=19$  поиск будет разворачиваться следующим образом. Вначале границы поиска равны 1 и 10. Имеем  $[(1+10)/2]=5$ , неравенство  $a_5 < b$ , т. е.  $11 < 19$ , справедливо, поэтому переходим к границам 6, 10. Имеем  $[(6+10)/2]=8$ , неравенство  $a_8 < b$ , т. е.  $19 < 19$ , не справедливо, поэтому переходим к границам 6, 8. Имеем  $[(6+8)/2]=7$ , неравенство  $a_7 < b$ , т. е.  $17 < 19$ , справедливо, поэтому переходим к границам 8, 8. Эти границы совпадают; следовательно, если в массиве вообще есть элемент со значением  $b$ , то это  $a_8$ . В данном случае равенство  $b=a_8$  справедливо, так как  $19=19$ .

Рис. 63 дает схематическое изображение процесса поиска. Цифра, надписанная над участком массива показывает, какое по счету деление пополам позволило перейти к этому участку.

Нетрудно убедиться, что если бы мы взяли в качестве  $b$  не 19, а 18, то до самого последнего момента применение алгоритма раз-

---

\*) Целая часть среднего арифметического двух натуральных чисел как раз и будет натуральным числом, лежащим примерно посередине между двумя данными числами.

ворачивалось бы точно так же, но в самом конце дополнительная проверка равенства  $a_8=b$  дала бы отрицательный результат, что явилось бы свидетельством того, что в массиве  $a_1, \dots, a_{10}$  нет элемента со значением 18. Отметим большое сходство алгоритма деления пополам, предназначенного для поиска элемента, и алгоритма деления пополам, предназначенного для приближенного вычисления корня уравнения (см. § 7)\*).

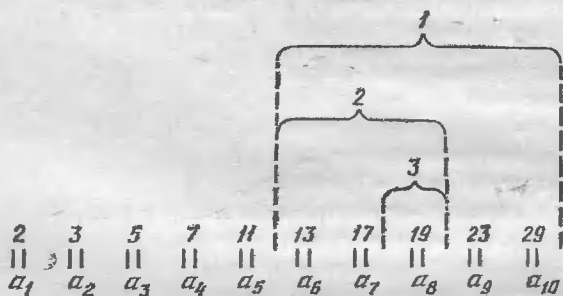


Рис. 63

Алгоритм поиска элемента делением пополам обладает высоким быстродействием. Если  $n=2^m$ , то число сравнений  $b$  с элементами  $a_1, \dots, a_n$  в процессе применения этого алгоритма не превзойдет  $m$ , а если  $n$  не равно  $2^m$  ни для какого целого  $m$ , то в качестве границы для числа сравнений можно взять наименьшее  $m$  такое, что  $n < 2^m$  (без учета дополнительного сравнения  $a_p=b$ , необходимого в том случае, когда заранее не известно, имеется или нет среди  $a_1, \dots, a_n$  элемент равный  $b$  \*\*). Например, если  $n=10$ , то число сравнений не превосходит 4, так как  $10 < 2^4$ , для  $n=32$  это число не превосходит 5, так как  $32=2^5$ , для  $n=1000$  это число не превосходит 10, так как  $1000 < 2^{10}$  и т. д.

Теперь мы приведем пример программы, использующей выписанную выше последовательность операторов, т. е. использующей алгоритм деления пополам.

Пусть таблица выигрышей лотереи представлена в виде двух массивов  $a_1, \dots, a_n$  и  $c_1, \dots, c_n$  ( $n$  — некоторая константа) так, что натуральные  $a_1, \dots, a_n$  — это выигравшие номера ( $a_1 < \dots < a_n$ ),

\*) Вообще идея деления пополам (сведение задачи к примерно вдвое более простой в количественном отношении) оказывается весьма плодотворной для построения многих алгоритмов. В научной литературе деление пополам иногда именуется греческим термином «дихотомия» или латинским «бисекция», а для обозначения сущности самой идеи иногда используются слова «разделяй и властвуй».

\*\*) Набросок доказательства см. в указании к задаче 311.



а  $c_1, \dots, c_n$  — действительные положительные числа, означающие выигрыши в рублях, выпавшие, соответственно, на номера  $a_1, \dots, a_n$ . Требуется найти выигрыши, выпавшие на ряд номеров (если номера нет в таблице, то выигрыш считается равным нулю). Будем пользоваться способом ведения диалога, продемонстрированным в зацикленной программе *синкор* из § 16 (от этого варианта без каких-либо трудностей можно перейти к другому варианту программы, аналогичному программе *синкор1* из § 16).

Схема программы:

```

program билеты(input, output);
  описания;
begin
  ввод массивов a и c;
  1: write('номер='); read(b);
     найти p — предполагаемое место b в массиве a;
     if a[p] = b then f := c[p] else f := 0;
     writeln('выигрыш =', f); goto 1
end.

```

Мы легко можем детализировать эту схему и получить настоящую программу. Условимся, что значения элементов массивов  $a$  и  $c$  заданы в следующей последовательности:  $a_1, c_1, a_2, c_2, \dots, a_n, c_n$ , где  $n$  — константа, значением которой будем считать 20:

```

program билеты(input, output);

label 1;
const n = 20;
type u = array[1..n] of integer; v = array[1..n] of real;
var a:u; c:v; i, p, q, s, b:integer; f:real;
begin for i := 1 to n do read(a[i], c[i]);
  1:   write('номер='); read(b);
      p := 1; q := n;
      while p < q do
        begin s := (p + q) div 2;
              if a[s] < b then p := s + 1
                else q := s
        end;
      if a[p] = b then f := c[p] else f := 0;
      writeln('выигрыш =', f); goto 1
end.

```

Идея деления пополам («разделяй и властвуй») может быть положена в основу алгоритма решения еще одной задачи, похожей на задачу поиска элемента. Пусть по-прежнему  $a_1 < a_2 < \dots < a_n$  и пусть  $b$  — некоторое число. Для числа  $b$  имеется  $n+1$  возможность:  $b \leq a_1, a_1 < b \leq a_2, a_2 < b \leq a_3, \dots, a_{n-1} < b \leq a_n, a_n < b$ .

Требуется определить, какая из возможностей имеет место. Ответом должно быть одно из чисел  $1, 2, \dots, n, n+1$  (порядковый номер возможности).

По результату любого сравнения  $a_s < b$ ,  $1 \leq s \leq n$  мы сразу определяем, лежит ли ответ в диапазоне от 1 до  $s$  или же в диапазоне от  $s+1$  до  $n+1$ . Если само  $s$  находится примерно посередине между 1 и  $n+1$ , то сравнение  $a_s < b$  сужает диапазон поиска примерно вдвое. Это приводит к следующему алгоритму:

```
p:=1; q:=n+1;
while p < q do
  begin s:=(p+q) div 2;
    if a[s] < b then p:=s+1
      else q:=s
  end
```

Единственное отличие последнего алгоритма от алгоритма поиска элемента состоит в начальном присваивании  $q:=n+1$  (вместо  $q:=n$ ). Этот алгоритм удобен, например, когда надо вставить в таблицу новый элемент, не нанося при этом ущерба упорядоченности таблицы. Этот алгоритм называется алгоритмом поиска места элемента (до него мы рассматривали алгоритм поиска элемента — названия похожи, но не совпадают).

## ЗАДАЧИ

303. Изменить программу *билеты*, используя как образец программу *синкор1* из § 16.

304. Изменить программу *билеты*: как только в качестве номера (т. е. значения переменной  $b$ ) с клавиатуры указывается отрицательное число, выполнение программы должно заканчиваться.

305. Изменить программу *билеты*: не нужно выводить все выигрыши по отдельности, но требуется вывести суммарный выигрыш, выпавший на все указанные номера. В качестве сигнала об окончании последовательности номеров используется любое отрицательное число (см. предыдущую задачу).

306. Изменить программу *билеты*, предполагая, что номера в таблице заданы в порядке убывания:  $a_1 > a_2 > \dots > a_n$ .

307. Даны натуральные  $a_1, \dots, a_{50}$ ,  $m$ ,  $b_1, \dots, b_m$  ( $a_1 > a_2 > \dots > a_{50}$ ). Подсчитать количество тех  $b_i$ ,  $1 \leq i \leq m$ , для которых нет равных среди  $a_1, \dots, a_{50}$ .

308. Составить программу определения задуманного человеком числа от 1 до 1000 с помощью 10 вопросов. Каждый вопрос имеет вид: «Верно ли, что задуманное число больше  $k$ ?» При этом указывается конкретное  $k$ . Ответы человека — это Д и Н. Применить идею деления пополам («разделяй и властвуй»).

309. Исследовать предложенную в предыдущей задаче игру в угадывание задуманного числа. Указать количество вопросов которое при «экономном» ведении игры будет достаточным для угадывания числа от 1 до соответственно 2000, 3000, 4000, 5000.

310. Доказать, что если  $n = 2^m$ , то число сравнений с элементами  $a_1, \dots, a_n$  в процессе применения алгоритма деления пополам равно  $m$ . (Без учета дополнительного сравнения  $a_p = b$ .)

311. Доказать, что если  $2^{m-1} < n \leq 2^m$ ,  $m \geq 1$ , то число сравнений  $b$  с элементами  $a_1, \dots, a_n$  в процессе применения алгоритма деления пополам равно  $m$ . (Без учета дополнительного сравнения  $a_p = b$ .)

Указание. Установить, что при  $m > 1$  первое сравнение приводит к поиску среди  $n'$  элементов, где  $n'$  таково, что выполнено  $2^{m-2} < n' \leq 2^{m-1}$ .

312. Воспользовавшись утверждением, сформулированным в предыдущей задаче, вычислить количество сравнений, которое потребуется произвести при  $n = 60, 70, 80, 90, 100, 110, 120, 130$ . (Без учета дополнительного сравнения  $a_p = b$ .)

313. Используя двоичный логарифм и целую часть, написать выражение  $m$  через  $n$ , вытекающее из утверждения задачи 311.

314. В конце предыдущего параграфа говорилось о применении линейной интерполяции в том случае, когда шаг таблицы не является постоянным. Использовать идею, заложенную в алгоритме поиска места элемента (алгоритм описан в конце текста настоящего параграфа) для поиска  $m$  такого, что данное  $t$  принадлежит отрезку  $[x_m, x_{m+1}]$  (предполагается, что  $x_1 \leq t \leq x_n$ ).

315. Даны целые  $a_1, \dots, a_n, b$  ( $n$  — некоторая константа,  $a_1 < a_2 < \dots < a_n$ ). Если среди чисел  $a_1, \dots, a_n$  есть равное  $b$ , то оставить  $a_1, \dots, a_n$  без изменений. В противном случае добавить  $b$  к  $a_1, \dots, a_n$  без нарушения упорядоченности по возрастанию. (Если данные для программы подготавливаются в виде  $b, a_1, \dots, a_n$ , т. е.  $b$  идет первым, и если не заботиться об экономии сравнений, то программа может быть составлена без привлечения массивов. Как это сделать?)

316. Изменить алгоритм поиска места элемента (см. конец текста настоящего параграфа), предполагая, что числа  $a_1, \dots, a_n$  упорядочены по убыванию:  $a_1 > a_2 > \dots > a_n$ .

317. В случае, если среди  $a_1, \dots, a_n$  допускаются равные ( $a_1 \leq a_2 \leq \dots \leq a_n$ ), то алгоритм поиска места элемента, записанный в виде последовательности операторов в конце текста настоящего параграфа, позволяет узнать самое первое из мест, на которое может быть вставлено число  $b$  в  $a_1, \dots, a_n$  без нарушения упорядоченности по неубыванию. Изменить последовательность операторов так, чтобы указывалось последнее из этих мест.

## § 22. Упорядочивание (сортировка) массива

Для программы линейной интерполяции, рассмотренной в § 20, существенна упорядоченность  $x_1, \dots, x_n$  — справедливость неравенств  $x_1 < \dots < x_n$ . Предполагается, что данные для программы заранее подготавливаются в упорядоченном виде. Задача упорядочивания возникает также при оформлении статистических сводок, справочных материалов и т. д. Процесс упорядочивания при большом объеме данных очень трудоемок и здесь часто прибегают к помощи компьютеров. Пока рассмотрим задачу упорядочивания в простейшей постановке: дан числовой массив  $x_1, \dots, x_n$ , элементы которого попарно различны; требуется переставить элементы массива так, чтобы после перестановки они были упорядочены в порядке возрастания:  $x_1 < \dots < x_n$ . Опишем один из алгоритмов решения этой задачи, который называется *алгоритмом сортировки выбором*.

Очевидно, что первое место в массиве должен занять наименьший элемент, второе место — наименьший из всех остальных элементов и т. д. Пусть  $x_1, \dots, x_{i-1}$  уже получили нужные значения. Тогда определение индекса  $k$  наименьшего элемента из  $x_i, x_{i+1}, \dots, x_n$  и перестановка  $x_i$  с  $x_k$  приведут к тому, что  $x_1, \dots, x_i$  будут иметь нужные значения. Таким образом, схема программы решения поставленной задачи может быть следующей:

```
program порядок(input, output);  
  описания;  
  begin  
    ввести массив x;  
    for i:=1 to n do  
      begin  
        найти индекс k наименьшего элемента  
        среди  $x_i, x_{i+1}, \dots, x_n$ ;  
        переставить  $x_i$  с  $x_k$ ;  
        writeln( $x_i$ )  
      end  
    end.
```

Как только элемент занял свое место, он сразу же выводится.

Приступим к детализации схемы. Ввод массива не доставляет затруднений:

```
for i:=1 to n do read( $x[i]$ )
```

и можно перейти к поиску индекса  $k$  наименьшего элемента среди  $x[i], \dots, x[n]$ :

```
k:= i;  
for j:= i+1 to n do  
  if  $x[j] < x[k]$  then k:= j
```

Для перестановки  $x[i]$  с  $x[k]$  привлекаем дополнительную переменную  $v$ :

$v := x[i]; x[i] := x[k]; x[k] := v$

Теперь можно написать программу (считаем элементы массива действительными числами):

```
program порядок(input, output);
  const n=20;
  type u=array[1..n] of real;
  var x: u; v: real; i, j, k: integer;
  begin
    for i:=1 to n do read(x[i]);
    for i:=1 to n do
      begin k:=i;
        for j:=i+1 to n do
          if x[j] < x[k] then k:=j;
        v:=x[i]; x[i]:=x[k]; x[k]:=v;
        writeln(x[i])
      end
    end.
```

При решении практических задач упорядочивание  $x_1, \dots, x_n$  как правило сопровождается некоторыми дополнительными действиями. Например, если  $x_1, y_1, x_2, y_2, \dots, x_n, y_n$  — это значения аргумента  $x$  и некоторой функции  $y=f(x)$ , то перестановка  $x_1, \dots, x_n$  должна сопровождаться перестановкой  $y_1, \dots, y_n$ . Элементы  $y_1, \dots, y_n$  переставляются так же, как  $x_1, \dots, x_n$  вне зависимости от значений самих  $y_1, \dots, y_n$ . Рассмотрим эту задачу; переставленные значения  $x_1, y_1, \dots, x_n, y_n$  будут выведены в два столбца:

$x_1$	$y_1$
$x_2$	$y_2$
...	...
$x_n$	$y_n$

Программа:

```
program табл(input, output);
  const n=20;
  type u=array[1..n] of real;
  var x, y: u; v: real; i, j, k: integer;
  begin
    for i:=1 to n do read(x[i], y[i]);
    for i:=1 to n do
      begin k:=i;
        for j:=i+1 to n do
          if x[j] < x[k] then k:=j;
```

```

v := x[i]; x[i] := x[k]; x[k] := v;
v := y[i]; y[i] := y[k]; y[k] := v;
writeln(x[i], y[i])

```

end

end.

## ЗАДАЧИ

318. Изменить программу *порядок* таким образом, чтобы в ходе ее выполнения элементы массива упорядочивания по убыванию:  $x_1 > x_2 > \dots > x_n$ .

319. Откажемся от предположения о том, что элементы  $x_1, \dots, x_n$  попарно различны. Пригодна ли программа *порядок* для упорядочивания массива в порядке неубывания:  $x_1 \leq x_2 \leq \dots \leq x_n$ ? Если нет, то внести необходимые изменения.

320. Для упорядочивания массива попарно различных чисел  $x_1, \dots, x_n$  может быть использован алгоритм *сортировки обмeнами*. Опишем этот алгоритм применительно к упорядочиванию по возрастанию. Последовательным просмотром  $x_1, \dots, x_n$  найти  $i$  такое, что  $x_i > x_{i+1}$ ; поменять  $x_i$  и  $x_{i+1}$  местами, продолжить просмотр с элемента  $x_{i+1}$  и т. д. Тем самым в результате первого просмотра наибольшее число передвинется на последнее место. Следующие просмотры начинать опять сначала, уменьшая на единицу количество просматриваемых элементов. Массив будет упорядочен после просмотра, в котором участвовали только первый и второй элементы. Написать программу, реализующую алгоритм сортировки обмeнами.

321. Для упорядочивания массива попарно различных чисел  $x_1, \dots, x_n$  может быть использован алгоритм *сортировки простыми вставками*. Опишем этот алгоритм применительно к упорядочиванию по возрастанию. Просматривать последовательно  $x_2, \dots, x_n$  и каждый новый элемент  $x_i$  вставлять на подходящее место в уже упорядоченную совокупность  $x_1, \dots, x_{i-1}$ . Это место определяется последовательным сравнением  $x_i$  с упорядоченными элементами  $x_1, \dots, x_{i-1}$ . Написать программу, реализующую алгоритм сортировки вставками.

322. Исследовать число сравнений и число перемещений (т. е. перестановок с одного места на другое) элементов  $x_1, \dots, x_n$  в процессе применения алгоритмов сортировки выбором, сортировки обмeнами и сортировки простыми вставками (см. задачи 320, 321). Показать, что число перемещений для алгоритма сортировки выбором ограничено некоторой линейной функцией от  $n$ . В это же время и число сравнений для алгоритма сортировки выбором, и обе указанные характеристики для алгоритмов сортировки обмeнами и сортировки простыми вставками в некоторых

случаях (например, когда исходные данные таковы, что  $x_1 > x_2 > \dots > x_n$ ) являются квадратичными функциями от  $n$ .

323. Алгоритм сортировки простыми вставками (см. задачу 321) можно изменить следующим образом. Место, на которое надо вставить  $x_i$  в уже упорядоченную совокупность  $x_1, \dots, x_{i-1}$ , определяется алгоритмом деления пополам (см. алгоритм поиска места элемента в конце предыдущего параграфа). Получается новый алгоритм сортировки, который называется алгоритмом *сортировки бинарными вставками* (слова «бинарная вставка» следует понимать как «вставка делением пополам»). Написать программу, реализующую этот алгоритм.

По числу сравнений алгоритм сортировки бинарными вставками намного лучше, чем рассмотренные выше алгоритмы. Различными авторами предпринимались попытки доказательства оптимальности алгоритма сортировки бинарными вставками и даже печатно сообщалось о якобы найденном доказательстве. Позднее выяснилось, что и этот алгоритм не оптимален, так как он требует восемь сравнений для упорядочивания пяти чисел, а на самом деле достаточно семи сравнений (см. следующую задачу).

324. То, что для упорядочивания пяти чисел достаточно семи сравнений—это не простой факт, хотя он и допускает формулировку в виде задачи на взвешивания. Имеются весы без гирь и пять различных грузов. На каждую из двух чашек весов можно класть по одному грузу. Расположить грузы по убыванию веса, используя семь взвешиваний.

325. Даны целые  $x_1, \dots, x_n$ . Получить в порядке возрастания все различные числа, входящие в  $x_1, \dots, x_n$ . Воспользоваться алгоритмом сортировки бинарными вставками (см. задачу 6). В процессе сортировки следует отбрасывать элементы, уже встречавшиеся раньше. Если в результате поиска места  $x_i$  в упорядоченной совокупности  $x_1, \dots, x_k$  ( $k < i$ ) обнаружится, что среди  $x_1, \dots, x_k$  есть элемент, равный  $x_i$ , то следует перейти к рассмотрению  $x_{i+1}$  без изменения  $x_1, \dots, x_k$ .

326. Даны действительные  $c_1, \dots, c_p, d_1, \dots, d_q$  ( $c_1 \leq c_2 \leq \dots \leq c_p, d_1 \leq d_2 \leq \dots \leq d_q$ ). Внести единую упорядоченность в  $c_1, \dots, c_p, d_1, \dots, d_q$ , получив  $f_1, f_2, \dots, f_{p+q}$  такие, что  $f_1 \leq f_2 \leq \dots \leq f_{p+q}$ . Число сравнений не должно превосходить  $p+q$ .

327. Алгоритм, сформулированный в задаче 320,—это, строго говоря, лишь один из алгоритмов сортировки обменами. Он иногда называется алгоритмом *пузырька* (наибольший элемент, продвигающийся на свое место, этим названием уподобляется пузырьку воздуха, поднимающемуся со дна к поверхности воды). Есть и другие алгоритмы, которые естественно отнести к алгоритмам сортировки обменами. Приведем пример такого алгоритма. Последовательными

просмотрами чисел  $a_1, \dots, a_n$  найти наименьшее  $i$  такое, что  $a_i > a_{i+1}$ . Поменять  $a_i$  и  $a_{i+1}$  местами и возобновить просмотр с начала массива. Когда не удастся найти  $i$ , массив будет упорядочен.

Написать программу, реализующую этот алгоритм.

328. Даны действительные попарно различные  $a_1, \dots, a_n$ . Получить попарно различные целые  $j_1, \dots, j_n$  такие, что  $1 \leq j_k \leq n$  ( $k=1, \dots, n$ ) и  $a_{j_1} < a_{j_2} < \dots < a_{j_n}$ .

Указание. Рассмотреть, что будет, если для программы таблицы из настоящего параграфа взять в качестве исходных значений  $y_1, \dots, y_n$  числа  $1, \dots, n$ .

329. Даны целые  $a_1, \dots, a_n$ . Найти наибольшее значение, содержащееся в последовательности чисел  $a_1, \dots, a_n$  после выбора из нее:

а) одного из членов со значением  $\max(a_1, \dots, a_n)$ ;

б) всех членов со значением  $\max(a_1, \dots, a_n)$ .

330. Рассмотрим два алгоритма выбора наименьшего из  $n$  неравных чисел. Первый алгоритм: найти меньшее из первых двух чисел и, сравнив его с третьим, опять взять меньшее, сравнить с четвертым и т. д. Второй алгоритм («кубковая» система): разбить числа на пары, меньшее из каждой пары пропустить в следующий тур. Если одному из чисел не нашлось парного, то это число тоже проходит в следующий тур. В следующем туре повторить то же самое и т. д., пока не останется одно число. Какой из этих алгоритмов требует меньше сравнений?

331. Доказать, что не существует алгоритма выбора наименьшего из  $n$  неравных чисел, который требовал бы менее  $(n-1)$ -го сравнения.

Указание. Утверждать, что  $a_i$  не является наименьшим можно только после того, как число  $a_i$  участвовало в некотором сравнении и оказалось большим.

332. Даны натуральные  $n, a_1, \dots, a_n$  ( $n \geq 4$ ). Числа  $a_1, \dots, a_n$  — это измеренные в сотых долях секунды результаты  $n$  спортсменов в беге на 100 м. Составить команду из четырех лучших бегунов для участия в эстафете  $4 \times 100$ , т. е. указать одну из четверок натуральных чисел  $i, j, k, l$  такую, что  $1 \leq i < j < k < l \leq n$  и  $a_i + a_j + a_k + a_l$  имеет наименьшее значение.



## ГЛАВА IV

### МАТРИЦЫ

#### § 23. Массивы массивов. Матрицы

В описании регулярного типа

$u = \text{array } [n_1..n_2] \text{ of } r$

в качестве базового типа  $r$  может выступать не только один из стандартных типов, т. е. *real*, *integer* или *char*, но и ранее описанный в программе тип. Пусть тип  $r$ , в свою очередь, был описан как регулярный:

$r = \text{array } [m_1..m_2] \text{ of integer}$

и пусть переменная  $a$ —это переменная типа  $u$ . Тогда  $a[i]$ —это переменная типа  $r$ , а  $a[i][j]$ —переменная типа *integer*.

Переменные с двумя индексами удобны для работы с таблицами. Таблица круга футбольного чемпионата после отбрасывания названий команд и после заполнения неиспользованных диагональных клеток нулями представляет собой квадратную числовую таблицу, обладающую тем свойством, что, во-первых, каждое число—это 0, 1 или 2 и, во-вторых, сумма числа очков, набранных  $i$ -й командой в игре с  $j$ -й, и числа очков, набранных  $j$ -й командой в игре с  $i$ -й, при  $i \neq j$  равна 2 (рис. 64). В программе

0	1	0	1
1	0	2	2
2	0	0	1
1	0	1	0

Рис. 64

удобно эти числа очков обозначить через  $a[i][j]$  и  $a[j][i]$ . К этому обозначению можно прийти так:  $a$ —это таблица (массив строк),  $a[i]$ — $i$ -я строка таблицы (массивов чисел),  $a[i][j]$ — $j$ -й элемент  $i$ -й строки (число).

Напишем программу проверки равенства двум каждой из сумм вида  $a[i][j] + a[j][i]$  при  $i \neq j$ . Предполагается, что таблица имеет размер  $15 \times 15$  и что данная последовательность целых чисел

$c_1, \dots, c_{225}$  — это выписанные одна за другой строки таблицы:  
 $c_1, \dots, c_{15}$  — первая строка,  $c_{16}, \dots, c_{30}$  вторая строка и т. д.

```

program kryz(input, output);
  label 1;
  type строка=array[1..15] of integer;
  таблица=array[1..15] of строка;
  var a: таблица; i, j: integer;
  begin
    for i:=1 to 15 do
      for j:=1 to 15 do read(a[i][j]);
    for i:=1 to 14 do
      for j:=i+1 to 15 do
        if a[i][j]+a[j][i] < > 2
          then begin
            write('есть ошибка');
            goto 1
          end;
        write('ошибок нет');
  1: end.

```

Здесь сделано так: после того как таблица введена, с помощью двойного цикла организуется перебор всех  $i, j$  таких, что  $1 \leq i < j \leq 15$ , т. е. перебор всех элементов, находящихся в правой верхней части таблицы. Эти элементы складываются с соответствующими им элементами левой нижней части.

В научной литературе квадратные и прямоугольные таблицы часто называют матрицами, при этом элементы матриц изображаются с помощью двух опущенных индексов. Как именно это делается, видно на примерах трех следующих матриц:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \quad \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \end{pmatrix}, \\
 \begin{pmatrix} c_{11} & \dots & c_{1n} \\ \dots & \dots & \dots \\ c_{m1} & \dots & c_{mn} \end{pmatrix}.$$

Первый индекс — номер строки, второй — номер столбца;  $a_{13}$  читается «а один три», а не «а тринадцать», этот элемент расположен в первой строке и в третьем столбце первой матрицы;  $c_{mn}$  — это «эм эн», т. е. элемент, расположенный в строке с номером  $m$  и в столбце с номером  $n$  третьей матрицы. В программах на Паскале используются, соответственно, конструкции  $a[1][3]$ ,  $c[m][n]$  и т. д.

Про матрицу, имеющую  $m$  строк и  $n$  столбцов, говорят, что она имеет размер  $m \times n$  («эм на эн»). Если  $m = n$ , то матрица называется квадратной.

В тех программах, которые мы будем составлять, предполагается, что матрицы задаются и выводятся построчно: вначале элементы первой строки, затем — второй и т. д. Именно так была подготовлена исходная матрица для программы *круг*.

Пусть дана квадратная матрица размера  $n \times n$  ( $n$  — некоторая константа), состоящая из действительных чисел, и пусть надо преобразовать матрицу: поэлементно вычесть последний столбец из всех столбцов, кроме последнего (столбец с номером  $j$  — это элементы  $a[1][j]$ ,  $a[2][j]$ , ...,  $a[n][j]$ ).

При составлении программы будем считать, что значение константы  $n$  равно 5:

```

program столбцы(input, output);
  const n=5;
  type строка=array[1..n] of real;
       матрица=array[1..n] of строка;
  var a: матрица; i j: integer;
  begin
    for i:=1 to n do
      for j:=1 to n do read(a[i][j]);
    for j:=1 to n-1 do
      for i:=1 to n do a[i][j]:=a[i][j]-a[i][n];
    for i:=1 to n do
      for j:=1 to n do write(a[i][j])
  end.

```

Следует помнить, что исходные данные для программы — это не таблицы и не массивы, а последовательность значений стандартных типов, которая может быть прочитана элемент за элементом с помощью оператора ввода *read*. Каким именно переменным (с индексами, без индексов и т. д.) будут присвоены эти значения, целиком зависит от того, как составлена программа.

**Пример.** Дана символьная матрица размера  $m \times n$  ( $m$  и  $n$  — некоторые константы). Требуется получить последовательность  $b_1, \dots, b_n$  из нулей и единиц, в которой  $b_i=1$  тогда и только тогда, когда в  $i$ -м столбце число символов \* не меньше числа пробелов.

Схема программы выписывается без труда:

```

program пробелы(input, output);
  описания;
  begin
    ввести матрицу( $s_{ij}$ );
    for j:=1 to n do
      begin
        подсчитать  $k$ —число звездочек и  $l$ —число про-
        белов среди  $s_{1j}, s_{2j}, \dots, s_{mj}$ ;

```

```

        if  $k \geq l$  then writeln(1) else writeln(0)
    end
end.

```

(Массив  $b_1, \dots, b_n$  здесь не используется — числа  $b_1, \dots, b_n$  получают одно за другим и сразу же выводятся.) Подсчет числа звездочек и пробелов задается операторами:

```

k := 0; l := 0;
for i := 1 to m do
    if s[i][j] = '*' then k := k + 1 else
        if s[i][j] = ' ' then l := l + 1

```

Получается программа ( $m$  и  $n$  считаем равными 4 и 5):

```

program пробелы(input, output);
    const m = 4; n = 5;
    type строка = array[1..n] of char;
    матрица = array[1..m] of строка;
    var s: матрица; i, j, k, l: integer;
    begin
        for i := 1 to m do
            for j := 1 to n do read(s[i][j]);
        for j := 1 to n do
            begin k := 0; l := 0;
                for i := 1 to m do
                    if s[i][j] = '*' then k := k + 1 else
                        if s[i][j] = ' ' then l := l + 1;
                    if  $k \geq l$  then write(1) else write (0)
                end
            end.

```

Последний пример. Дана действительная матрица размера  $n \times n$ , все элементы которой различны. Требуется упорядочить (переставить) строки матрицы по возрастанию первых элементов строк. Взяв за образец схему программы *порядок* из § 22, получаем следующее:

```

program порядок1(input, output);
    описания;
    begin
        ввести матрицу( $x_{ij}$ );
        for i := 1 to n do
            begin
                среди строк с номерами  $i, i+1, \dots, n$  найти
                такую (точнее её номер  $k$ ), которая имеет наи-
                меньший первый элемент;
                переставить строки с номерами

```

*i и k; вывести l-ю строку*

*end*

*end.*

Поиск числа  $k$  — номера строки с наименьшим первым элементом задается операторами

```

k := i;
for j := i + 1 to n do
  if x[j][1] < x[k][1] then k := j

```

Для перестановки строк с номерами  $i$  и  $k$  можно было бы воспользоваться оператором цикла

```

for l := 1 to n do
  begin v := x[i][l]; x[i][l] := x[k][l];
        x[k][l] := v
  end

```

Паскаль представляет программисту очень удобную возможность, которой здесь можно воспользоваться. Для переменных  $a$  и  $b$  одного и того же регулярного типа допустим оператор присваивания  $a := b$ . Поэтому перестановка строк может быть задана операторами

```
v := x[i]; x[i] := x[k]; x[k] := v
```

где  $v$  — переменная того же регулярного типа, что и переменные  $x[k]$ ,  $x[i]$ , ... . Напомним, что значениями переменных  $x[k]$ ,  $x[i]$ , ... являются строки матрицы (числовые массивы).

Итак, программа:

```

program порядок1(input, output);
const n = 5;
type строка = array[1..n] of real;
  матрица = array[1..n] of строка;
var v: строка; x: матрица; i, j, k: integer;
begin
  for i := 1 to n do
    for j := 1 to n do read(x[i][j]);
  for i := 1 to n do
    begin k := i;
      for j := i + 1 to n do
        if x[j][1] < x[k][1] then k := j;
      v := x[i]; x[i] := x[k]; x[k] := v;
      for j := 1 to n do write(x[i][j])
    end
  end.

```

Если снять условие, что все элементы матрицы попарно различны, то предложенная программа обеспечит такую перестановку строк, после которой строки расположены по неубыванию их первых элементов.

## ЗАДАЧИ

333. Изменить программу *круг* так, чтобы число строк и столбцов данной квадратной матрицы  $(a_{ij})$  определялось константой  $n$ .

334. Изменить программу *столбцы* так, чтобы:

а) число строк в матрице  $(a_{ij})$  могло отличаться от числа столбцов. Число строк определяется константой  $n$ , число столбцов — константой  $m$ ;

б) элементы каждой строки матрицы выводились с новой строки экрана.

335. Изменить программу *пробелы* так, чтобы:

а) программа была бы применима только к квадратным матрицам размера  $n \times n$ . В программе разрешается использовать лишь одну константу  $n$ ;

б) способ сравнения числа звездочек с числом пробелов сводился к изменению одной целочисленной переменной  $k$  (в зависимости от значения переменной  $a[i][j]$  следует либо увеличивать  $k$  на 1, либо уменьшать  $k$  на 1, либо оставлять  $k$  без изменения) и к последующему сравнению  $k$  с нулем;

в) число пробелов сравнивалось не с числом звездочек, а с числом латинских букв (малых и больших).

336. Пусть дана действительная квадратная матрица размера  $n \times n$ . Требуется преобразовать матрицу: поэлементно вычесть последнюю строку из всех строк, кроме последней.

337. Дана целочисленная квадратная матрица  $(a_{ij})$  размера  $7 \times 7$ . Получить  $b_1, \dots, b_7$ , где  $b_i$  — это

а) наименьшее из значений элементов, находящихся в начале  $i$ -й строки матрицы до элемента  $a_{ii}$  включительно;

б) значение первого по порядку положительного элемента  $i$ -й строки (если таких элементов нет, то принять  $b_i = -1$ ).

338. Дана целочисленная квадратная матрица размера  $n \times n$ . Найти номера столбцов:

а) все элементы которых — нули;

б) элементы в каждом из которых одинаковы;

в) элементы каждого из которых образуют возрастающую последовательность  $(b_{1i} < b_{2i} < \dots < b_{ni})$ .

339. В данной целочисленной квадратной матрице размера  $n \times n$  ( $n$  — некоторая константа) указать индексы всех элементов, имеющих наибольшее значение.

340. Дана символьная матрица размера  $5 \times 6$ . Найти:

а) номер первого по порядку столбца, содержащего наибольшее число цифр;

б) номер последнего по порядку столбца, содержащего наименьшее число букв ш и щ.

341. С помощью действительной матрицы

$$\begin{pmatrix} x_{11} & \dots & x_{1n} \\ x_{21} & \dots & x_{2n} \end{pmatrix}$$

на плоскости задано  $n$  точек так, что  $x_{1j}, x_{2j}$  — координаты  $j$ -й точки. Точки попарно соединены отрезками. Найти длину наибольшего отрезка.

342. Дана действительная квадратная матрица размера  $n \times n$ . Получить  $x_1x_n + x_2x_{n-1} + \dots + x_nx_1$ , где  $x_k$  — наибольшее значение элементов  $k$ -го столбца данной матрицы.

343. Транспонированием квадратной матрицы называется такое ее преобразование, при котором строки и столбцы меняются ролями:  $i$ -й столбец становится  $i$ -й строкой. Например, транспонирование матрицы

$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$$

дает матрицу

$$\begin{pmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{pmatrix}$$

Дана квадратная матрица размера  $n \times n$ . Получить транспонированную матрицу.

344. Таблица круга футбольного чемпионата, в котором участвовало  $n$  команд, задана своей верхней правой частью: первые  $n$  чисел данной последовательности  $c_1, c_2, \dots$  относятся к первой строке таблицы, следующие  $n-1$  чисел — ко второй и т. д. Построить всю таблицу целиком.

345. Таблица круга футбольного чемпионата задана квадратной матрицей размера  $n \times n$ .

а) Найти число команд, имеющих больше побед, чем поражений.

б) Определить номера команд, прошедших чемпионат без поражений.

в) Выяснить, имеется ли хотя бы одна команда, выигравшая более половины игр.

346. Дана действительная матрица  $(x_{ij})$  размера  $m \times n$ ; упорядочить (переставить) строки матрицы:

а) по убыванию сумм элементов строк;

- б) по неубыванию наименьших элементов строк;  
 в) по невозрастанию наибольших элементов строк.

**Указание.** Определить числовой массив  $b_i, \dots, b_m$ , положив  $b_i$  равным, соответственно, сумме элементов  $i$ -й строки, наименьшему элементу  $i$ -й строки, наибольшему элементу  $i$ -й строки. Можно вместо массива  $b_i, \dots, b_m$  рассмотреть дополнительный столбец  $x_{1\ n+1}, x_{2\ n+1}, \dots, x_{m\ n+1}$ .

347. Шахматную доску будем представлять символьной матрицей размера  $8 \times 8$ . Даны натуральные  $p$  и  $q$  ( $1 \leq p \leq 8, 1 \leq q \leq 8$ ) — номера вертикали и горизонтали, определяющие местоположение ферзя. Соответствующий элемент матрицы надо положить равным символу Ф. Поля, находящиеся под угрозой ферзя, надо положить равными символу \*, а остальные поля — символу 0. Строки матрицы вывести одну под другой. Решить аналогичную задачу для коня.

348. Дано: натуральные  $x_1, y_1, \dots, x_{10}, y_{10}$ , целочисленная матрица  $(a_{ij})$  ( $i=1, \dots, 10, j=1, \dots, 10$ ). Последовательность  $x_1, y_1, \dots, x_{10}, y_{10}$  задает положение десяти точек на экране. Матрица указывает, как точки соединены между собой:  $a_{ij}=1$ , если  $i$ -я точка соединена с  $j$ -й и  $a_{ij}=0$  в противном случае ( $a_{ij}=a_{ji}$ ). Получить на экране точки, заданные последовательностью  $x_1, y_1, \dots, x_{10}, y_{10}$  и соединить их так, как указано в данной матрице.

## § 24. Решение систем линейных уравнений

В школьных курсах математики и физики встречаются задачи, которые сводятся к решению систем линейных уравнений. Эти системы состоят, как правило, из двух или трех уравнений:

$$\begin{cases} 1.2x - 3y = 1.7, \\ 0.5x + y = -1.9, \end{cases} \quad \begin{cases} 2x_1 - 4x_2 + 4x_3 = 6, \\ -5x_1 + 12x_2 - 14x_3 = -35, \\ 6x_1 - 7x_2 + 5x_3 = 10 \end{cases}$$

и т. д. Серьезные практические задачи, возникающие в физике, экономике, технике и т. д., часто приводят к таким системам, которые содержат сотни и даже тысячи линейных уравнений со столь же большим числом неизвестных. Не прибегая к помощи компьютера, эти системы решить невозможно. Рассмотрим проблему решения систем линейных уравнений более подробно.

Напомним, что линейным уравнением с неизвестными  $x_1, \dots, \dots, x_n$  называется уравнение вида

$$a_1x_1 + a_2x_2 + \dots + a_nx_n = b,$$

при этом числа  $a_1, a_2, \dots, a_n$  называются коэффициентами уравнения, а  $b$  — его правой частью. Общий вид системы  $n$  линейных





Получили систему, равносильную исходной:

$$\begin{aligned}x_1 - 2x_2 + 2x_3 &= 3, \\2x_2 - 4x_3 &= -20, \\5x_2 - 7x_3 &= 8.\end{aligned}$$

В этой новой системе второе и третье уравнения содержат только неизвестные  $x_2$  и  $x_3$ . Делением второго уравнения на 2 получаем уравнение, в котором коэффициент при  $x_2$  равен 1:

$$x_2 - 2x_3 = -10.$$

Домножая это уравнение на  $-5$  и затем прибавляя его к третьему уравнению, получаем

$$3x_3 = 42.$$

Вся система приобретает вид

$$\begin{aligned}x_1 - 2x_2 + 2x_3 &= 3, \\x_2 - 2x_3 &= -10, \\3x_3 &= 42.\end{aligned}$$

Разделив последнее уравнение на 3, получаем треугольную систему

$$\begin{aligned}x_1 - 2x_2 + 2x_3 &= 3, \\x_2 - 2x_3 &= -10, \\x_3 &= 14,\end{aligned}$$

в которой коэффициенты на диагонали равны 1. Эта система легко решается последовательным рассмотрением уравнений от последнего к первому:

$$\begin{aligned}x_3 &= 14, \\x_2 &= -10 + 2x_3 = 18, \\x_1 &= 3 + 2x_2 - 2x_3 = 11.\end{aligned}$$

Получим

$$x_1 = 11, \quad x_2 = 18, \quad x_3 = 14.$$

В общем случае алгоритм Гаусса предписывает приведение системы  $n$ -го порядка к равносильной ей треугольной системе с единичными коэффициентами на диагонали:

$$\begin{array}{rcl} x_1 + a'_{12}x_2 + a'_{13}x_3 + \dots + a'_{1n}x_n & = & b'_1 \\ x_2 + a'_{23}x_3 + \dots + a'_{2n}x_n & = & b'_2 \\ . & . & . \\ . & . & . \\ . & . & . \\ & & x_n = b'_n \end{array}$$

(этот этап называется *прямым ходом алгоритма Гаусса*), а затем (обратный ход) — вычисление значения неизвестных, начиная с  $x_n$  и кончая  $x_1$ . Приведение исходной системы путем последовательного исключения неизвестных к равносильной треугольной системе с единичными коэффициентами на диагонали возможно тогда и только тогда, когда исходная система совместна и обладает единственным решением  $x_1 = \alpha_1, x_2 = \alpha_2, \dots, x_n = \alpha_n$  (доказательство мы не приводим).

Написание программы, реализующей алгоритм Гаусса, можно начать с весьма грубой схемы:

```
program Гаусс(input, output);  
  описания;  
  begin  
    ввести коэффициенты и правые части уравнений;  
    выполнить прямой ход;  
    выполнить обратный ход;  
    вывести результаты  
  end.
```

Наиболее сложная часть программы, это, конечно, прямой ход, который представляет собой серию однотипных действий: получить первое уравнение в виде, удобном для исключения  $x_1$ , и исключить  $x_1$  из уравнений с номерами 2, ...,  $n$ ; получить второе уравнение в виде, удобном для исключения  $x_2$ , и исключить  $x_2$  из уравнений с номерами 3, ...,  $n$  и т. д.:

```
for i := 1 to n do  
  begin  
    получить i-е уравнение в виде, удобном  
    для исключения  $x_i$ ;  
    исключить  $x_i$  из уравнений с номерами  $i+1, \dots, n$   
    с помощью i-го уравнения  
  end.
```

В разобранный пример получения  $i$ -го уравнения в виде, удобном для исключения  $x_i$ , состояло в делении  $i$ -го уравнения на коэффициент при  $x_i$ . Но этот коэффициент может случайно оказаться равным нулю, или же оказаться столь малым по модулю, что деление на него даст число, выходящее из диапазона представимых на данном компьютере чисел. Правильнее будет перед исключением  $x_i$  сначала выбрать из уравнений с номерами  $i, i+1, \dots, n$  то, которое имеет наибольший по модулю коэффициент при  $x_i$  и поменять его местами с  $i$ -м уравнением, а затем уже выполнить деление на коэффициент при  $x_i$ . Этап получения  $i$ -го уравнения в виде, удобном для исключения  $x_i$ , запишется так:

выбрать из уравнений с номерами  $i, i+1, \dots, n$  то, в котором коэффициент при  $x_i$  имеет наибольшее по модулю значение (точнее — найти номер  $k$  этого уравнения);  
если  $k \neq i$ , то поменять местами  $k$ -е и  $i$ -е уравнения;  
разделить  $i$ -е уравнение на его коэффициент при  $x_i$ .

Мы теперь можем дать более подробную схему всей программы Гаусс (попутно несколько детализируем этап обратный ход):

```
program Гаусс (input, output);  
  описания;
```

*begin*

*ввести коэффициенты и правые части уравнений;*

*for i := 1 to n do*

*begin*

*выбрать из уравнений с номерами i, i+1, ..., n*  
*то, в котором коэффициент при x<sub>i</sub> имеет*  
*наибольшее по модулю значение (точнее —*  
*найти номер k этого уравнения);*

*если k ≠ i, то поменять местами i-е и k-е*  
*уравнения;*

*разделить i-е уравнение на его*

*коэффициент при x<sub>i</sub>;*

*for l := i+1 to n do*

*исключить x<sub>i</sub> из l-го уравнения с помощью*  
*i-го уравнения;*

*end;*

*получить x<sub>n</sub>;*

*for i := n-1 downto 1 do вычислить x<sub>i</sub>;*

*вывести x<sub>1</sub>, ..., x<sub>n</sub>*

*end.*

Заметим, что действия над уравнениями, заключающиеся в перестановке уравнений, умножении или делении уравнения на некоторое число, в почленном прибавлении одного уравнения к другому и т. д., удобнее описывать, если исходная система представлена в программе не квадратной матрицей коэффициентов и столбцом правых частей, а одной только матрицей размера  $n \times (n+1)$ :

$$\begin{pmatrix} a_{11} & \dots & a_{1n} & a_{1\ n+1} \\ a_{21} & \dots & a_{2n} & a_{2\ n+1} \\ \dots & \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} & a_{n\ n+1} \end{pmatrix},$$

в которой первые  $n$  столбцов — это столбцы матрицы коэффициентов, а последний столбец — это столбец правых частей. Это так называемая расширенная матрица системы. Для системы третьего порядка, на примере которой был продемонстрирован алгоритм Гаусса, расширенная матрица имеет вид

$$\begin{pmatrix} 2 & -4 & 4 & 6 \\ -5 & 12 & -14 & -35 \\ 6 & -7 & 5 & 10 \end{pmatrix}$$

При такой форме представления данных не надо отдельно описывать действия над правыми частями  $b_i$  ( $i=1, \dots, n$ ). Приняв эту форму, можно полностью детализировать этапы, связанные с прямым и обратным ходом. Но предварительно договоримся о следующем. Во-первых: в процессе выполнения программы коэффи-

иенты системы изменяются, но продолжают занимать места в той же самой расширенной матрице (т. е. по-прежнему являются значениями тех же самых переменных вида  $a[i][j]$ ). Во-вторых: если из уравнения с номером  $l$  уже исключены неизвестные  $x_1, \dots, x_i$ , то, хотя переменные  $a[l][1], a[l][2], \dots, a[l][i]$  и имеют какие-то числовые значения, мы с этими значениями уже не связываем никакого смысла; после завершения прямого хода эти числовые значения заполняют весь левый нижний угол матрицы (под диагональю из единичных коэффициентов).

Опишем со всеми подробностями этап выбора уравнения с наибольшим по модулю коэффициентом при  $x_i$ :

```
maxabs := abs(a[i][i]); k := i;
for l := i + 1 to n do
    if abs(a[l][i]) > maxabs then
        begin maxabs := abs(a[l][i]); k := l end
```

А теперь — этап перестановки  $i$ -го и  $k$ -го уравнений:

```
for j := i to n + 1 do
    begin v := a[i][j]; a[i][j] := a[k][j];
        a[k][j] := v
    end
```

(здесь мы пишем  $for j := i to n + 1 do \dots$ , так как работаем с расширенной матрицей, которая имеет  $n + 1$  столбец). Далее идет этап деления  $i$ -го уравнения на коэффициент при  $x_i$ :

```
v := a[i][i];
for j := i to n + 1 do a[i][j] := a[i][j]/v
```

и этап исключения  $x_i$  из  $l$ -го уравнения с помощью  $i$ -го:

```
v := a[l][i]
for j := i + 1 to n + 1 do
    a[l][j] := a[l][j] - a[i][j]*v
```

Этим завершается набор заготовок для описания прямого хода алгоритма Гаусса. Значение  $x_n$ , как говорилось, определяется уже на этапе прямого хода, и для присваивания  $x[n]$  нужного значения достаточно выполнить оператор  $x[n] := a[n][n + 1]$ . Значения остальных неизвестных вычисляются, начиная с  $(n - 1)$ -й и кончая первой:

```
for i := n - 1 downto 1 do
    begin x[i] := a[i][n + 1];
        for j := i + 1 to n do
            x[i] := x[i] - a[i][j]*x[j]
        end
```

Как обычно,  $n$  опишем в программе как константу (положим, для определенности,  $n = 10$ ). Но для описания матрицы размера

$n \times (n+1)$  придется рассмотреть еще одну константу  $n!$  со значением, равным  $n+1$ . В итоге приходим к программе:

```

program Гайсц(input, output);
  const n=10; n1=11;
  type строка=array[1..n1] of real;
      матрица=array [1..n] of строка;
      столбец=array[1..n] of real;
  var a: матрица; x: столбец;
      maxabs, v: real; i, j, l, k: integer;
  begin
    for i:=1 to n do
      for j:=1 to n+1 do read(a[i][j]);
    for i:=1 to n do
      begin maxabs:=abs(a[i][i]); k:=i;
        for l:=i+1 to n do
          if abs(a[l][i]) > maxabs then
            begin maxabs:=abs(a[l][i]);
              k:=l;
            end;
          if k < > i then
            for j:=i to n+1 do
              begin v:=a[i][j]; a[i][j]:=a[k][j];
                a[k][j]:=v;
              end;
            v:=a[i][i];
            for j:=i to n+1 do a[i][j]:=a[i][j]/v;
            for l:=i+1 to n do
              begin v:=a[l][i];
                for j:=i+1 to n+1 do
                  a[l][j]:=a[l][j]-a[i][j]*v;
                end;
              end;
            end;
      x[n]:=a[n][n+1];
      for i:=n-1 downto 1 do
        begin x[i]:=a[i][n+1];
          for j:=i+1 to n do
            x[i]:=x[i]-a[i][j]*x[j];
          end;
        for i:=1 to n do writeln(x[i])
      end.

```

Предполагается, что данные для этой программы подготовлены так, что вначале идут коэффициенты первого уравнения и его правая часть, затем — коэффициенты второго уравнения и его правая часть и т. д.

349. а) Выписать расширенные матрицы для следующих систем:

$$\left\{ \begin{array}{l} 5x_1 + 35x_2 + 5x_3 + 5x_4 = 115, \\ 3x_1 + 4x_2 + 2x_3 + x_4 = 16, \\ 2x_1 + x_2 + 3x_3 + 5x_4 = 10, \\ 4x_1 - 3x_2 + 4x_3 + 6x_4 = 1, \end{array} \right. \quad \left\{ \begin{array}{l} 5x_1 + 35x_2 + 5x_3 + 5x_4 = 115, \\ 3x_1 + 4x_2 + 2x_3 + x_4 = 16, \\ 2x_1 - 3x_2 + 2x_3 + 4x_4 = -13, \\ 4x_1 - 3x_2 + 4x_3 + 6x_4 = 1, \end{array} \right. \quad \left\{ \begin{array}{l} 5x_1 + 35x_2 + 5x_3 + 5x_4 = 115, \\ 3x_1 + 4x_2 + 2x_3 + x_4 = 16, \\ 2x_1 + x_2 + 3x_3 + 5x_4 = 10, \\ 4x_1 - 3x_2 + 4x_3 + 6x_4 = 2. \end{array} \right.$$

350. Решить треугольные системы с единичными коэффициентами на диагонали:

$$\begin{aligned} \text{a) } x_1 - 2x_2 + 3x_3 - 4x_4 &= 3, \\ x_2 - 2x_3 + 3x_4 &= 2, \\ x_3 - 2x_4 &= 1, \\ x_4 &= 0. \end{aligned}$$

$$\begin{array}{rcl} 6) & x_1 + x_2 + x_3 + \dots + x_n = 1, \\ & x_2 + x_3 + \dots + x_n = 1, \\ & x_3 + \dots + x_n = 1, \\ & \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\ & x_{n-1} + x_n = 1, \\ & x_n = 1. \end{array}$$

$$\begin{aligned} \text{в) } x_1 - 3x_2 &= 0, \\ x_2 - 3x_3 &= 0, \\ . &. . . . . \\ x_{n-1} - 3x_n &= 0, \\ x_n &= \frac{1}{3^{m-1}}. \end{aligned}$$

$$\begin{aligned}x_1 + x_2 &= 0, \\x_1 + 2x_2 + x_3 &= 0, \\x_2 + 2x_3 + x_4 &= 0, \\x_3 + 2x_4 + x_5 &= 0, \\x_4 + 5x_5 &= 4.\end{aligned}$$

Как выглядит результат применения прямого хода алгоритма Гаусса к этой системе (т. е. каков вид равносильной треугольной системы с единичными коэффициентами на диагонали)? Каково решение этой системы?

Дать ответы на аналогичные вопросы относительно системы  $n$ -го порядка:

$$\begin{aligned}x_1 + x_2 &= 0, \\x_1 + 2x_2 + x_3 &= 0, \\x_2 + 2x_3 + x_4 &= 0, \\&\dots \dots \dots \\x_{n-2} + 2x_{n-1} + x_n &= 0, \\x_{n-1} + nx_n &= n-1.\end{aligned}$$

352. Изменить программу *Гаусс* в расчете на другой порядок исходных данных: вначале задается матрица коэффициентов, затем столбец правых частей.

353. а) Избавиться в тексте программы *Гаусс* от массива  $x_1, \dots, x_n$  добившись того, чтобы значение  $x_i$  занимало место  $a_{i\ n+1}$  ( $i=1, \dots, n$ ).

б) За счет вывода значений переменных в обратном порядке, т. е. в порядке  $x_n, x_{n-1}, \dots, x_1$ , упростить программу *Гаусс* (исключить из рассмотрения массив  $x$  и удалить последний оператор цикла).

354. От выполнения некоторых операций, предписываемых программой *Гаусс*, можно отказаться без ущерба для дела. А именно: при делении коэффициентов  $i$ -го уравнения на  $a_{ii}$  можно начинать не с коэффициента  $a_{ii}$ , а с  $a_{i\ i+1}$ , т. е. оператор цикла

`for j := i to n+1 do a[i][j] := a[i][j]/v`

можно заменить оператором

`for j := i+1 to n+1 do a[i][j] := a[i][j]/v`

Убедиться, что замена действительно не повлияет на результат выполнения программы.

355. Заменим в программе операторы

`maxabs := abs(a[i][i]); k := i;`

оператором `maxabs := 0`. Для того, чтобы программа оказалась правильной, достаточно дополнительно внести в нее одно небольшое изменение. Какое?

356. Применить программу *Гаусс* к следующим системам линейных уравнений:

$$\begin{aligned}\text{а) } 0.45x_1 + 0.03x_2 - 0.01x_3 + 0.02x_4 - 0.111x_5 &= 0.379, \\0.02x_1 + 0.375x_2 - 0.01x_3 - 0.01x_4 &= 0.375, \\0.07x_2 + 0.44x_3 &+ 0.113x_5 = 0.623,\end{aligned}$$



$$\begin{aligned}
 & -0.03x_1 + 0.015x_2 - 0.2x_3 + 0.41x_4 - 0.084x_5 \doteq 0.291, \\
 & \quad 0.02x_1 + 0.01x_2 \quad \quad \quad + 0.29x_5 = 0.32, \\
 \text{б) } & 0.38x_1 - 0.05x_2 + 0.01x_3 + 0.02x_4 + 0.07x_5 = 0.43, \\
 & \quad 0.052x_1 + 0.595x_2 \quad \quad \quad - 0.04x_4 + 0.04x_5 = 0.647, \\
 & \quad 0.03x_1 \quad \quad \quad + 0.478x_3 - 0.14x_4 + 0.08x_5 = 0.349, \\
 & -0.06x_1 + 0.126x_2 \quad \quad \quad - 0.47x_4 - 0.02x_5 = 0.516, \\
 & \quad 0.25x_1 \quad \quad \quad + 0.09x_3 + 0.01x_4 + 0.56x_5 = 0.91.
 \end{aligned}$$

Решение каждой из этих систем — это  $x_1 = x_2 = x_3 = x_4 = x_5 = 1$ , из-за округлений в процессе выполнения программы результат может несколько отличаться от истинного решения.

357. За счет ошибок округления значения неизвестных  $x_1, \dots, x_n$ , получаемые на компьютере при выполнении программы Гаусса, оказываются не вполне точными, и если для полученных значений вычислить величины

$$\delta_i = a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n - b_i \quad (i = 1, \dots, n),$$

то эти величины не будут точными нулями. Величины  $\delta_1, \dots, \delta_n$  называются *невязками*. Расширить программу Гаусса так, чтобы дополнительно к значениям  $x_1, \dots, x_n$  вычислялись:

- а) невязки  $\delta_1, \dots, \delta_n$ ;
- б) значение наибольшей по модулю невязки.

358. При применении на компьютерах алгоритма Гаусса (например, программы Гаусс) отмечается, что как правило наибольшая по модулю невязка (см. предыдущую задачу) соответствует тому уравнению, которое после перестановок уравнений оказывается в самом низу (из которого в конечном счете определяется  $x_n$ ), а наименьшая — тому, которое оказывается наверху (из которого в конечном счете определяется  $x_1$ ). Дать объяснение этому обстоятельству.

359. Исследовать затрачиваемое алгоритмом Гаусса число операций (в предположении, что порядок системы равен  $n$ ):

- а) деления;
- б) умножения;
- в) сложения и вычитания.

Основным средством исследования служит формула  $1 + 2 + \dots + k = \frac{k(k+1)}{2}$ , справедливая для любого натурального  $k$ .

360. Квадратная металлическая пластина является деталью некоторого устройства. Во время работы устройства во всех точках края пластины поддерживается определенная температура. На пластине начерчена сетка с квадратными ячейками. Распределение температуры в принадлежащих краю узлах сетки указано на рис. 65а. Из физических законов, описывающих перенос тепла, следует, что при достаточно малой длине стороны ячейки сетки можно считать температуру в каждом внутреннем узле сетки рав-

ной среднему арифметическому температур в ближайших четырех узлах (принадлежащих краю или внутренним):  $t_A = \frac{t_B + t_C + t_D + t_E}{4}$  (рис. 65б). Требуется узнать, каким будет распределение температур во всех внутренних узлах. Для этого предлагается каким-то способом пронумеровать числами 1, 2, ..., 9 внутренние узлы сетки, обозначить искомые значения температуры через  $t_1, t_2, \dots, t_9$

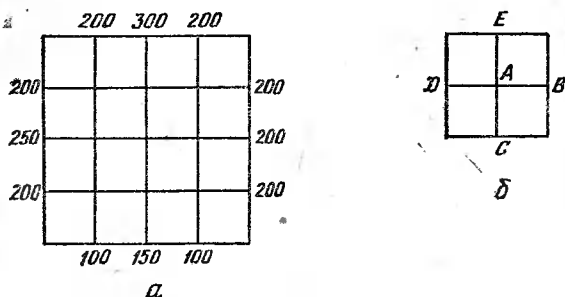


Рис. 65

и составить, исходя из сформулированного «принципа среднего арифметического», девять линейных уравнений с неизвестными  $t_1, t_2, \dots, t_9$ . Затем надо решить эту систему с помощью компьютера (эта система совместна и обладает единственным решением, хотя доказать последнее не просто). Интересно попытаться определить распределение температуры «на глаз» и то, что получится сравнить с результатами работы компьютера.

361. Алгоритм Жордана, также предназначенный для решения систем линейных уравнений, отличается от алгоритма Гаусса тем, что с помощью  $i$ -го уравнения неизвестная  $x_i$  исключается не только из уравнений с номерами  $i+1, i+2, \dots, n$ , но и из уравнений с номерами  $1, 2, \dots, i-1$ . В результате этого прямой ход алгоритма приводит к системе вида

$$\begin{aligned} x_1 &= c_1, \\ x_2 &= c_2, \\ &\dots \dots \dots \\ x_n &= c_n \end{aligned}$$

и обратный ход оказывается ненужным. Написать программу, реализующую алгоритм Жордана (в сравнении с программой Гаусса получается более простой вариант программы решения системы линейных уравнений, но число операций при этом несколько увеличивается).

362. Исследовать число операций, затрачиваемое алгоритмом Жордана (в предположении, что порядок системы равен  $n$ ). Про-

вести сравнение с характеристиками алгоритма Гаусса (см. задачу 359).

363. Программу Гаусс можно сделать более общей, добившись того, что одновременно будут решаться  $m$  систем линейных уравнений с одной (и той же матрицей коэффициентов размера  $n \times n$  и с  $m$  различными столбцами правых частей. Расширенная матрица имеет в этом случае размер  $n \times (n + m)$ . Какие изменения потребуется внести в программу Гаусс? Аналогичную программу написать с использованием алгоритма Жордана.

## ГЛАВА V

### ФАЙЛЫ

#### § 25. Файлы. Файловые типы

Удобным способом сохранения информации, полученной в ходе выполнения программы, служит запись этой информации на магнитный носитель. Разновидностями этих носителей являются твердые диски, гибкие диски (дискеты), магнитные ленты. Запись особенно желательна, если объем информации велик и если в дальнейшем предполагается использовать эту информацию в других программах. Для записи и чтения информации используются дисководы и магнитофоны.

В Паскале предусмотрены специальные объекты — файлы, операции над которыми сводятся к работе с лентами и дисками. *Файл* — это *последовательность компонент*, являющихся объектами одного и того же типа. Количество компонент в файле заранее не оговаривается, компоненты файла не имеют индексов. До некоторой компонент можно добраться, только перебрав по очереди все предыдущие компоненты.

Пример. Описание, имеющее вид

$v = \text{file of integer}^*)$

это описание типа, имя которого  $v$ . Объектами типа  $v$  будут файлы с целочисленными компонентами.

Объясняя принципы работы с файлами, будем для простоты и наглядности считать, что каждый файл записан на своей собственной ленте, и при этом [с самого начала ленты записано имя файла (идентификатор). После имени файла записаны компоненты, а вслед за самой последней компонентой записан *признак конца файла*. На рис. 66а светлые прямоугольники изображают компоненты, узкий темный прямоугольник — признак конца. В качестве имени файла здесь и в следующих рисунках использован иденти-

---

\*) File — подшивка, картотека.

фикатор  $f$ . На рис. 66б изображен файл, не имеющий ни одной компоненты (такая возможность допускается), — *пустой файл*.

В те моменты, когда на магнитофоне не происходит ни чтения, ни записи, головка чтения-записи совмещена либо с началом некоторой компоненты, либо с признаком конца. На рис. 67а и б представлены эти возможности, головка изображена треугольником.



Рис. 66

Теперь опишем операции над файлами. Пусть  $f$  — имя рассматриваемого файла, и пусть  $a$  — переменная того типа, объектами которого являются компоненты файла. Отметим предварительно,

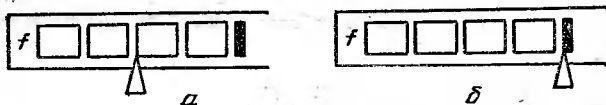


Рис. 67

что файл может быть *открыт для записи* и при этом *закрыт для чтения*, может быть *открыт для чтения* и при этом *закрыт для записи*, а может быть *закрыт* и для записи и для чтения. Подробности будут сообщены ниже.

**Запись в файл.** Эта операция возможна, только когда файл открыт для записи и головка совмещена с признаком конца. Пря

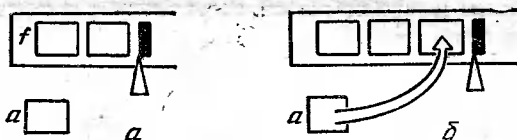


Рис. 68

соблюдении этих условий выполнение оператора  $write(f, a)$  приведет к тому, что в файл будет записана еще одна компонента, равная значению переменной  $a$  (\*). Признак конца будет записан после новой компоненты. Головка будет совмещена с признаком конца. Фазы выполнения этой операции изображены на рис. 68а и б.

\*) Перед выполнением оператора  $write(f, a)$  проверяется, описан ли идентификатор  $f$  как переменная, значением которой должен быть файл. Если да, то происходит запись в файл, иначе оператор  $write(f, a)$  выполняется как обычный оператор вывода. Это же касается оператора  $read(f, a)$ , о котором речь пойдет ниже.

*Подготовка к записи с начала файла.* В результате выполнения оператора *rewrite(f)\*)* все компоненты файла пропадают, признак конца помещается в самое начало, головка совмещается с признаком конца. Файл становится открытым для записи и закрытым для чтения. Фазы выполнения этой операции изображены на рис. 69а и б.



Рис. 69

*Пример.* Пусть *f*—файл, компонентами которого могут быть целые числа. Приведем фрагмент программы, обеспечивающий запись в *f* квадратов ста первых натуральных чисел:

```
rewrite(f); for i := 1 to 100 do
begin j := sqr(i); write(f, j) end
```

переменные *i* и *j* должны иметь тип *integer*.

*Чтение из файла.* Эта операция возможна, только когда файл открыт для чтения и головка совмещена с началом некоторой компоненты. При соблюдении этих условий после выполнения оператора *read(f, a)* переменная *a* будет иметь значение, равное той компоненте файла, с началом которой была совмещена головка. После выполнения оператора головка будет совмещена с началом следующей компоненты, если же следующей компоненты нет, то она будет совмещена с признаком конца. Фазы выполнения этой операции изображены на рис. 70а и б.

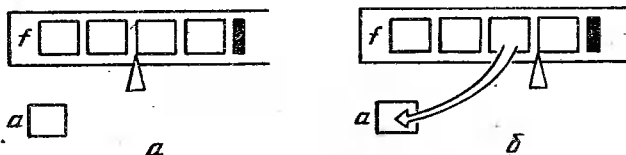


Рис. 70

*Подготовка к чтению с начала файла.* Если файл не пуст, то после выполнения оператора *reset(f)\*\*)* головка будет совмещена с началом первой компоненты файла. Две фазы показаны на рис. 71а и б. Если файл пуст, то головка указывает на признак конца, расположенный после имени файла (рис. 69б).

\*) Rewrite — перезапись.  
\*\*) Reset — переустановка.

**Распознавание конца файла.** Условие  $\text{eof}(f)$  \*) можно использовать в условном операторе после *if* и в операторе цикла после *while*. Это условие соблюдается тогда и только тогда, когда головка совмещена с признаком конца файла *f*.

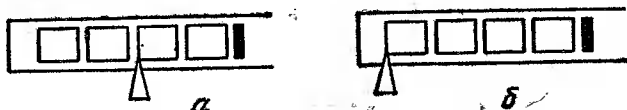


Рис. 71

**Пример.** Выполнение оператора

*if eof(f) then goto 1*

повлечет переход к оператору, помеченному меткой 1, тогда и только тогда, когда головка совмещена с признаком конца.

Часто бывает нужно после *if* или после *while* поместить не условие конца файла, а условие того, что файл не закончен. Это делается с помощью *not*:

*if not eof(f) then read(f, a)*

и т. д.

**Пример.** Фрагмент программы, обеспечивающий чтение из файла *f*, компонентами которого служат действительные числа, всех его компонент и вычисление суммы их квадратов:

```
reset(f); s := 0;
while not eof(f) do
  begin
    read(f, t); s := sqr(t) + s
  end
```

(переменные *s* и *t* должны иметь тип *real*).

Скажем более подробно об описаниях вида

*v = file of r*

Тип *v*, описанный таким образом, называется *файловым типом*. В качестве *r* может выступать любой из стандартных типов *real*, *integer*, *char*. Тип *r* называется базовым типом по отношению к *v*. Имя каждого файла, который будет использован в данной программе, должно быть указано в скобках после названия программы. Например,

*program P(input, output, f)*

или

*program P(f1, output, f2)*

\*) Eof — сокращение от end of file — конца файла.

и т. д. Программа должна содержать описания тех файловых типов, к которым принадлежат файлы, упомянутые в заголовке программы. Идентификаторы, служащие именами этих файлов, должны быть описаны в программе как переменные соответствующих типов.

**Пример.** Программа, в результате выполнения которой выводятся все малые латинские буквы из данного символьного файла *symb*:

```
program бук (symb, output);  
  type v = file of char;  
  var symb: v; s: char;  
  begin reset(symb);  
    while not eof(symb) do  
      begin read(symb, s);  
        if (s <= 'z') and (s >= 'a')  
          then writeln(s)  
        end  
      end  
  end.
```

Если к компонентам файла *f1* надо добавить еще какие-то компоненты, то для этого приходится переносить компоненты файла *f1* и новые компоненты в дополнительный файл, а потом, если нужно, из дополнительного файла переносить все компоненты в файл *f1*. Программа добавления к целочисленному файлу числа 100:

```
program p100(f1, f2);  
  type t = file of integer;  
  var f1, f2: t; a: integer;  
  begin reset(f1); rewrite(f2);  
    while not eof(f1) do  
      begin read(f1, a); write(f2, a) end;  
      a := 100; write(f2, a)  
    end.
```

Если все компоненты следует собрать в файле *f1*, то в конце программы надо поместить еще операторы

```
rewrite(f1); reset(f2);  
while not eof(f2) do  
  begin read(f2, a); write(f1, a) end
```

В персональных компьютерах файл, как правило, записывается не на ленту, а на диск. На дорожках одного диска (их общее число обычно равно сорока) может быть записано несколько файлов. Структура программы на Паскале, предназначенной для работы с файлами, не зависит от вида используемого магнитного носителя информации, т. е. не зависит от того, что именно исполь-



зуется: лента или диск. Мысленно всегда можно ориентироваться на ту ленточную модель, которая была описана в настоящем параграфе.

## ЗАДАЧИ

364. Компонентами файла  $f$  являются действительные числа.

Найти:

- а) сумму компонент;
- б) произведение компонент;
- в) сумму квадратов компонент;
- г) модуль суммы и квадрат произведения компонент;
- д) последнюю компоненту файла.

365. Компонентами файла  $f$  являются действительные числа.

Найти:

- а) наибольшее из значений компонент;
- б) сумму наибольшего и наименьшего значений компонент;
- в) разность первой и последней компонент.

366. Компонентами файла  $g$  являются натуральные числа.

Найти:

- а) количество четных чисел среди компонент;
- б) количество удвоенных нечетных среди компонент;
- в) количество квадратов нечетных чисел среди компонент.

367. Дано натуральное  $n$ . Записать в файл  $g$  целые числа  $b_1, \dots, b_n$ , определенные так, как указано в заданиях а) — д) задачи 139.

368. Последовательность  $x_1, x_2, \dots$  образована по закону

$$x_k = \frac{k - 0.1}{k^3 + |\operatorname{tg} 2k|} \quad (k = 1, 2, \dots). \text{ Дано действительное } \varepsilon > 0.$$

Записать в файл  $h$  члены последовательности  $x_1, x_2, \dots$ , остановившись после первого члена, для которого выполнено  $|x_k| < \varepsilon$ .

369. Дан символьный файл  $f$ . Получить копию файла в файле  $g$ .

370. Даны символьные файлы  $f1$  и  $f2$ . Переписать с сохранением порядка следования компоненты файла  $f1$  в файл  $f2$ , а компоненты файла  $f2$  в файл  $f1$ . Использовать вспомогательный файл  $h$ .

371. Дан символьный файл  $f$ . В файле  $f$  не менее двух компонент. Определить, являются ли два первых символа файла цифрами. Если да, то установить, является ли число, образованное этими цифрами, четным.

372. Дан файл  $f$ , компоненты которого являются натуральными числами. Получить в файле  $g$  все компоненты файла  $f$ :

- а) являющиеся четными числами;
- б) делящиеся на 3 и не делящиеся на 7;
- в) являющиеся полными квадратами.

373. Дан файл  $f$ , компоненты которого являются степенями числа 1.5:  $a_i = 1.5^i$  ( $i = 1, 2, \dots, n$ ). Количество компонент  $n$  заранее неизвестно. Получить в файле  $g$  последовательность  $a_1, \dots, a_n, a_{n+1}$ , где  $a_{n+1} = 1.5^{n+1}$ . Рассмотреть аналогичную задачу для случая, когда  $a_1, \dots, a_n, a_{n+1}$  надо получить в прежнем файле  $f$ .

374. Дан файл  $f$ , компоненты которого являются целыми числами. Записать в файл  $g$  все четные числа файла  $f$ , а в файл  $h$  — все нечетные. Порядок следования чисел сохраняется.

375. Дан символьный файл  $f$ . Записать в файл  $g$  компоненты файла  $f$  в обратном порядке.

376. Даны символьные файлы  $f$  и  $g$ . Записать в файл  $h$  сначала компоненты файла  $f$ , затем — компоненты файла  $g$  с сохранением порядка.

377. Дан файл  $f$ , компоненты которого являются целыми числами. Никакая из компонент файла не равна нулю. Числа в файле идут в следующем порядке: десять положительных, десять отрицательных, десять положительных, десять отрицательных и т. д. Переписать компоненты файла  $f$  в файл  $g$  так, чтобы в файле  $g$  числа шли в следующем порядке:

а) пять положительных, пять отрицательных, пять положительных, пять отрицательных и т. д. (предполагается, что число компонент файла  $f$  делится на 20);

б) двадцать положительных, двадцать отрицательных, двадцать положительных, двадцать отрицательных и т. д. (предполагается, что число компонент файла  $f$  делится на 40).

378. Даны символьные файлы  $f$  и  $g$ . Определить, совпадают ли компоненты файла  $f$  с компонентами файла  $g$ . Если нет, то получить номер первой компоненты, в которой файлы  $f$  и  $g$  отличаются между собой. В случае когда один из файлов имеет  $n$  компонент ( $n \geq 0$ ) и является началом другого (более длинного) файла, ответом должно быть число  $n+1$ .

379. Дан символьный файл  $f$ . Переписать компоненты файла в файл  $g$ , заменив при этом:

а) каждый восклицательный знак точкой;

б) каждую точку многоточием (т. е. тремя точками);

в) каждую из групп стоящих рядом точек одной точкой;

г) каждую из групп стоящих рядом точек многоточием (т. е. тремя точками).

380. Дан файл  $f$ , компоненты которого являются натуральными числами. Количество чисел в файле кратно 4. Первые два числа из каждой четверки задают положение на экране левого верхнего угла прямоугольника, следующие два числа — положение его правого нижнего угла. Построить прямоугольники, попадающие в верхнюю половину экрана.

## § 26. Классификация файлов

Мы видели, что имена файлов, записанных на внешних носителях, появляются в заголовке программы рядом с идентификаторами *input*, *output* или вместо этих идентификаторов:

```
program P(input, output, f)
program P(f, g)
```

и т. д. Считается, что *input* и *output* — это имена специальных файлов. Точнее говоря, считается, что исходные данные программы, вводимые с клавиатуры, образуют *входной файл input*, а результаты, выводимые на экран — *выходной файл output*. С той точки зрения на файлы, которая была принята в предыдущем параграфе, группы входных данных и результатов могут быть названы файлами лишь условно, и мы в дальнейшем не будем забывать об этой условности. Но, однако, между группой входных данных (хранимой на бумаге или в голове и вводимой постепенно с клавиатуры) и файлом, записанным на диске или ленте, в действительности есть и значительное сходство: в обоих случаях информация вводится в память компьютера последовательно: например, десятая компонента вводится только после того, как введены первые девять. Аналогично и результаты выводятся последовательно — также, как записываются компоненты в «настоящий» файл. В дальнейшем мы будем называть файлы, введенные в рассмотрение в предыдущем параграфе, *внешними файлами*. Таким образом, классификация файлов такова: входной файл, выходной файл, внешние файлы.

Главное отличие между входным и выходным файлом, с одной стороны, и внешними файлами — с другой, состоит в том, что входной и выходной файлы могут попеременно содержать компоненты стандартных типов *real*, *integer*, *char*, а для компонент любого конкретного внешнего файла обязательна принадлежность к какому-то одному типу (который, однако, может не быть стандартным — об этом речь будет идти ниже). К входному и выходному файлу не применяются *rewrite* и *reset*, к выходному файлу не применяется *eof*. Но, вместе с этим, *eof* можно применять к входному файлу: если в некоторый момент группа исходных данных, введенная с клавиатуры, исчерпана, то условие *eof(input)* окажется выполненным; в программе можно, например, использовать оператор цикла

```
while not eof(input) do
begin
  read(a); s := s + a
end
```

условный оператор

*if eof(input) then goto 1*

и т. д.

Теперь перейдем к случаям, когда в качестве компонент внешних файлов выступают объекты нестандартных типов, но сначала резюмируем и несколько дополним ранее сказанное о файловых типах. Общий вид описания файлового типа есть

*v = file of r*

где *r* — это *real*, *integer*, *char* или имя типа, ранее определенного в программе. Тип *r* называется базовым по отношению к типу *v*. Сам файловый тип не может быть базовым по отношению к другому типу. Операции над значениями типа *v* — это последовательный доступ к компонентам файлов и запись компонент типа *r* с начала файла. Операторы присваивания *a := b* с файловыми перемещениями *a* и *b* недопустимы. Имя каждого файла, который будет использован программой, должно быть указано в заголовке программы.

Рассмотрим пример. Компонентами внешнего файла *f* служат массивы  $a_1, \dots, a_{10}$  действительных чисел. Требуется вывести, т. е. получить в выходном файле, наибольшие значения элементов всех этих массивов:

```
program  $\Phi(f, output)$ ;  
  type t = array[1..10] of real;  
  v = file of t;  
  var a: t; f: v; r: real; i: integer;  
  begin reset(f);  
    while not eof(f) do  
      begin read(f, a);  
        r := a[1];  
        for i := 2 to 10 do  
          if a[i] > r then r := a[i];  
        writeln(r)  
      end  
    end.  
end.
```

Пример. Дано натуральное *n* (во входном файле). Требуется получить во внешнем файле *g* *n* матриц размера  $10 \times 10$ , при этом *k*-я матрица определяется следующим образом:

$$a_{ij} = \frac{k}{2i+j+k} \quad (i=1, \dots, 10; j=1, \dots, 10).$$

```
program M(input, g);  
  type строка = array [1..10] of real;  
  матрица = array [1..10] of строка;
```

```

    t = file of матрица;
var n, i, j, k: integer; a: матрица; g: t;
begin read (n); rewrite (g);
    for k:=1 to n do
        begin for i:=1 to 10 do
            for j:=1 to 10 do
                a [i] [j]:= k/(2*i + j + k);
            write(g, a)
        end
    end.

```

И последний пример. Файл *f* составлен из целочисленных квадратных матриц размера  $n \times n$  ( $n$  — некоторая константа). Получить эти матрицы в транспонированном виде в файле *g*. Операция транспонирования определена в условии задачи 343.

Программа (считаем, что  $n = 5$ ):

```

program трансп(f, g);
const n=5;
type строка=array[1..n] of integer;
    матрица=array[1..n] of строка;
    v=file of матрица;
var a: матрица; i, j, p: integer; f, g: v;
begin reset(f); rewrite(g);
    while not eof(f) do
        begin read(f, a);
            for i:=1 to n-1 do
                for j:=i+1 to n do
                    begin p:=a[i] [j];
                        a[i] [j]:=a[j] [i]; a[j] [i]:=p
                    end;
                write(g, a)
            end
        end.

```

В этой программе с помощью двойного цикла перебираются пары индексов  $i, j$  такие, что  $1 \leq i < j \leq n$ , т. е. пары индексов, соответствующие элементам правого верхнего угла матрицы. Для каждой такой пары  $i, j$  элементы  $a_{ij}$  и  $a_{ji}$  меняются местами.

### ЗАДАЧИ

381. а) Какие файлы используются программой  $\Phi$ ?

б) Можно ли в программе  $\Phi$  заменить оператор *reset(f)* на *rewrite(f)*?

в) Можно ли в программе  $\Phi$  заменить оператор  $r:=a[1]$  на оператор  $r:=0$ ?

382. Почему в программе  $\Phi$  не используется оператор *rewrite*?

383. К каким последствиям приведет замена фрагмента программы *трансп*

```
for i:=1 to n-1 do  
  for j:=i+1 to n do
```

на

```
for i:=1 to n-1 do  
  for j:=1 to n do?
```

Будет ли новая программа обеспечивать транспонирование? Ответить на аналогичные вопросы, касающиеся замены этого же фрагмента на

```
for i:=1 to n-1 do  
  for j:=i to n do
```

384. Дан файл  $f$ , компонентами которого являются массивы действительных чисел  $a_1, \dots, a_{15}$ . Подсчитать для каждого массива число положительных элементов в нем и поместить каждое из этих чисел:

- а) в выходной файл;
- б) в файл  $g$ .

385. Во входном файле даны действительные числа  $r_1, r_2, \dots$  про общее количество которых известно только, что оно кратно шестнадцати. Получить в файле  $f$  последовательность массивов действительных чисел по шестнадцать элементов в каждом:  $a_1, \dots, a_{16}$ . При этом первые шестнадцать чисел входного файла образуют первый массив, следующие шестнадцать — второй и т. д.

386. Эта задача аналогична предыдущей, но в файле требуется получить последовательность матриц:

- а) размера  $4 \times 4$ ;
- б) размера  $2 \times 8$ ;
- в) размера  $8 \times 2$ .

387. Дан файл  $h1$ , компонентами которого являются массивы целых чисел  $a_0, a_1, \dots, a_7$ . Записать в файл  $f2$  последовательность массивов, получающихся преобразованием исходных массивов:

- а)  $a_1, \dots, a_7, a_0$ ;
- б)  $a_0a_1, a_1a_2, \dots, a_7a_0$ ;

в)  $\frac{a_0}{1+a_2^2}, \frac{a_1}{1+a_3^2}, \dots, \frac{a_6}{1+a_0^2}, \frac{a_7}{1+a_1^2}.$

388. Дан файл  $h1$ , компонентами которого являются целочисленные массивы  $a_1, \dots, a_{10}$ . Преобразовать каждый из массивов следующим образом: заменить отрицательные элементы на  $-1$ , положительные — на  $1$ , а нулевые оставить без изменения. Записать полученные массивы в файл  $h2$ .

389. Дан файл  $f$ , компонентами которого являются символьные массивы  $s_1, \dots, s_{10}$ . Получить в файле  $f1$  символьные массивы по двадцать элементов в каждом. Эти массивы должны получаться следующими преобразованиями исходных массивов:

а)  $s_1, \dots, s_{10}, s_1, \dots, s_{10}$ ;

б)  $s_{10}, \dots, s_1, s_1, \dots, s_{10}$ .

390. Дан файл  $f$ , компонентами которого являются символьные массивы  $s_1, \dots, s_{15}$ . Требуется преобразовать каждый из массивов, переставив содержащиеся в нем пробелы в его конец. Преобразованные массивы должны быть записаны в тот же самый файл  $f$ . Разрешается использовать вспомогательный файл  $g$ .

391. Дан файл  $f$ , компонентами которого являются целочисленные массивы  $a_0, a_1, \dots, a_6$ . Требуется преобразовать каждый из массивов, заменив элементы с наибольшим значением нулем. Полученные массивы должны быть записаны в тот же самый файл  $f$ . Разрешается использовать вспомогательный файл  $g$ .

392. Дан файл  $g$ , компонентами которого являются квадратные матрицы размера  $5 \times 5$  с действительными элементами. По каждой матрице требуется построить массив  $b_1, \dots, b_5$ , где  $b_i$  — наибольшее значение элементов  $i$ -ой строки матрицы ( $i=1, \dots, 5$ ). Полученные массивы записать в файл  $h$ .

393. Во входном файле даны действительные числа  $x_1, x_2, \dots$  количество которых заранее неизвестно. Получить в файле  $f$  последовательность матриц размера  $n \times m$  ( $n$  и  $m$  — некоторые константы), где  $k$ -я матрица такова, что ее элемент с индексами  $i, j$  равен  $i + 2j + x_k$ .

394. Дан файл  $f$ , компонентами которого являются целочисленные массивы  $x_1, \dots, x_7$ . Каждый массив преобразовать в квадратную матрицу размера  $7 \times 7$ :

$$\text{а) } \begin{pmatrix} x_1 \dots x_7 \\ x_1^2 \dots x_7^2 \\ \vdots \\ x_1^7 \dots x_7^7 \end{pmatrix}; \quad \text{б) } \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_7 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^6 & x_2^6 & \dots & x_7^6 \end{pmatrix}$$

и записать в файл  $g$ .

395. Дан файл  $f$ , компонентами которого являются действительные матрицы размера  $4 \times 6$ . Преобразовать каждую из матриц, разделив ее элементы на сумму единицы и модуля наибольшего значения элементов матрицы. Полученные матрицы записать в файл  $g$ .

396. Вернуться к заданиям, сформулированным в задаче 379 из предыдущего параграфа. Выполнить эти задания, считая, что файл  $f$  — это входной файл *input*.

397. Содержимым входного файла является некоторая последовательность символов. Требуется переписать эти символы в выходной файл, выбрасывая при этом символы, расположенные между

скобками (, ). Сами скобки тоже выбрасываются. Предполагается, что внутри каждой пары скобок нет других скобок (, ).

**Указание.** Использовать в программе целочисленную переменную  $i$  такую, что в каждый момент  $i$  равно 0 или 1, и  $i=1$  означает, что ранее была прочитана левая скобка, для которой пока еще не нашлось парной правой.

## § 27. Записи. Комбинированные типы

Компьютеры очень часто используются в различных информационных службах. Эти службы имеются в больших библиотеках, в торговле, в производстве, в адресных бюро и т. д. Программы поиска информации справочного характера называют информационными системами. Реальные информационные системы работают с очень большими наборами данных. Такие наборы данных — сведения обо всех книгах библиотеки, сведения обо всех учениках школы, адреса всех жителей города и т. д., обычно заносятся в файл. В двух предыдущих параграфах была продемонстрирована техника работы с файлами. Все компоненты файла должны иметь один и тот же тип, и во многих рассмотренных примерах компонентами файлов были объекты стандартного типа *real*, *integer* или *char*, а самыми сложными компонентами были матрицы с элементами одного из стандартных типов. Но в информационных системах довольно часто приходится иметь дело с комбинациями разнотипных объектов. Обратимся к простейшим примерам: наименование класса состоит из числа (года обучения) и буквы; багаж может характеризоваться целым числом (числом вещей) и действительным числом (общим весом); краткие сведения об ученике школы могут состоять из двух последовательностей букв (имени и фамилии) и наименования класса.

В Паскале комбинациями объектов разного типа являются *записи*. Составляющие запись объекты называются ее *полями*. В записи каждое поле имеет свое собственное имя (идентификатор).

Количество полей, тип и имя каждого поля фиксируются в описании того типа, к которому относится запись. Описание, имеющее вид

*класс* = *record* *годобуч* : *integer*; *буква* : *char* *end* \*)

это описание типа, имя которого *класс*. Объектами типа *класс* будут записи, состоящие из двух полей, одно из которых имеет тип *integer*, а другое — *char*. Пусть переменная  $x$  описана в программе как переменная типа *класс*:

*var*  $x$  : *класс*

Тогда при выполнении программы значениями  $x$  будут записи с числом полей, равным 2. Переменная  $x$  — это переменная типа *класс*,

\*) *Record* — запись.



переменные *х.годобуч* и *х.буква*—это переменные типа *integer* и типа *char* соответственно. Значениями переменных *х.годобуч* и *х.буква* будут поля *годобуч* и *буква* той записи, которая служит значением переменной *х*. С переменной *х.годобуч* можно обращаться как с обычной переменной типа *integer*, а с переменной *х.буква*—как с обычной переменной типа *char*. Операции над объектами типа *класс*—это доступ к отдельным полям с помощью имен полей и изменение отдельных полей с помощью операций, связанных с типами *integer* и *char*.

**Пример.** Пусть *х* и *у*—переменные типа *класс*, тогда в результате выполнения условного оператора

```
if (х.годобуч=у.годобуч) and (х.буква < > у.буква)
  then write('параллельные')
  else write('не параллельные')
```

будет выведено *параллельные*, если рассматриваемые школьные классы являются, в принятой терминологии, параллельными, как например 10 «А» и 10 «Б»; в противном случае будет выведено *не параллельные*.

Резюмируем и дополним сказанное. Если тип *υ* описан в программе с помощью конструкции *record ... end*, то он называется *комбинированным* типом. Общий вид описания комбинированного типа *υ* есть

$o = \text{record } l_1: r_1; l_2: r_2; \dots; l_k: r_k \text{ end}$

где каждое из  $r_1, r_2, \dots, r_k$ —это *real, integer, char* или имя ранее определенного типа, а каждое из  $l_1, \dots, l_k$ —это либо имя поля (идентификатор), либо несколько имен полей, перечисленных через запятую. Типы  $r_1, r_2, \dots, r_k$  называются базовыми по отношению к типу *υ*. Если переменная *х*—это переменная типа *υ*, н, например,  $l_i$  имеет вид  $p, q, \dots, s$ , то  $х.p, х.q, \dots, х.s$ —это переменные типа  $r_i$  и т. д. Операции над объектами типа *υ*—это доступ к отдельным полям записей посредством указания имен полей, а также изменение отдельных полей с помощью операций, связанных с базовыми типами.

Для переменных *х, у* одного и того же комбинированного типа можно использовать оператор присваивания  $х := у$ .

Воспользуемся возможностью привлечения типа, который определен в программе, в качестве базового типа. Мы можем определить тип *ученик*. В записях, являющихся объектами этого типа, будут фиксироваться фамилия, имя ученика (массивы длины 15 с элементами типа *char*) и класс (тип *класс* описан выше):

```
φи = array[1..15] of char;
ученик = record φ, и: φи; кл: класс end;
```

Тогда, если  $x$  — переменная типа *ученик*, то

$x.u$ ,  $x.f$  — переменные типа *фи*,  
 $x.u[j]$ ,  $x.f[j]$  — переменные типа *char*,  
 $x.кл$  — переменная типа *класс*,  
 $x.кл.годобуч$  — переменная типа *integer*,  
 $x.кл.буква$  — переменная типа *char*.

Пусть в файле *Ш*, компонентами которого являются объекты (записи) типа *ученик*, собраны сведения об учениках некоторой школы. Требуется вывести первую букву имени и фамилию каждого из учащихся указанного класса, следуя примеру

И. Петров

П. Иванов

.....

Программа:

```
program одноклассники(input, output, Ш);
  type класс = record годобуч: integer;
                     буква: char
                   end;
  фи = array [1..15] of char;
  ученик = record ф, и: фи;
             кл: класс
           end;
  шк = file of ученик;
  var i: integer; к: класс; Ш: шк;
      у: ученик;
  begin read(к.годобуч, к.буква);
        reset(Ш);
        while not eof(Ш) do
          begin read(Ш, у);
                if (к.годобуч = у.кл.годобуч) and
                   (к.буква = у.кл.буква)
                then
                  begin writeln('_ ');
                        write (у.и[1], ', ');
                        for i := 1 to 15 do
                          write(у.ф[i])
                        end
                  end
                end
          end
  end.
```

В ряде задач могут рассматриваться не файлы, а массивы, элементами которых являются записи (объекты некоторого комбинированного типа). Это облегчает доступ к данным, но для этого как минимум необходимо, чтобы рассматриваемый набор данных имел не слишком большой объем и целиком умещался в памяти

компьютера. Пусть рассматриваются массивы с элементами типа *ученик*:

*школа* = *array*[1..*u*] of *ученик*,

где *u* — некоторая константа. Тогда, если *s* — переменная типа *школа*, то

*s*[*i*] — переменная типа *ученик*,

*s*[*i*].*u*, *s*[*i*].*ф* — переменные типа *фи*,

*s*[*i*].*u*[*j*], *s*[*i*].*ф*[*j*] — переменные типа *char*,

*s*[*i*].*кл* — переменная типа *класс*,

*s*[*i*].*кл.годобуч* — переменная типа *integer*,

*s*[*i*].*кл.буква* — переменная типа *char*.

Что касается настоящих информационных систем, то как уже говорилось, рассматриваемый объем данных там бывает достаточно большим. В памяти компьютера хранятся не сами данные, а некоторые сведения о том, как данные распределены в файле (или в файлах). Эти сведения позволяют ускорить поиск нужной информации в файлах.

## ЗАДАЧИ

398. Обязательно ли число служебных слов *begin*, встречающихся в некоторой программе, совпадает с числом слов *end*? Сформулировать общее правило.

399. Переделать программу *одноклассники* настоящего параграфа так, чтобы список учащихся указанного класса печатался в соответствии со следующим примером:

Петров Игорь  
Иванов Павел

.....

Между фамилией и именем помещается один пробел.

400. Требуется проверить правильность заполнения файла *Ш* (см. программу *одноклассники* в тексте параграфа). Всегда ли год обучения лежит в пределах от 1 до 11?

401. Данные об учениках расположены во входном файле, при этом данные о каждом отдельном ученике идут в следующем порядке: имя (15 символов), фамилия (15 символов), год обучения (целое число), буква (символ). Требуется перенести эти данные в файл *Ш*, описываемый так, как это сделано в программе *одноклассники* настоящего параграфа.

402. Эта задача аналогична предыдущей, но имя и фамилия задаются последовательностями букв, заканчивающимися каждая одним пробелом. По-прежнему считается, что и имя, и фамилия содержат не более 15 букв.

403. Дан файл *Ш*, содержащий сведения об учениках некоторой школы (см. программу *одноклассники* в тексте параграфа).

а) Собрать в файле *девятые* сведения об учениках девятых классов школы *Ш*;

б) Выяснить, на сколько человек в восьмых классах больше, чем в девятых.

404. Багаж пассажира характеризуется количеством вещей и общим весом вещей. Дан файл *багаж*, содержащий сведения о багаже нескольких пассажиров. Сведения о багаже каждого пассажира представляют собой запись с двумя полями: одно поле целого типа (количество вещей) и одно — действительное (вес в килограммах).

а) Найти багаж, средний вес одной вещи в котором отличается не более, чем на 0,3 кг от общего среднего веса одной вещи.

б) Найти число пассажиров, имеющих более двух вещей и число пассажиров, количество вещей которых превосходит среднее число вещей.

в) Выяснить, имеется ли пассажир, багаж которого состоит из одной вещи весом менее 30 кг.

405. Упорядочить сведения о багаже, записанные в файле *багаж* (см. предыдущую задачу) по невозрастанию веса багажа. Предполагается, что число пассажиров, зарегистрировавших багаж, известно заранее и равно  $n$  (некоторая константа), при этом  $n$  — не слишком велико.

*Указание.* Перенести сведения о багаже из файла *багаж* в массив  $B_1, \dots, B_n$ , затем упорядочить этот массив, используя то, что для переменных  $x, y$  одного и того же комбинированного типа можно использовать оператор присваивания  $x := y$  (См. программу *порядок* и задачу 319). После этого переписать элементы массива  $B_1, \dots, B_n$  в файл *багаж*.

406. Требуется удалить из данного файла *багаж* (см. задачу 404) сведения о багаже, общий вес вещей в котором меньше, чем 10 кг. Использовать вспомогательный файл *f*.

407. Переписать сведения о багаже из файла *багаж* (см. задачу 404) в файл *баг*. В файле *баг* сведения о багаже каждого пассажира представляются массивом из двух целых чисел — числа вещей и общего веса вещей, выраженного в граммах. Составить также программу обратного преобразования: переписи сведений о багаже из файла *баг* в файл *багаж*.

408. Дан файл *библ*, содержащий сведения о книгах. Сведения о каждой из книг — это фамилия автора, название и год издания \*).

---

\*) В этой и следующих задачах настоящего параграфа предлагается самостоятельно разработать форму представления сведений в виде объекта комбинированного или регулярного типа.

а) Найти названия книг данного автора, изданных с 1960 г.

б) Определить, имеется ли книга с названием «Информатика». Если да, то сообщить фамилию автора и год издания. Если таких книг несколько, то сообщить имеющиеся сведения обо всех этих книгах.

409. Дан файл *T*, который содержит номера телефонов сотрудников учреждения: указывается фамилия сотрудника, его инициалы и номер телефона. Найти номер телефона сотрудника по его фамилии и инициалам.

410. Дан файл, содержащий различные даты. Каждая дата — это число, месяц и год. Найти:

а) год с наименьшим номером;

б) все весенние даты;

в) самую позднюю дату.

411. Дан файл *товар*, содержащий сведения об экспортируемых товарах: указывается наименование товара, страна, импортирующая товар, и объем поставляемой партии в штуках. Составить список стран, в которые экспортируется данный товар, и общий объем его экспорта.

412. Дан файл *ассортимент*, содержащий сведения об игрушках: указывается название игрушки (кукла, кубики, конструктор и т. д.), ее стоимость в копейках и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Получить следующие сведения:

а) название игрушек, цена которых не превышает 4 р. и которые подходят детям 5 лет;

б) цену самого дорогого конструктора;

в) названия наиболее дорогих игрушек (цена которых отличается от цены самой дорогой игрушки не более, чем на 1 р.);

г) названия игрушек, которые подходят как детям 4 лет, так и детям 10 лет;

д) можно ли подобрать игрушку, любую, кроме мяча, подходящую ребенку 3 лет, и дополнительно мяч так, чтобы суммарная стоимость игрушек не превосходила 5 р.?

## ГЛАВА VI

### ИСПОЛЬЗОВАНИЕ ВСПОМОГАТЕЛЬНЫХ АЛГОРИТМОВ (ПРОЦЕДУРЫ И ФУНКЦИИ)

#### § 28. Процедуры без параметров.

##### Параметры — переменные

При составлении программы иногда получается так, что, по сути дела, одну и ту же последовательность операторов надо выписать несколько раз. Рассмотрим пример: пусть требуется составить программу вычисления площади выпуклого четырехугольника, заданного длинами четырех сторон и диагонали (рис. 72).

Диагональ делит выпуклый четырехугольник на два треугольника, к которым применима формула Герона

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где  $a, b, c$  — длины сторон треугольника,  $p = (a+b+c)/2$ . Простейшее решение — дважды выписать в программе операторы, задающие вычисления по этой формуле. Однако этого повторения можно избежать.

Паскаль позволяет ввести имя для составного оператора (а из любой группы операторов всегда можно сделать один составной оператор). После того как имя введено, в нужных местах программы помещается не сам оператор, а его имя.

Оператор, для которого введено имя, представляет собой процедуру. Использование в программе имени процедуры в качестве оператора называется *обращением к процедуре* или *оператором процедуры*.

Для того, чтобы ввести имя (идентификатор)  $P$  для составного оператора  $S$ , достаточно включить в программу описание процедуры, имеющее вид

procedure P; S; \*)

\*) Procedure — процедура.

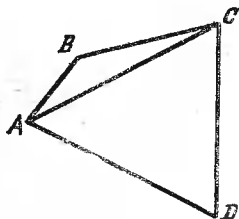


Рис. 72

В программе может содержаться несколько описаний различных процедур, все они, одно за другим, располагаются после совокупности описаний переменных.

Напишем первый вариант программы вычисления площади четырехугольника:

```

program F1(input, output);
  var AB, BC, CD, DA, AC, sl, s, a, b, c, p: real;
  procedure str1;
    begin p:=(a+b+c)/2;
          s:=sqrt(p*(p-a)*(p-b)*(p-c))
    end;
  begin read(AB, BC, CD, DA, AC);
        a:=AB; b:=BC; c:=AC; str1; sl:=s;
        a:=DA; b:=AC; c:=CD; str1; sl:=sl+s;
        write(sl)
  end.

```

В программе — два обращения к процедуре *str1*. Непосредственно перед каждым обращением идет группа операторов присваивания, задающих значения переменных *a*, *b* и *c*; каждое обращение влечет выполнение процедуры; после обращения к процедуре переменная *s* имеет значение, равное площади соответствующего треугольника.

Итак, связь процедуры *str1* с остальными операторами программы осуществляется через переменные *a*, *b*, *c*, *s*; в программе имеется еще переменная *p*, которая является вспомогательной внутри процедуры. Можно избежать совпадения вспомогательных переменных с переменными основной программы. Паскалем разрешается включать в описание процедуры совокупность описаний переменных:

```

procedure str2;
  var p: real;
  begin p:=(a+b+c)/2;
        s:=sqrt(p*(p-a)*(p-b)*(p-c))
  end;

```

Переменная, описанная в процедуре, называется *локальной* по отношению к процедуре. Слово «локальная» означает местная, имеющая местное назначение. Если переменная *p* — локальная по отношению к некоторой процедуре, то при выполнении этой процедуры работа с переменной *p* никак не будет влиять на значение переменной *p*, описанной в программе вне процедуры (если такая переменная имеется). Как только процедура будет выполнена значение локальной переменной *p* забудется. Аналогично, в процедуре могут описываться и использоваться локальные метки. Типы

и процедуры тоже могут быть локальными по отношению к процедуре (мы этой возможностью пользоваться не будем). Введение локальных меток, переменных и т. д. делает оформление процедуры похожим на оформление программы.

Если в программе *F1* использовать вместо процедуры *str1* процедуру *str2*, то можно из описания переменных программы удалить переменную *p*. Но можно и оставить описание прежним.

Продемонстрированный способ использования процедур не удобен из-за большого числа операторов присваивания, которые до обращения к процедуре определяют значения переменных *a*, *b* и *c*. Есть другой способ — описать процедуру с параметрами (аргументами), что позволит обращаться к процедуре, например, так:

*str3(AB, BC, AC, sl)*

Это обращение обеспечит вычисление площади треугольника со сторонами *AB*, *BC*, *AC* и присваивание значения площади переменной *sl*. Описание может выглядеть следующим образом:

```
procedure str3(var a, b, c, s: real);  
  var p: real;  
  begin p := (a + b + c)/2;  
        s := sqrt(p * (p - a) * (p - b) * (p - c))  
  end;
```

указанные в скобках после имени процедуры параметры *a*, *b*, *c*, *s* — это *формальные параметры*, при выполнении программы в момент обращения к процедуре они заменяются *фактическими параметрами*: первый формальный параметр заменяется первым фактическим параметром, второй формальный параметр — вторым фактическим и т. д.; фактическим параметром при таком описании процедур может быть переменная того типа, который указан при соответствующем ему формальном параметре. В данном примере фактические параметры должны быть переменными типа *real*. После подстановки фактических параметров на место формальных параметров процедура выполняется.

Формальные параметры не описываются.

Дадим еще один вариант программы вычисления площади четырехугольника:

```
program F2(input, output);  
  var AB, BC, CD, DA, AC, sl, s2: real;  
  procedure str3(var a, b, c, s: real);  
    var p: real;  
    begin p := (a + b + c)/2;  
          s := sqrt(p * (p - a) * (p - b) * (p - c))  
    end;  
  begin read(AB, BC, CD, DA, AC);
```



```

— str3(AB, BC, AC, s1);
  str3(CD, DA, AC, s2);
  write(s1 + s2)
end.

```

Первое обращение к процедуре приведет к выполнению оператора

```

begin p := (AB + BC + AC)/2;
      s1 := sqrt(p * (p - AB) * (p - BC) * (p - AC))
end

```

второе обращение — к выполнению оператора

```

begin p := (CD + DA + AC)/2;
      s2 := sqrt(p * (p - CD) * (p - DA) * (p - AC))
end.

```

Резюмируем и дополним сказанное. Процедура — это часть программы (составной оператор)  $S$ , получающая некоторое имя (идентификатор)  $P$  с помощью описания процедуры. Описание процедуры начинается с заголовка

*procedure P;*

или

*procedure P (l<sub>1</sub>; l<sub>2</sub>; ...; l<sub>m</sub>);*

здесь  $l_1, l_2, \dots, l_m$  — группы формальных параметров. В группе формальных параметров *var a, b, ..., x: t* идентификаторы  $a, b, \dots, x$  и есть, собственно, формальные параметры, а  $t$  должно быть именем типа. Первый заголовок — это заголовок *процедуры без параметров*, второй — заголовок *процедуры с параметрами*, точнее — заголовок процедуры с параметрами-переменными. Пример заголовка процедуры с параметрами-переменными

*procedure Q(var a: t; var b, c: char; var d: t; var e: real);*

Вслед за заголовком могут идти описания локальных меток и локальных переменных. Но эти описания могут и отсутствовать. Далее идет процедура, т. е. оператор, который получает имя. Описание процедуры заканчивается точкой с запятой.

Все описания процедур идут одно за другим после совокупности описаний переменных. В процедуре могут содержаться обращения к ранее описанным процедурам.

Обращение в программе к процедуре без параметров влечет непосредственное выполнение процедуры. Обращение к процедуре с параметрами-переменными влечет подстановку в процедуре переменных, указанных в качестве фактических параметров, на место формальных параметров, а затем — выполнение процедуры. Тип

фактического параметра должен совпадать с типом, сопоставленным в заголовке формальному оператору. Обращение к процедуре  $Q$  (см. выше пример заголовка) может иметь вид

$Q(v[j+1], b1, b2, u, r[j].st)$

при этом переменные  $v[j+1]$ ,  $u$  обязаны иметь тип  $t$ , переменные  $b1, b2$  — тип  $char$ , переменная  $r[j].st$  — тип  $real$ .

Если в обращении к процедуре с параметрами-переменными фигурируют переменные, включающие индексы, как, например,  $v[j+1]$ ,  $r[j].st$ , то перед подстановкой этих фактических параметров на место формальных параметров все индексы заменяются их числовыми значениями. При  $j=3$  будут подставлены  $v[4]$  и  $r[3].st$ .

Локальные переменные и метки доступны только внутри процедуры. Значения локальных переменных после окончания выполнения процедуры забываются.

Приведем пример программы с процедурой, которая предназначена для проверки гипотезы Гольдбаха для данного четного  $n$ . Эта гипотеза (по сегодняшний день не опровергнутая и полностью не доказанная) заключается в том, что каждое четное число, большее двух, представляется в виде суммы двух простых чисел. Проверка может быть проведена так: для каждого  $i=2, 3, \dots, n/2$  выясняем, является ли оно простым, и если да, то дополнительно выясняем, является ли простым  $n-i$ . Напишем отдельно группу операторов, выполнение которой приводит к присваиванию переменной  $p$  значения 1, коль скоро значение  $k \geq 2$  является простым, и к присваиванию переменной  $p$  значения 0 в противном случае. Воспользуемся тем, что каждому большему или равному  $\sqrt{k}$  делителю  $i$  числа  $k$  соответствует меньший или равный  $\sqrt{k}$  делитель числа  $k$  (делитель равный  $k/i$ ):

```
p:=1;
for i:=2 to round(sqrt(k)) do
if k mod i=0 then begin p:=0; goto 1 end; 1;
```

(последний оператор — пустой, помеченный меткой 1).

Эту последовательность операторов оформим в программе проверки гипотезы Гольдбаха как процедуру  $pr$ :

```
program Гольдбах(input, output);
label 0, 10;
var n, s, i, m: integer;
procedure pr(var k, p: integer);
label 1;
var i: integer;
begin p:=1;
for i:=2 to round(sqrt(k)) do
```

```

        if  $k \bmod i = 0$  then
            begin  $p := 0$ ; goto 1 end;
1: end;
begin read(n);
    for  $i := 2$  to  $n \div 2$  do
        begin pr(i, s);
            if  $s = 0$  then goto 0;
             $m := n - i$ ; pr(m, s);
            if  $s = 0$  then goto 0;
            write(i, m); goto 10;
        0: end;
        write(n);
10: end.

```

Выполнение этой программы приводит к выводу самого числа  $n$ , если пары простых слагаемых не найдено. Если же простые слагаемые удалось найти, то будут выведены они. Отметим некоторые важные моменты, связанные с использованием процедуры *pr*:

1) локальная, по отношению к процедуре *pr* переменная  $i$  не смешивается с переменной  $i$ , описанной вне процедуры;

2) первое обращение, имеющее вид *pr* ( $i$ ,  $s$ ), несколько рискованно — нужно обязательно предварительно убедиться что такое обращение не повлечет изменения значения переменной  $i$ , являющейся параметром цикла в основной программе. Но в данном случае опасности нет, так как формальный параметр  $k$  не встречается в процедуре слева от  $:=$ ;

3) перед вторым обращением пришлось поместить оператор присваивания  $m := n - i$ , так как формальному параметру — переменной может соответствовать в качестве фактического параметра только переменная, и мы не можем написать *pr* ( $n - i$ ,  $s$ ).

В следующем параграфе будет указан способ преодоления неудобств, отмеченных в двух последних пунктах.

## ЗАДАЧИ

413. Какое из следующих описаний процедур без параметров составлено правильно, а какие — нет, и почему:

- а) *procedure dis*;  
 $d := \text{sqr}(b) - 4 * a * c$ ;
- б) *procedure dis*;  
 $\text{var } d: \text{real}$ ;  
 $d := \text{sqr}(b) - 4 * a * c$ ;
- в) *procedure dis*;  
 $\text{begin } d := \text{sqr}(b) - 4 * a * c \text{ end}$ ;

414. Рассмотрим два описания процедур без параметров

<i>procedure P;</i>	<i>procedure Q;</i>
<i>begin</i>	<i>begin</i>
$x := x - y;$	$y := x + y;$
$y := x + y$	$x := x - y$
<i>end;</i>	<i>end;</i>

Одинаковы ли последствия обращения к процедурам *P* и *Q*?

415. Рассмотрим два описания процедур без параметров:

<i>procedure F;</i>	<i>procedure G;</i>
<i>var x: real;</i>	<i>var x, y, z: real;</i>
<i>begin</i>	<i>begin</i>
$x := y + z$	$x := y + z$
<i>end;</i>	<i>end;</i>

Один из операторов процедуры *F* и *G* не может быть выполнен ни при каких обстоятельствах, а другой — может в определенных случаях. Какой именно из операторов *F* и *G* и в каких случаях может быть выполнен? Каковы последствия этого выполнения?

416. Для каждого из приведенных ниже описаний процедур сформулировать назначение соответствующей процедуры:

а) *procedure S(var x, y, z: real);*  
    *begin z := x + y end;*

б) *procedure P(var x, y, z, t: real);*  
    *begin z := x + y; t := x \* y end;*

в) *procedure C(var x, y, r: real; var p: integer);*  
    *begin*  
        *if*  $\text{sqr}(x) + \text{sqr}(y) \leq \text{sqr}(r)$  *then*  $p := 1$   
            *else*  $p := 0$   
    *end;*

417. Что будет выведено при выполнении следующей программы:

```
program П(output);  
    var x, y, z: real;  
    procedure P(var a, b: real);  
        var z: real;  
        begin z := a; a := b; b := z end;  
    procedure Q(var a, b: real);  
        begin z := a; a := b; b := z end;  
    begin  $x := 1.1; y := 2.2; z := 3.3;$   
         $P(x, y); \text{writeln}(x, y, z);$   
         $x := 1.1; y := 2.2; z := 3.3;$   
         $Q(x, y); \text{writeln}(x, y, z)$   
    end.
```

418. Если в программе и в процедуре описана одна и та же переменная, то какое из этих описаний имеет силу в процедуре?

419. Процедура описана следующим образом:

*procedure F(var x, y: integer);*  
*begin x:=y end;*

Допустимо ли обращение к процедуре, имеющее вид  $F(a, b-1)$ ?

420. Имеет место следующий интересный геометрический факт: если координаты вершин треугольника в прямоугольной системе координат равны, соответственно  $(x_1, y_1)$ ,  $(x_2, y_2)$ ,  $(x_3, y_3)$ , то площадь этого треугольника равна

$$0.5 | x_1 y_2 + x_2 y_3 + x_3 y_1 - x_1 y_3 - x_2 y_1 - x_3 y_2 |.$$

Используя этот факт, написать программу вычисления площади выпуклого четырехугольника  $ABCD$  (рис. 72), заданного координатами четырех вершин  $(x_A, y_A)$ ,  $(x_B, y_B)$ ,  $(x_C, y_C)$ ,  $(x_D, y_D)$ .

421. Написать программу проверки существования «близнецов», т. е. простых чисел, разность между которыми равна двум, среди чисел  $n, n+1, \dots, 2n$ , где  $n$ —данное число (по сегодняшний день неизвестно, бесконечно ли множество пар близнецов)

422. а) Написать программу, в ходе выполнения которой компоненты файла  $f1$  переписываются в файл  $f2$ , а компоненты файла  $f2$ —в файл  $f1$ . Использовать файл  $h$  как вспомогательный. Компоненты всех файлов имеют тип *real*. В Паскале не разрешены операторы присваивания вида  $f:=g$ , где  $f$  и  $g$ —имена файлов, поэтому следует описать процедуру присваивания *priscv(f, g)*;

б) С помощью процедуры *priscv(f, g)* (см. задание а)) написать программу, в ходе выполнения которой файлы  $f1, f2, f3, f4, f5$  обмениваются компонентами в соответствии со следующей схемой:

$f1$	$f2$	$f3$	$f4$	$f5$
$f3$	$f4$	$f5$	$f2$	$f1$

т. е. компоненты файла  $f1$  переписываются в файл  $f3$ , компоненты файла  $f2$  переписываются в файл  $f4$  и т. д. Разрешается использовать только один дополнительный файл.

423. Вернемся к программе *одноклассники* из предыдущего параграфа: потребуем теперь, чтобы выполнение программы приводило к печати фамилий и имен столбиком, каждая строка которого выглядит так: фамилия, 1 пробел, имя, 1 пробел, запятая (после последней строки—точка). Описать процедуру *печать* с одним параметром, значением которого является массив с элементами типа *char*, имеющими индексы от 1 до 15. При обращении к процедуре должны печататься элементы массива до появления пробела.

424. Переделать программу *Гольдбах* и процедуру *pt* так, чтобы выполнение процедуры приводило, если число составное, к переходу к метке 0 (не локальной по отношению к процедуре). Если же число простое, то должен происходить естественный переход

к следующему в программе оператору. Оператор цикла основной программы можно будет записать так:

```
for i:= 2 to n div 2 do
  begin prl(i); m:= n-i; prl(m);
  write(i, m); goto 10;
0: end
```

425. Дано: натуральные  $n, m$  ( $m > 1$ ), целые  $a_1, \dots, a_n, b_1, \dots, b_m, c_1, \dots, c_{30}$ . Получить

$$l = \begin{cases} \min(b_1, \dots, b_m) + \min(c_1, \dots, c_{30}) & \text{при } |\min(a_1, \dots, a_n)| > 10, \\ 1 + (\min(c_1, \dots, c_{30}))^2 & \text{в противном случае.} \end{cases}$$

426. Дано: натуральные  $k, l, m$ , действительные  $x_1, \dots, x_k, y_1, \dots, y_l, z_1, \dots, z_m$ . Получить

$$t = \begin{cases} (\max(y_1, \dots, y_l) + \max(z_1, \dots, z_m))/2 & \text{при } \max(x_1, \dots, x_k) \geq 0, \\ 1 + (\max(x_1, \dots, x_k))^3 & \text{в противном случае.} \end{cases}$$

427. Дано: натуральное  $n$ , действительные  $a_1, \dots, a_{3n}$ . Получить  $x + y^2 + z^3$ , где

$$x = a_1 \cdot a_2 \cdot \dots \cdot a_n,$$

$$y = a_{n+1} \cdot a_{n+2} \cdot \dots \cdot a_{2n},$$

$$z = a_{2n+1} \cdot a_{2n+2} \cdot \dots \cdot a_{3n}.$$

## § 29. Параметры-значения

Заголовок процедуры может быть устроен так, что некоторые группы формальных параметров не содержат слова *var*. Например,

```
procedure str4(a, b, c: real; var s: real);
procedure prim(k: integer; var p: integer);
```

и т. д. Формальные параметры, которые входят в группы, не содержащие слова *var*, называются *формальными параметрами-значениями*. В первом примере  $a, b, c$ —это формальные параметры-значения, а  $s$ —это формальный параметр-переменная. Во втором примере  $k$ —это формальный параметр-значение, а  $p$ —это формальный параметр-переменная.

Фактическим параметром, соответствующим такому формальному параметру-значению, при котором указан тип *real*, *integer* или *char*, может быть не только переменная. Так, если при формальном параметре-значении указан тип *real* (в первом примере такими формальными параметрами будут  $a, b, c$ ), то соответствующим фактическим параметром может быть любое выражение, т. е. переменная типа *real* или *integer*, число или же более сложная конструкция, возможно, содержащая знаки арифметических операций и функций. Если указан тип *integer* (во втором примере таким формальным параметром будет  $k$ ), то соответствующим фак-

тическим параметром может быть любое выражение со значением типа *integer*. К процедуре *str4* можно обратиться, например, так:

```
str4(3.14, x, sqrt(10—sqr(x)), y)
```

к процедуре *prim*, в свою очередь, — так:

```
prim(n—i, m)
```

Фактическим параметром, соответствующим такому формальному параметру-значению, при котором указан тип *char*, может быть не только переменная типа *char*, но и конкретный символ, взятый в кавычки, например 'a', '\*' и т. д.

Пусть в некоторый момент выполнения программы происходит обращение к процедуре, имеющей параметры-значения. Тогда в начало составного оператора следующего в описании процедуры за заголовком (и, возможно, за описаниями локальных меток и локальных переменных) для каждого формального параметра-значения вставляется оператор присваивания; слева от  $:=$  помещается формальный параметр-значение, справа — соответствующий ему фактический параметр. В получившийся составной оператор на место формальных параметров-переменных подставляются переменные, являющиеся фактическими параметрами. После этого оператор выполняется.

Если описанием процедуры *str4* служит

```
procedure str4(a, b, c: real; var s: real);  
  var p: real;  
  begin p:= (a+b+c)/2;  
        s:= sqrt(p*(p-a)*(p-b)*(p-c))  
  end;
```

то обращение *str4*(3.14, x, sqrt(10—sqr(x)), y) повлечет выполнение составного оператора

```
begin a:= 3.14; b:= x; c:= sqrt(10—sqr(x));  
  p:= (a+b+c)/2;  
  y:= sqrt(p*(p-a)*(p-b)*(p-c))  
end
```

Вернемся теперь к программе *Гольдбах*, рассмотренной в предыдущем параграфе. В новом варианте этой программы опишем процедуру *pr1*, используя соображение, которое сформулировано в задаче 424. Это будет процедура с параметром-значением:

```
program Гольдбах1(input, output);  
  label 0, 10;  
  var n, i: integer;  
  procedure pr1(k: integer);  
    var i: integer;
```

```

begin
  for i:=2 to round(sqrt(k)) do
    if k mod i=0 then goto 0
  end;
begin read(n);
  for i:=2 to n div 2 do
    begin pr1(i); pr1(n-i);
      write(i, n-i); goto 10;
    0: end;
  write(n);
10: end.

```

В программе—два обращения к процедуре *pr1*. Первое обращение приводит к выполнению оператора

```

begin k:=i;
  for i:=2 to round(sqrt(k)) do
    if k mod i=0 then goto 0
  end

```

второе—к выполнению оператора

```

begin k:=n-i;
  for i:=2 to round(sqrt(k)) do
    if k mod i=0 then goto 0
  end

```

Из общих правил о параметрах-переменных и о параметрах-значениях, а также из приведенных примеров видно, что если некоторый формальный параметр изображает результат выполнения процедуры (как, например, параметр *s* в рассмотренных вариантах процедуры вычисления площади треугольника), то этот формальный параметр должен быть формальным параметром-переменной. Формальные параметры, при которых указан файловый тип, запрещается объявлять формальными параметрами-значениями, они обязаны быть формальными параметрами-переменными.

Все формальные параметры, кроме тех, которые изображают результаты выполнения процедуры, и тех, при которых указан файловый тип, на первых этапах занятий программированием рекомендуется объявлять в программах формальными параметрами-значениями.

В Паскале принято соглашение, что если формальный параметр-значение обозначен тем же идентификатором, что и некоторая переменная программы, то при выполнении программы для этого формального параметра выбирается другой идентификатор, и никаких недоразумений не возникает. Например, формальный параметр процедуры *pr1*, которая содержится в программе *Гольдбах*,



мог бы быть обозначен через  $n$  — это не изменило бы результата выполнения программы. Однако формальный параметр этой процедуры нельзя обозначать через  $i$ , так как в процедуре есть локальная переменная  $i$ .

Подчеркнем, что параметры-значения не обязательно имеют стандартный тип. Пусть, например, в программе описана константа  $n$  и регулярный тип  $T$ :

$T = \text{array}[1..n] \text{ of } \text{real}$

Для вычисления величины  $s = a_1^2 + a_2^2 + \dots + a_n^2$  и  $t = \max(|a_1|, |a_2|, \dots, |a_n|)$  может быть использована процедура

```

procedure  $r(a: T; \text{var } s, t: \text{real});$ 
  var  $i: \text{integer}; u: \text{real};$ 
  begin
     $s := \text{sqr}(a[1]); t := \text{abs}(a[1]);$ 
    for  $i := 2 \text{ to } n$  do
      begin  $s := s + \text{sqr}(a[i]);$ 
         $u := \text{abs}(a[i]);$ 
        if  $u > t$  then  $t := u$ 
      end
    end;

```

при обращении к этой процедуре с помощью оператора процедуры  $r(x, p, q)$  будет выполняться составной оператор

```

begin  $a := x;$ 
   $p := \text{sqr}(a[1]); q := \text{abs}(a[1]);$ 
  for  $i := 2 \text{ to } n$  do
    begin
       $p := p + \text{sqr}(a[i]);$ 
       $u := \text{abs}(a[i]);$ 
      if  $u > q$  then  $q := u$ 
    end
  end

```

В операторе присваивания  $a := x$  левая и правая части являются переменными регулярного типа  $T$ .

## ЗАДАЧИ

428. Верно ли, что при обращении к процедуре на место формальных параметров-значений подставляются значения соответствующих фактических параметров?

429. Можно ли при обращении к процедуре, имеющей формальный параметр-значение, задать в качестве соответствующего фактического параметра некоторую переменную?

430. В программе описана процедура  $P$  с формальными параметрами  $x, y$  и процедура  $Q$  с формальными параметрами  $s, t$ . Среди операторов программы встречаются операторы процедуры  $P(1, a)$  и  $Q(b, d+f)$ . Какие из формальных параметров процедур  $P$  и  $Q$  заведомо являются параметрами-значениями?

431. Пусть процедура имеет формальный параметр-значение. Сколько раз и в какие моменты будет вычисляться значение соответствующего фактического параметра?

432. Пусть программа содержит описание процедуры

```
procedure max(x, y: real; var z: real);
begin
    if x < y then z:=y else z:=x
end;
```

пусть в ходе выполнения программы переменные  $a, b$  и  $c$  получили некоторые действительные значения. Верно ли, что после выполнения операторов  $\text{max}(a, b, d)$ ;  $\text{max}(d, c, e)$  будет выполнено  $e \geq a$ ?

433. Выписать составной оператор, который будет выполняться при следующем обращении к процедуре  $\text{max}$ , описанной в предыдущей задаче:  $\text{max}(a-1, b+1, d)$ .

434. Рассмотрим описание процедуры  $P$ :

```
procedure P(var a, b: integer);
begin
    if a < 0 then a:=a+1;
    b:=sqr(a)
end;
```

и описание процедуры  $Q$ :

```
procedure Q(a: integer; var b: integer);
begin
    if a < 0 then a:=a+1;
    b:=sqr(a)
end;
```

Будут ли совпадать значения переменной  $n$  после выполнения операторов  $m:=-1$ ;  $P(m, n)$ ;  $n:=n+m$  и, соответственно, операторов  $m:=-1$ ;  $Q(m, n)$ ;  $n:=n+m$ ? Тот же самый вопрос для операторов  $m:=1$ ;  $P(m, n)$ ;  $n:=n+m$  и  $m:=1$ ;  $Q(m, n)$ ;  $n:=n+m$ .

435. Вернемся к процедуре  $r$ , рассмотренной в конце настоящего параграфа. Назначение локальной переменной  $u$  — уменьшение числа обращений к функции  $\text{abs}$ . Этой же цели можно добиться, минуя введение дополнительной переменной:

```
a[i]:=abs(a[i]);
if a[i] > t then t:=a[i].
```

Благодаря тому, что  $a$  является формальным параметром-значением, выполнение оператора  $a[i]:=abs(a[i])$  не повлечет изменения соответствующего фактического параметра. Убедиться в этом, выписав составной оператор, который будет выполнен при обращении к процедуре с помощью оператора процедуры  $r(x, p, q)$ . Выписать аналогичный оператор для случая когда  $a$  — формальный параметр-переменная. Будет ли изменяться  $x$ ?

Дополнительно ответить на вопрос, можно ли в описании процедуры без ущерба для дела заменить

```
s:=sqr(a[1]); t:=abs(a[1])
for i:=2 to n do
```

.....

на

```
s:=0; t:=0;
for i:=1 to n do
```

.....

436. Написать программу вычисления площади многоугольника, изображенного на рис. 73. Длины известных отрезков написаны рядом с этими отрезками. Предложить два варианта программы:

а) описав процедуру *str4*, рассмотренную в этом параграфе,

б) описав процедуру вычисления площади треугольника *str5*, имеющую три параметра — длины сторон треугольника. В результате обращения к этой процедуре

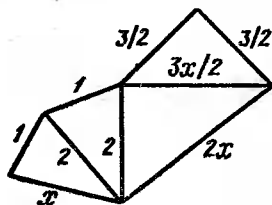


Рис. 73

вычисленная площадь должна прибавляться к значению переменной  $s$ , которая не является параметром и не является локальной переменной по отношению к процедуре.

437. Написать программу исследования файла  $f$  с компонентами типа *игрушка* (см. задачу 412). В результате выполнения этой програм-

мы должны быть выведены цены всех игрушек, сведения о которых именуются в данном файле. Цены должны указываться в рублях и копейках: 3 р. 15 к., 0 р. 07 к., 10 р. 00 к. (число копеек записывается всегда двумя цифрами). В программе полезно описать процедуру печати цены; цена первоначально задана в виде целого числа копеек; например, 315, 7, 1000.

438. Даны целые числа  $k, l, m$  и последовательность символов  $s_1, \dots, s_{30}$ . Надо написать программу, которая позволяет вывести данные символы в следующем виде:

```
□...□□□ s1□□...□ sl□□...□*
```

$k$  пробелов  $l$  пробелов  $m$  пробелов

```

      s2           s17           *
    . . . . .
      s15         s30           *

```

Полезно описать процедуру *print(s, n)*, обращение к которой дает вывод символа *s* после *n* пробелов.

439. Даны натуральные *k, m*. Требуется вывести на экран рамку из звездочек

```

***
*   *
*   *
*   *
*   *
***

```

высота которой — *k* строк, ширина — *m* знаковых позиций. Полезно описать процедуру *печать(s, n)*, обращение к которой дает вывод *n* символов *s*.

440. Дано: натуральное *n*, целые неотрицательные  $a_1, \dots, a_n$ . Рассмотреть отрезки последовательности  $a_1, \dots, a_n$  (подпоследовательности идущих подряд членов), состоящей из:

- а) полных квадратов; б) степеней пятерки;
- в) простых чисел; г) совершенных чисел.

В каждом случае получить наибольшую из длин рассматриваемых отрезков. (Описать процедуры, позволяющие распознавать полные квадраты, степени пятерки, простые числа, совершенные числа.)

441. Даны действительные  $x_1, y_1, x_2, y_2, \dots, x_{10}, y_{10}$ . Найти периметр десятиугольника, вершины которого имеют, соответственно, координаты  $(x_1, y_1), (x_2, y_2), \dots, (x_{10}, y_{10})$ . Описать процедуру вычисления расстояния между двумя точками, заданными своими координатами.

442. Даны целые  $u_1, u_2, v_1, v_2, w_1, w_2$  ( $u_2, v_2, w_2 \neq 0$ ). Вычислить  $2u + \frac{3uw}{2 + v^2} - 7$ , где  $u, v, w$  — рациональные числа  $u_1/u_2, v_1/v_2, w_1/w_2$ . Ответ получить в виде двух целых взаимно простых чисел  $p_1, p_2$  — числителя и знаменателя дроби, являющейся значением выписанного выражения. Описать процедуру полного сокращения рационального числа, заданного числителем и знаменателем  $x, y$  ( $y > 0$ ). Описать также процедуры сложения и умножения рациональных чисел.

## § 30. Построение диаграмм

Часто бывает удобно результаты измерения некоторой величины или результаты вычислений представлять не в виде последовательности чисел, а в графической форме, например, в виде диаграмм. Диаграммы используются, например, при отображении экономической информации. Рассмотренные нами ранее графические

средства дают возможность достаточно просто выполнять построение диаграмм на экране компьютера.

Пусть  $x_1, x_2, \dots, x_{15}$  — количество изделий, выпущенных некоторым предприятием за 15 дней. Напишем программу построения диаграммы, отображающей на экране информацию о количестве выпущенных изделий. Каждому значению  $x_i$  на экране будет соответствовать прямоугольник со сторонами, параллельными сторонам экрана. Основания всех прямоугольников будут лежать на одной горизонтальной прямой и иметь одинаковую длину. Высота каждого из прямоугольников пропорциональна соответствующему значению  $x_i$ . Программа:

```

program cml(input);
  type t=array[1..15] of integer;
  var x, y: t; m, i, a, z: integer;
  begin read(x[1]); m:=x[1];
    for i:=2 to 15 do
      begin read(x[i]);
        if x[i] > m then m:=x[i]
      end;
    for i:=1 to 15 do y[i]:=round(100*x[i]/m);
    a:=120; graphcolor mode;
    draw(10, a, 220, a, 1);
    for i:=1 to 15 do
      begin z:=10+15*(i-1);
        draw(z, a, z, a-y[i], 2);
        draw(z, a-y[i], z+7, a-y[i], 2);
        draw(z+7, a-y[i], z+7, a, 2);
        draw(z+7, a, z, a, 2)
      end
    end.
  end.

```

При выполнении этой программы одновременно с вводом определяется максимальное из заданных значений. После этого определяются значения  $y[1], \dots, y[15]$  — выраженные в количестве точек экрана высоты прямоугольников, соответствующих значениям  $x[1], \dots, x[15]$ . При этом максимальному значению из  $x[i]$  ( $i=1, \dots, 15$ ) соответствует высота, равная 100. После высвечивания на экране отрезка горизонтальной прямой, на которой будут лежать основания прямоугольников, выполняется собственно построение диаграммы. Построение выполняется в цикле. На каждом шаге цикла на экран выводится один прямоугольник с основанием длиной 7 и высотой  $y[i]$ . При этом пара значений  $z, a$  определяет положение левой нижней вершины прямоугольника на экране, пара значений  $z+7, a-y[i]$  — положение правой верхней вершины. При  $i=1, \dots, 15$  переменная  $z$  принимает значения 10, 25, 40, ..., 220.

На рис. 74 показана диаграмма, получающаяся при вводе в качестве исходных данных чисел 12, 18, 13, 20, 26, 21, 29, 23, 22, 19, 22, 27, 30, 32, 34. Диаграммы такого вида называются *столбчатыми диаграммами*.

Все диаграммы, которые можно получить с помощью рассмотренной программы, будут располагаться в прямоугольной области экрана, ограниченной отрезками, соединяющими точки, определяемые парами значений 10, 120, 10, 20, 227, 20 и 227, 120. Программу

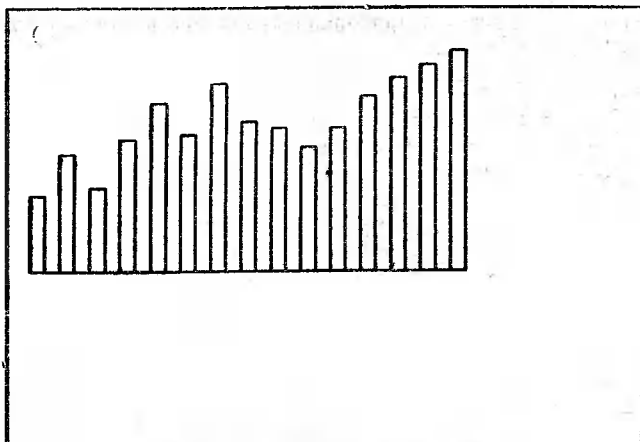


Рис. 74

можно усовершенствовать, если вместе с данными о количестве выпущенных изделий вводить информацию о размерах и месте расположения области экрана, в которой будет располагаться диаграмма. В усовершенствованной программе опишем процедуру построения прямоугольника. Параметрами процедуры будут значения, определяющие положение левого нижнего угла прямоугольника на экране, ширина и высота прямоугольника, а также цвет, в который он окрашивается. Программа:

```
program cm2(input);
  type mac=array[1..15] of integer;
  var x, y: mac; a, b, u, v: integer;
      m, z, i, s: integer;
  procedure box(x, y, l, h, c: integer);
    begin draw(x, y, x, y+h, c);
          draw(x, y+h, x+l, y+h, c);
          draw(x+l, y+h, x+l, y, c);
          draw(x+l, y, x, y, c);
```

```

fillshape(x+l div 2, y-h div 2, c, c)
end;
begin read(a, b, u, v);
read(x[1]); m:=x[1];
for l:=2 to 15 do
begin read(x[l]);
if x[l] > m then m:=x[l]
end;
for i:=1 to 15 do y[i]:=round(v*x[i]/m);
s:=u div 16; graphcolor mode;
draw(a, b, a+u, b, 1);
for i:=1 to 15 do
begin z:=a+s*(i-1);
box(z, b, s div 2+1, y[i], 2)
end
end.

```

При выполнении этой программы значения переменных  $a$  и  $b$  определяют положение левой нижней вершины прямоугольной области экрана, в которой выполняется построение диаграммы, значения

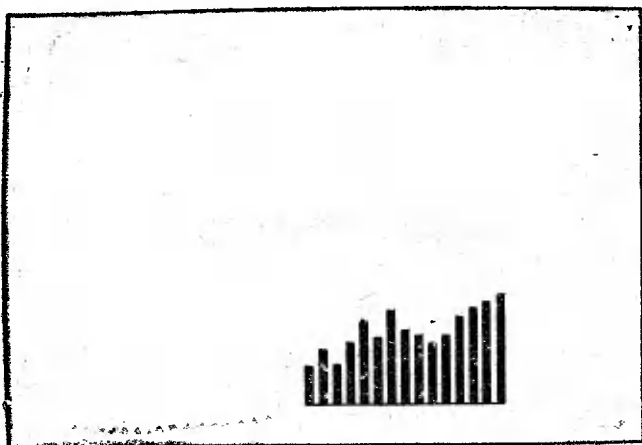


Рис. 75

переменных  $u$  и  $v$  — ширину и высоту этой области. После ввода исходных данных и отыскания максимального из данных значений определяются значения  $y[i]$  ( $i=1, \dots, 15$ ), равные высотам прямоугольников, соответствующих значениям  $x[i]$  ( $i=1, \dots, 15$ ). При этом максимальному из значений  $x[i]$  соответствует высота, равная значению переменной  $v$ . Затем переменная  $s$  получает значение, равное расстоянию на диаграмме между левыми нижними верши-

нами соседних прямоугольников. Построение самой диаграммы выполняется в цикле, на каждом шаге цикла строится и окрашивается в красный цвет прямоугольник ширины  $s \div 2 + 1$  и высоты  $y[i]$ . При тех же начальных данных, что и в предыдущем примере, перед которыми расположены числа 150, 180, 100, 50, получим диаграмму шириной 100 точек и высотой 50, смещенную к правому нижнему углу экрана (рис. 75).

### ЗАДАЧИ

**443.** Дано 30 целых чисел  $x_1, \dots, x_{15}, z_1, \dots, z_{15}$ . Рассматривая их как количества изделий, выпущенных двумя предприятиями за 15 дней, построить диаграмму, отдельные элементы которой

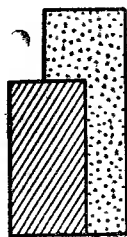


Рис. 76



Рис. 77

имеют вид, показанный на рис. 76. Два прямоугольника на рис. 76 соответствуют количеству изделий, выпущенных в один день на двух предприятиях и окрашиваются в разные цвета.

**444.** Написать программу построения диаграммы с элементами, имеющими вид как на рис. 77.

**445.** Информацию, содержащуюся в сообщении «зерновые убраны с  $x$  гектаров, из них на  $y$  гектарах обмолочены» графически можно представить прямоугольником, разделенным отрезком горизонтальной прямой на 2 части. При этом высота нижней части прямоугольника (см. рис. 78) относится к высоте всего прямоугольника, как  $y$  к  $x$ . Дано 30 натуральных чисел  $x_1, y_1, x_2, y_2, \dots, x_{15}, y_{15}$ , взятых из сообщения о состоянии дел с уборкой зерновых в 15 областях. Построить диаграмму с элементами указанного вида, соответствующую этим данным.



Рис. 78

**446.** Написать программу построения диаграммы, имеющей вид, как на рис. 79. Количество элементов диаграммы должно задаваться константой, определяемой в программе.

**447.** На столбчатых диаграммах отрицательным значениям измеряемой или вычисленной величины может соответствовать прямо-



угольник, расположенный ниже прямой, на которой лежат основания всех прямоугольников. На рис. 80 показано, как может выглядеть участок такой диаграммы. Написать программу построения такой диаграммы в прямоугольной области экрана, определяемой исходными данными. *Указание.* Здесь нужно будет определять положение прямой, на которой лежат основания прямоугольников, так,

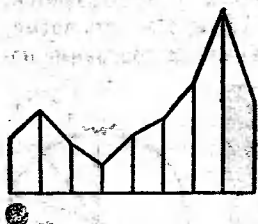


Рис. 79

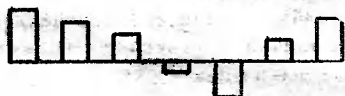


Рис 80

чтобы вся диаграмма поместилась в указанную прямоугольную область экрана.

448. Усовершенствовать программы, полученные при решении задач 444 и 446 с учетом того, что среди данных чисел могут встречаться отрицательные.

449. Оформить программу *ст2* в виде процедуры с параметрами, задающими положение диаграммы на экране, ее размеры, а также отображаемые с помощью диаграммы значения и цвет соответствующих этим значениям прямоугольников. Получить с помощью этой процедуры на экране три диаграммы разного цвета, отображающие динамику выпуска изделий за три пятидневки.

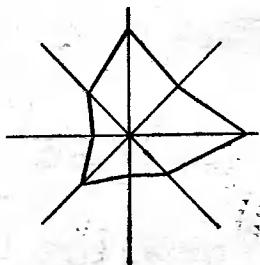


Рис. 81

450. Оформить построение диаграммы из задачи 445 в виде процедуры и получить на экране 7 диаграмм, показывающих положение дел с уборкой и обмолотом зерновых за неделю.

451. Даны натуральные  $v_1, v_2, \dots, v_8$ , задающие число дней за год, в которые преобладало, соответственно, северное, северо-восточное, восточное, юго-восточное, южное, юго-западное, западное или северо-

западное направление ветра. Построить розу ветров (рис. 81). Длины отрезков, соединяющих центр розы ветров с границами замкнутой ломаной, пропорциональны значениям  $v_1, v_2, \dots, v_8$ .

452. Написать программу, в результате выполнения которой упорядочиваются данные 20 чисел. По ходу выполнения программы получать на каждом шаге сортировки диаграмму, отражающую

текущее состояние упорядочиваемой последовательности чисел, задерживать ее на некоторое время на экране и затем уничтожать. Для того чтобы задержать изображение на экране, в соответствующее место программы можно вставить оператор цикла вида

*for l:=1 to n do f:=l\**

где *f* — переменная, значения которой больше нигде не используются, а *n* — некоторая константа. Меняя значения *n* можно менять время, на которое изображение задерживается на экране. В программе полезно описать процедуру построения диаграммы.

453. С помощью процедуры *box* получить на экране изображение шахматной доски.

454. Построить параллелепипед, три видимые грани которого окрашены в разные цвета (рис. 82).

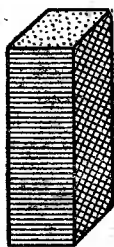


Рис. 82

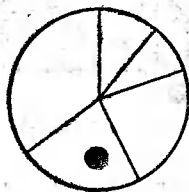


Рис. 83

455. Описав процедуру построения параллелепипеда, модифицировать программы *cm1* и *cm2* так, чтобы при их выполнении строились диаграммы, отдельные элементы которых являются не прямоугольниками, а параллелепипедами.

456. Секторная диаграмма — это круг, разделенный на секторы, площади которых пропорциональны данным значениям (рис. 83). Написать программу построения секторной диаграммы.

## § 31. Функции

В Паскале, помимо процедур, разрешены похожие на них конструкции, которые называются функциями. Обращение к функции приводит к вычислению ее значения — объекта типа *real*, *integer* или *char*. Тип значения фиксируется в описании функции.

Если значение некоторой функции (например, функции  $f(a, b)$ ) имеет тип *real* или *integer*, то способ употребления этой функции аналогичен способу употребления функций *sin*, *cos*, *round* и т. д.: в программе могут встретиться, например, операторы присваивания  $x := x + f(x, y)$ ,  $y := \sin(f(x, x)/2)$  или оператор цикла

*while f(x, y) < f(y, x) do x := f(x, x)*

Если же значение некоторой функции (например, функции  $h(a)$ ) имеет тип *char*, то эта функция может быть использована в программе наравне с конкретными символами, взятыми в кавычки. В программе может встретиться, например, оператор присваивания

$s := h(i)$ , условный оператор

$\text{if } s = h(i) \text{ then } s := 'a' \text{ else } s := h(i)$

и т. д.

Имеются две отличительные особенности описания функции в сравнении с описанием процедуры. Первая особенность связана с заголовком: он должен начинаться со служебного слова *function* \*) и заканчиваться именем того типа, которому принадлежит значение функции. Например:

```
function f(a: real; var b: t): real;  
function g(var a, b: integer): integer;  
function h(a: integer): char;
```

и т. д. Вторая особенность: составной оператор, который располагается в описании функции после заголовка и, возможно, после описания локальных меток и переменных, должен обязательно содержать внутри себя оператор присваивания, в котором слева от  $:=$  помещено имя функции, например:

```
f := 3.14  
g := a + 2 * b  
if a mod 2 = 0 then h := '*' else h := '1'
```

и т. д. Таких операторов присваивания может быть несколько, но при каждом конкретном обращении к функции «сработать» должен только один из них. Это присваивание и определит значение функции.

Относительно параметров функций имеется полная аналогия с параметрами процедур. Описания функций, как и описания процедур, располагаются в программе после совокупности описаний переменных.

Программу вычисления площади четырехугольника, варианты которой мы рассматривали в § 28, можно написать и так:

```
program F3(input, output);  
var AB, BC, CD, DA, AC: real;  
function triangl(a, b, c: real): real;  
var p: real;  
begin p := (a + b + c) / 2;  
      triangl := sqrt(p * (p - a) * (p - b) * (p - c))  
end;  
begin read(AB, BC, CD, DA, AC);  
      write(triangl(AB, BC, AC) + triangl(CD, DA, AC))  
end.
```

---

\*) Function — функция.

При составлении программы проверки гипотезы Гольдбаха можно было бы воспользоваться функцией *primer (a)*, описав ее таким образом, что

$$primer(a) = \begin{cases} 1, & \text{если } a \text{ — простое число,} \\ 0, & \text{если } a \text{ — составное число.} \end{cases}$$

Следующий пример будет связан с вычислением значений многочлена. Для вычисления значения  $a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  часто применяется экономный алгоритм, носящий имя английского математика XIX века У. Дж. Горнера. Согласно алгоритму Горнера данный многочлен предварительно преобразуется к виду

$$(((\dots(a_n x + a_{n-1})x + \dots)x + a_1)x + a_0,$$

а затем уже производятся вычисления. Более подробно: при вычислении значения  $y = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  последовательно находятся

[illegible]

Каждое следующее значение получается из предыдущего домножением на  $x$  и прибавлением очередного коэффициента многочлена:

```

y:=a[n];
for i:=n-1 downto 0 do y:=y*x+a[i]

```

Алгоритм Горнера требует для вычисления значения многочлена  $n$ -й степени  $2n$  операций:  $n$  сложений и  $n$  умножений.

Пусть даны действительные  $s, t, a_0, \dots, a_n$  ( $n$  — некоторая константа) и пусть требуется вычислить  $p(0.2) - p(t) + p^2(s-t) - -1.7p^3(0.2)$ , где  $p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ .

Программа (считаем, что  $n=7$ ):

```

program mn(input, output);
  const n=7;
  type T=array [0..n] of real;
  var a: T; s, t, u: real; i: integer;
  function p (x: real): real;
    var y: real; i: integer;
    begin y:=a[n];
      for i:=n-1 downto 0 do
        y:=y*x+a[i];
      p:=y
    end;

```

```

begin read (s, t);
  for i:=0 to n do read(a[i]);
  u:=p(0.2);
  write(u-p(t)+sqr(p(s-t))-1.7*u*u*u)
end.

```

Массив  $a_0, \dots, a_n$  не включен в число параметров процедуры, так как он не изменяется при различных обращениях к процедуре  $p$ .

Здесь мы вновь вернемся к тем функциям, которые используются в программах без предварительного описания — к функциям  $\sin$ ,  $\cos$ ,  $\exp$  и т. д. Набор этих функций (которые называются стандартными или встроенными функциями) достаточно широк. Большое число таких функций уже использовалось нами при составлении программ. Следующие встроенные функции так же являются очень полезными:

$\exp(E)$  — показательная функция  $E$ , т. е.  $e^E$ ;

$\ln(E)$  — натуральный логарифм  $E$ ;

$\arctan(E)$  — главное значение арктангенса  $E$ .

Например, если требуется вычислить  $a^r$  для  $a > 0$  и произвольного  $r$ , то можно взять значение выражения  $\exp(r \cdot \ln(a))$ . Значения функций  $\exp$ ,  $\ln$  и  $\arctan$  имеют тип *real*.

## ЗАДАЧИ

457. Используя функцию *primer* ( $a$ ), описанную таким образом, что

$$\text{primer}(a) = \begin{cases} 1, & \text{если } a \text{ — простое число,} \\ 0, & \text{в противном случае,} \end{cases}$$

составить новый вариант программы:

а) проверки гипотезы Гольдбаха;

б) поиска близнецов среди чисел

(см. задачу 421 из § 28).

458. Предложить новое решение задачи 440, используя при этом функции, принимающие значения 0 и 1 — образец такой функции см. в предыдущей задаче.

459. Предложить новое решение задачи 441, используя при этом функцию  $d(x_1, y_1, x_2, y_2)$ , значение которой равно расстоянию между точками с координатами  $x_1, y_1$  и  $x_2, y_2$ .

460. Даны действительные  $s, t$ . Получить  $g(1.2, -s) + g(t, s) - g(2s - 1, st)$ , где  $g(a, b) = \frac{a^2 + b^2}{a^2 + 3ab + 3b^2 + 4}$ .

461. Даны действительные  $s, t$ . Получить  $f(t, -2s, 1.17) + f(2.2, t, s - t)$ , где  $f(a, b, c) = \frac{2a - b - \sin c}{5 + |c|}$ .

462. Дано действительное  $y$ . Получить  $\frac{1.7f(0.25)+2f(1+y)}{6-f(y^2-1)}$ ,

$$\text{где } f(x) = \frac{\frac{x}{1!} + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!}}{\frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \frac{x^8}{8!}}.$$

463. Даны действительные  $a, b, c$ . Получить

$$\frac{\max(a, a+b) + \max(a, b+c)}{1 + \max(a+bc, 1.15)}.$$

464. Даны действительные  $a, b$ . Получить

$$u = \min(a, b), \quad v = \min(ab, a+b), \quad \min(u+v^2, 3.14).$$

465. Найти значения острых углов прямоугольного треугольника, заданного:

а) длиной катетов;

б) длиной катета и гипотенузы.

Указание. Воспользоваться функцией  $\arctan$ .

466. В процессе лечебного голодания вес пациента за 30 дней снизился с 96 до 60 кг. Было установлено, что ежедневные потери веса пропорциональны весу тела. Выяснить, чему был равен вес пациента во 2-й, 3-й, ..., 29-й день голодания.

Указание. Определить коэффициент пропорциональности с помощью функции  $\ln$ , затем воспользоваться функцией  $\exp$ .

467. Даны четыре символьных файла  $f1, f2, f3, f4$ . Вывести имя того из этих файлов, который имеет наибольшее число компонент (если таких файлов более одного, то вывести имя одного из них). Описать в программе функцию  $l(f)$ , значение которой равно количеству компонент файла.

468. Дано: натуральное  $n$ , действительные  $a_1, \dots, a_n$ . Требуется найти приближенные решения  $n$  уравнений:  $0.5(a_i x)^4 - \cos x = 0$  ( $i=1, 2, \dots, n$ ). Каждое из уравнений должно быть решено методом деления отрезка пополам. Корни разыскиваются на отрезке  $[0, \pi/2]$ , точность решения  $i$ -го уравнения равна  $1/(i+3)$ . В программе описать функцию  $f(t, x) = 0.5(tx)^4 - \cos x$  и функцию  $\text{root}(t, \text{eps})$  вычисления приближенного значения корня уравнения  $f(t, x) = 0$  при фиксированном  $t$  на отрезке  $[0, \pi/2]$  с точностью  $\text{eps}$  методом деления пополам.

469. Алгебраическое уравнение нечетной степени

$$a_n x^n + \dots + a_1 x + a_0 = 0$$

задано массивом коэффициентов  $a_0, \dots, a_n$  ( $n$  — константа с нечетным значением). Требуется найти с точностью 0.00001 какой-нибудь один корень рассматриваемого уравнения, применив для этого алгоритм деления пополам и алгоритм определения границ корней алгебраического уравнения, изложенный в задаче 103. Описать в программе функцию, реализующую алгоритм Горнера для вычисления значения многочлена.

470. Дан символьный файл, компонентами которого являются малые русские буквы. Требуется вывести компоненты файла, преобразуя их по следующему правилу: гласные заменяются буквой *г*, согласные — буквой *с*, буквы *ь*, *ъ*, *й* остаются без изменения. Описать в программе функцию, сопоставляющую букве, соответственно, букву *г*, *с*, *ь*, *ъ* или *й*.

471. Даны целые  $f_0, \dots, f_n$  ( $n$  — некоторая константа). Исследовать существование целочисленных корней уравнения  $f_n x^n + f_{n-1} x^{n-1} + \dots + f_0 = 0$ . (Если  $f_0 = 0$ , то имеется корень 0, если же  $f_0 \neq 0$ , то целочисленный корень, если он существует, принадлежит конечному множеству положительных и отрицательных делителей числа  $f_0$ .) Здесь полезно определить функцию вычисления значения многочлена с целыми коэффициентами по алгоритму Горнера, а также процедуру, которая по двум заданным целым числам  $k$  и  $m$  ( $m > k \geq 0$ ) позволяет определить значение наименьшего делителя числа  $m$ , содержащегося среди чисел  $k+1, k+2, \dots, m$ .

## § 32. Построение графиков функций

В этом параграфе будут рассмотрены приемы построения на экране компьютера графиков функций. Пусть, например, требуется построить график функции  $y = x^2 - 3$  на отрезке  $[-3, 3]$ . Кроме кривой, изображающей график этой функции, на экране должны быть высвечены координатные оси  $Ox$  и  $Oy$ . Договоримся располагать начало системы координат  $Oxy$  в середине экрана (т. е. в точке, определяемой парой чисел 160, 100). Необходимо условиться еще и о количестве точек экрана, соответствующих единице измерения в системе координат  $Oxy$ , т. е. о масштабном множителе. Пусть его значение равно 10. В этом случае положение точки графика с координатами  $(x, y)$  на экране определяется парой значений  $160 + 10 \cdot x$ ,  $100 - 10 \cdot y$ . Напишем программу построения графика:

```
program парабола;
  var x, y: real; i: integer;
  begin graphcolormode;
    draw(10, 100, 300, 100, 1); draw(170, 100, 170, 97, 1);
    draw(160, 10, 160, 190, 1); draw(160, 90, 163, 90, 1);
    for i := -30 to 30 do
      begin x := 0.1*i; y := x*x - 3;
        plot (round(160 + 10*x), round(100 - 10*y), 2)
      end
    end.
```

При выполнении этой программы на экране вначале высвечиваются горизонтальный и вертикальный отрезки, проходящие через сере-

дину экрана. На каждом из них отмечается точка, соответствующая единице в системе координат  $Oxy$  (рис. 84а). Затем в цикле с параметром  $i$  выполняется построение графика функции  $y = x^2 - 3$

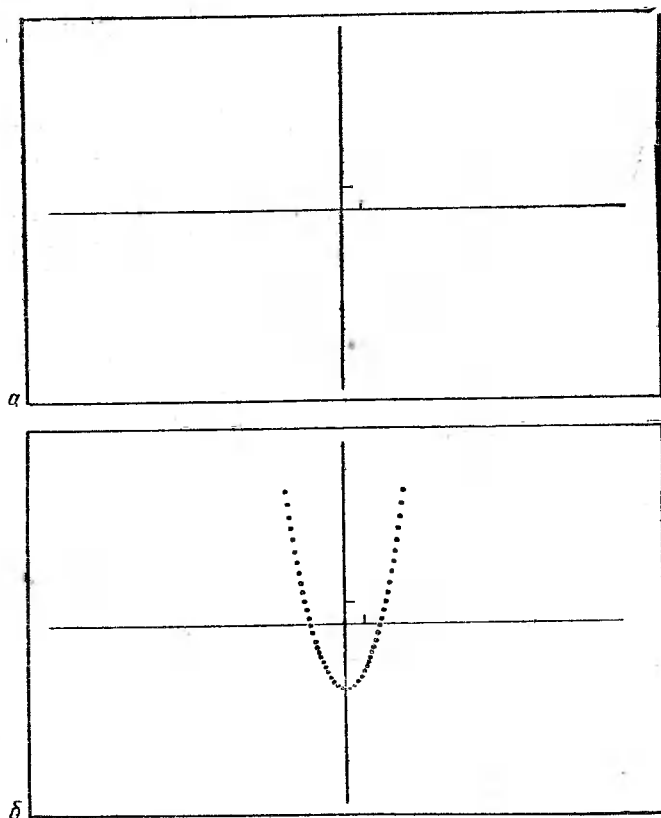


Рис. 84

(рис. 84б). В то время, как переменная  $i$  принимает значения  $-30, -29, -28, \dots, 0, 1, 2, \dots, 30$ , переменная  $x$  принимает значения  $-3, -2.9, -2.8, \dots, 0, 0.1, 0.2, \dots, 3$ , а выражение  $\text{round}(160 + 10 * x)$  — значения  $130, 131, 132, \dots, 160, 161, 162, \dots, 190$ . Таким образом последовательным значениям аргумента на экране будут соответствовать точки, расположенные в соседних столбцах. Этого мы достигли за счет выбора в качестве приращения аргумента величины, обратной масштабному множителю, а именно — числа 0.1. Можно увеличить приращение аргумента, например, заменить первый оператор в теле цикла на  $x := 0.2 * i$ , а заголовок цикла — на *for*  $i := -15$  *to* 15 *do*. Тогда график будет построен



примерно вдвое быстрее, но между соседними точками графика будут заметны просветы (рис. 85). Время построения графика можно уменьшить, не ухудшая качества изображения. Для этого нужно увеличить приращение аргумента и построить ломаную,

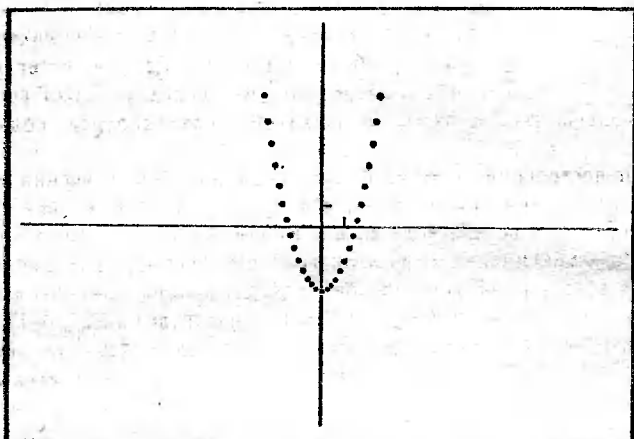


Рис. 85

состоящую из отрезков, соединяющих соседние точки графика. Перепишем программу *парабола* в соответствии со сказанным:

```

program nap1;
  var x, y: real; i, a, b, a1, b1: integer;
  begin graphcolormode;
    draw(10, 100, 300, 100, 1); draw(170, 100, 170, 97, 1);
    draw(160, 10, 160, 190, 1); draw(160, 90, 163, 90, 1);
    x:=-3; y:=x*x-3;
    a:=round(160+10*x); b:=round(100-10*y);
    for i:=-5 to 6 do
      begin x:=0.5*i; y:=x*x-3
        a1:=round(160+10*x); b1:=round(100-10*y);
        draw(a, b, a1, b1, 2);
        a:=a1; b:=b1
      end
    end.
  
```

При выполнении этой программы переменная  $x$  последовательно принимает значения  $-3, -2.5, -2, \dots, 0, 0.5, 1, \dots, 3$ . Значения  $a, b$  и  $a1, b1$  при выполнении оператора *draw* в теле цикла определяют положение на экране двух точек графика, соответствующих последовательным значениям аргумента.

Напишем теперь программу, позволяющую получать графики функций в любой части экрана. Пусть заданы четыре натуральных числа  $u, v, l, h$ , которые определяют положение левого нижнего угла, ширину и высоту прямоугольной области экрана, в которой должен быть построен график данной функции. Кроме того, задан отрезок, на котором строится график,  $-[a, b]$  и  $n$  — количество частей, на которые нужно разбить отрезок  $[a, b]$  при построении графика. Условимся, что начало системы координат  $Oxy$  всегда будет совмещаться с центром заданной прямоугольной области экрана.

При построении графика будут использоваться значения данной функции при  $x = a, a + s, a + 2s, \dots, b$ , где  $s = (b - a)/n$ . Рассмотрим абсолютные величины значений функции в этих точках. Пусть  $f_m$  — максимальная из этих величин. Тогда график функции не выйдет за верхнюю и нижнюю границы данной прямоугольной области, если масштабный множитель не превосходит  $[h/(2f_m)]$ . Пусть  $a_m = \max(|a|, |b|)$ . Тогда график функции не выйдет за левую и правую границы данной прямоугольной области, если масштабный множитель не превосходит  $[l/(2a_m)]$ . Следовательно, если взять в качестве значения масштабного множителя меньшую из величин  $[h/(2f_m)], [l/(2a_m)]$ , то график функции целиком поместится в данной прямоугольной области экрана. Программа:

```

program график (input);
  var u, v, l, h: integer;
      s, a, b, x, y, fm, am: real;
      m, i, n, u0, v0: integer;
      xp, yp, xpl, ypl: integer;
  function f (x: real): real;
    begin f := ... end;
  begin read(u, v, l, h); read(a, b, n);
    s := (b - a) / n; x := a; fm := abs(f(x));
    for i := 1 to n do
      begin x := x + s;
        if abs(f(x)) > fm then fm := abs(f(x))
      end;
    am := abs(a);
    if abs(b) > am then am := abs(b);
    m := trunc(h / (2 * fm));
    if m > trunc(l / (2 * am))
      then m := trunc(l / (2 * am));
    graphcolor mode;
    u0 := u + l div 2; v0 := v - h div 2;
    draw(u, v0, u + l, v0, 1);
    draw(u0 + m, v0, u0 + m, v0 - 2, 1);

```

```

draw(u0, v, u0, v-h, 1);
draw(u0, v0-m, u0+2, v0-m, 1);
x:=a; y:=f(x);
xp:=round(u0+m*x); yp:=round(v0-m*y);
for i:=1 to n do
    begin x:=x+s; y:=f(x);
          xp1:=round(u0+m*x);
          yp1:=round(v0-m*y);
          draw(xp, yp, xp1, yp1, 2);
          xp:=xp1; yp:=yp1
    end
end.

```

Здесь после отыскания значения масштабного множителя  $m$  выполняется построение графика функции  $f(x)$ . Значения  $u_0, v_0$  определяют положение начала координат на экране. График, как и в программе *par1* представляет собой ломаную линию с вершинами  $(a, f(a)), (a+s, f(a+s)), \dots, (b, f(b))$ . Для того, чтобы программа *график* могла быть выполнена, нужно заменить многоточие в описании функции  $f$  на арифметическое выражение, определяющее некоторую конкретную функцию (или заменить оператор присваивания  $f:=\dots$  на последовательность операторов, в результате выполнения которой вычисляется значение функции  $f$ ).

### ЗАДАЧИ

472. Изменить все рассмотренные в этом параграфе программы так, чтобы на координатные оси наносилась полная разметка (рис. 86).

473. Дано 15 чисел  $a_1, b_1, c_1, a_2, b_2, c_2, \dots, a_5, b_5, c_5$ . Каждая тройка чисел  $a_i, b_i, c_i$  ( $i=1, \dots, 5$ ) определяет параболу  $y=a_i x^2 + b_i x + c_i$ . Построить графики всех этих парабол на отрезке  $[-5, 5]$ . График параболы с номером  $i$  должен иметь цвет с номером  $i \bmod 3 + 1$ .

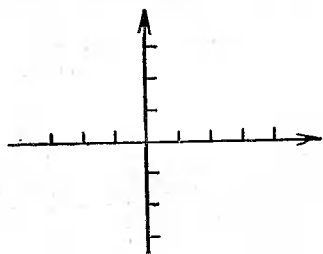


Рис. 86

474. Исследовав область определения и выбрав расположение координатных осей на экране и масштаб, построить графики функций:

а)  $y = \frac{1}{x+1}$ , б)  $y = \frac{x+3}{x-2}$ , в)  $y = 1 + \frac{2}{x} + \frac{3}{x^2}$ ,

г)  $y = 3 - \frac{2}{x} - \frac{1}{x^2}$ , д)  $y = \frac{1}{3x^2 + 2x + 1}$ , е)  $y = \frac{1}{x^2 + 2x + 1}$ .

475. Построить кривые по их уравнениям, заданным в полярных координатах (см. рис. 87а, б \*).

а) циссоида  $\rho = a \sin^2 \varphi / \cos \varphi$ ,

б) строфоида  $\rho = -a \cos 2\varphi / \cos \varphi$ ,

в) лемниската  $\rho = a \sqrt{2 \cos 2\varphi}$ .

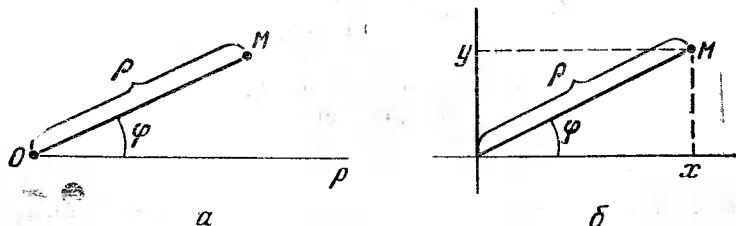


Рис. 87

476. Написать программу, в ходе выполнения которой одновременно с отысканием приближенного решения уравнения  $f(x) = 0$  с помощью алгоритма хорд (см. задачу 296 из § 20) строится иллюстрация применения этого алгоритма. График функции  $f(x)$ , хорды и вспомогательные вертикальные отрезки (рис. 88) должны быть окрашены в разные цвета.

477. В ходе приближенного вычисления площади криволинейной трапеции получить на экране иллюстрацию применения соответствующего алгоритма.

478. Таблица значений некоторой функции задана числами  $x_1, y_1, x_2, y_2, \dots, x_{50}, y_{50}$ , при этом  $x_1 < x_2 < \dots < x_{50}$ . Построить график этой функции.

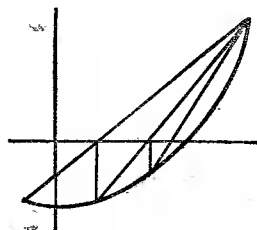


Рис. 88

479. Таблица значений некоторой функции задана числами  $x_1, x_2, \dots, x_{40}, y_1, y_2, \dots, y_{40}$ , при этом значения  $x_i$  ( $i = 1, \dots, 40$ ) не упорядочены. Упорядочить таблицу значений функции и построить график этой функции.

480. Таблица значений некоторой функции задана числами  $x_1, y_1, x_2, y_2, \dots, x_{40}, y_{40}$ , где  $x_1 < x_2 < \dots < x_{40}$ . Получить коэффициенты линейной зависимости  $y = kx + b$  с помощью алгоритма наименьших квадратов (см. задачу 299 из § 20) и высветить на экране точки  $x_i, y_i$  ( $i = 1, \dots, 40$ ) и прямую  $y = kx + b$ .

\* Полярные координаты  $\rho, \varphi$  точки  $M$  на плоскости — это расстояние  $\rho = OM$  от фиксированной точки  $O$  (полюса) до точки  $M$  и угол  $\varphi = \angle POM$  между  $OM$  и полярной осью (полупрямой)  $OP$ . Если совместить начало прямоугольных координат  $OXY$  с полюсом  $O$  так, чтобы ось  $OX$  совпадала с полярной осью  $OP$ , то будут справедливы соотношения между полярными и прямоугольными координатами точки:  $x = \rho \cdot \cos \varphi, y = \rho \cdot \sin \varphi$  (рис. 87а и б).

481. Метательное устройство находится на расстоянии 800 м от цели. Изменяя начальную скорость полета снаряда  $v_0$  и угол  $\varphi$ , под которым он выпускается, можно регулировать дальность полета. Написать программу с исходными данными  $v_0$  и  $\varphi$ . В левом нижнем углу экрана должен появляться прямоугольник, изобража-

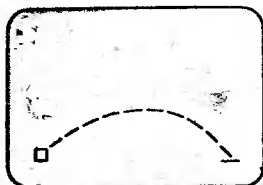


Рис. 89

ющий метательный аппарат, в правом нижнем углу — отрезок, изображающий цель. Точка, изображающая снаряд, должна переместиться по экрану от аппарата в сторону цели по параболе, определяемой исходными данными (рис. 89). В случае попадания снаряда в цель, отрезок, изображающий цель, исчезнет.

482. Написать игровую программу, позволяющую выполнять несколько попыток при «стрельбе» по цели. В случае попадания в цель она должна менять свое положение на экране. После окончания «стрельбы» должна сообщаться информация о количестве выполненных попыток и о количестве успешных попыток.

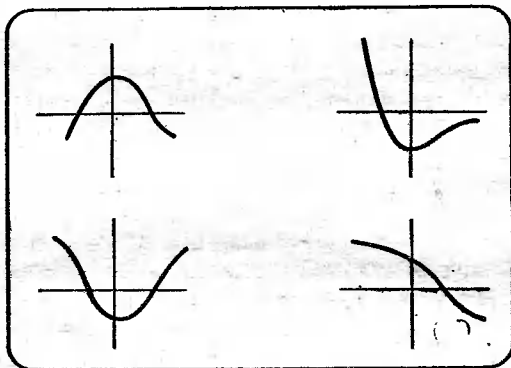


Рис. 90

483. Написать программу построения графика функции  $f(x)$  так, чтобы участки графика, на которых  $f(x) > 0$ , и участки, на которых  $f(x) < 0$  окрашивались в разный цвет.

484. В файле *числа* с компонентами типа *real* содержится 400 чисел. Каждая из четырех сотен чисел  $x_1, y_1, x_2, y_2, \dots, x_{50}, y_{50}$  ( $x_i < x_{i+1}; i = 1, \dots, 49$ ) определяет таблицу значений некоторой функции на отрезке  $[x_1, x_{50}]$ . Описать процедуру построения графика таблично заданной функции в прямоугольной области экрана, определяемой значениями параметров  $u, v, l, h$ . С помощью этой процедуры построить графики всех функций из файла *числа* в разных углах экрана (рис. 90).

## ГЛАВА VII

### БОЛЕЕ СЛОЖНЫЕ ПРИМЕРЫ АЛГОРИТМОВ И ПРОГРАММ

#### § 33. Лингвистический пример (перенос слова с одной строки на другую)

Преобразование текстов на естественных языках (русском, английском и т. д.) — одна из важных областей применения компьютеров. Возникающие здесь задачи оказываются довольно сложными, так как для применения грамматических правил обычно требуется понимание смысла слов и фраз (это можно наблюдать уже в задаче разбора слова по составу, т. е. в поиске приставок, корней, суффиксов, окончаний, и в задаче поиска главных и второстепенных членов предложения). Совершенно очевидно, что проведение компьютером смыслового анализа сопряжено со значительными трудностями. В некоторых задачах преобразования текстов удается избежать этого анализа, заплатив тем, что результат выполнения программы будет в ряде случаев содержать ошибки. Если процент ошибок не слишком высок, а сами ошибки — не слишком грубые, то, как правило, с этими ошибками можно мириться. Характерный пример — разделение слова на части для переноса. Многие тексты, получаемые на экране или бумаге с помощью компьютеров, являются инструкциями, справками, объявлениями и тому подобными документами канцелярского характера. Небольшие отступления от правил переноса, окупающиеся оперативностью изготовления документа, здесь вполне приемлемы (тем более, что в последнее время правила переноса становятся все менее жесткими).

Как показывают исследования, разделение русского слова на части для переноса с одной строки на другую с большой вероятностью будет выполнено без ошибок, если следовать трем простым правилам.

1) Две идущие подряд гласные можно разделить, если первой из них предшествует, а за второй—следует какая-нибудь буква (за-яц, ма-як и т. д.).

2) Две идущие подряд согласные можно разделить, если первой из них предшествует гласная, а в той части слова, которая идет за второй согласной, имеется хотя бы одна гласная (бод-рость, брат-ство и т. д.).

3) Если не удастся применить правила 1), 2), то следует попытаться разделить слово так, чтобы правая часть содержала более, чем одну букву и оканчивалась на гласную, а вторая содержала более, чем одну букву, среди которых имеется хотя бы одна гласная (пря-мо, гру-ша и т. д.).

В правилах 1)—3) предполагается, что буква *й* присоединяется к предыдущей гласной и рассматривается вместе с ней как единая гласная (зай-цы, смей-ся и т. д.), и точно так же буквы *ь* и *э* присоединяются к предшествующей согласной (боль-ше, объ-езд и т. д.).

Приведенные правила позволяют разделять слово на несколько частей (баранка → баран-ка → ба-ран-ка, картошка → кар-тошка → кар-тош-ка и т. д.); после такого разделения можно выбрать тот или иной вариант переноса. Однако, для простоты, мы остановимся на задаче поиска какого-нибудь одного места переноса.

Перед разработкой программы наметим некоторые ее компоненты. Во-первых, при разборе слова нет необходимости различать гласные буквы между собой, точно так же можно не различать между собой и согласные буквы. Однако необходимо относительно каждой буквы установить, является ли она, соответственно, гласной, согласной или же одной из букв *й*, *ь*, *э*. Поэтому будет полезно перейти от слова к набору букв, отличающемуся от исходного слова тем, что каждая гласная заменена буквой *г*, каждая согласная—буквой *с*, а каждая из букв *й*, *ь*, *э* оставлена без изменения: кошка → *сгсгсг*, больше → *сгсбсг*, зайцы → *сгйсг*, съезжал → *сгсгсгсг* и т. д. Этот набор букв назовем *макетом* слова.

Из правил 1)—3) вытекает, что перенос возможен не раньше, чем после первой гласной, а если эта гласная—самая первая буква слова, то и не раньше, чем после второй буквы. Ясно также, что перенос нельзя выполнить позднее, чем перед последней гласной слова, а если эта гласная—вообще последняя буква в слове, то позднее, чем перед предпоследней буквой. При определении начальной и конечной частей слова, внутри которых заведомо невозможен перенос, надо еще учесть, что буквы *й*, *ь*, *э* нельзя отделять от предшествующей буквы. Например, для слова, макет которого есть *сгсгсгйсг* перенос возможен заведомо не раньше, чем после второй буквы и не позднее, чем после седьмой буквы; для слова, макет которого есть *сгсгсбг*,— не раньше, чем после второй буквы и не позднее, чем после третьей буквы; для слова, макет

которого есть *sgig* перенос невозможен. В слове, состоящем менее, чем из четырех букв перенос невозможен, так как начальная и конечная неделимые части слова должны содержать не менее двух букв каждая.

Итак, пусть в слове (*n*, следовательно, в его макете) содержится *n* букв. Пусть уже найдены (с учетом приведенных выше соображений) натуральные *k* и *m*, которые показывают, что перенос заведомо невозможен раньше, чем после *k*-й буквы и позднее, чем после *m*-й буквы. Пусть  $n \geq 4$  и  $m \geq k$ . Тогда можно, в соответствии с правилами 1), 2), 3), поступить следующим образом:

1') в части макета, содержащей буквы с *k*-й по (*m*+1)-ю, попытаться найти сочетание *gg* или *yg*;

2') если предыдущий пункт не привел к успеху, то в той же части макета попытаться найти сочетание *ss* или *ys* (в случае успешного поиска надо будет взять самое первое такое сочетание);

3') если предыдущие пункты не привели к успеху, то в той же части макета попытаться найти букву *y* или букву *g*, за которой не следует буква *y*.

При успешном поиске мы сможем указать место переноса (порядковый номер последней буквы слова, остающейся в верхней строке), при безуспешном — объявим, что перенос невозможен (укажем в качестве номера число 0).

Будем считать, что последовательность букв слова задана во входном файле. Будем также считать, что число букв в этой последовательности не превосходит 25. В программе можно использовать массив  $I_1, \dots, I_{25}$  с элементами типа *char*; первые *n* элементов массива будут иметь значения букв макета слова, остальные элементы рассматриваться не будут (*n* — число букв в исходном слове). Результатом выполнения программы будет значение переменной *t*. Схема программы:

```
program перенос(input, output);
  описания;
  begin
    получить n — число букв n
     $I_1, \dots, I_n$  — макет данного слова;
     $t := 0$ ;
    if  $n < 4$  then goto 0;
    определить натуральные k и m, о
    которых говорилось выше;
    среди  $I_k, \dots, I_{m+1}$  попытаться найти сочетание  $I_i$ ,
     $I_{i+1}$  вида gg или yg,
    если сочетание найдено, то
      begin  $t := i$ ; goto 0 end;
    среди  $I_k, \dots, I_{m+1}$  попытаться найти
```



сочетание  $l_i, l_{i+1}$  вида *сс* или *вс*,  
 если сочетание найдено, то  
     *begin*  $t := i$ ; *goto* 0 *end*;  
 среди  $l_k, \dots, l_{m+1}$  попытаться найти  
 $l_i$  — букву *й* или букву *з*, за которой не  
 следует буква *й*, если буква найдена, то  
     *begin*  $t := i$ ; *goto* 0 *end*;  
     0: *write*( $t$ )  
*end*.

Нетрудно описать функцию, которая позволяет получать, исходя из данной русской буквы, соответственно одну из букв *з*, *с*, *й*, *в*, *з* так, как это требуется при построении макета. Этой функции мы дадим в программе имя  $M$ . Построение макета и вычисление  $n$  можно задать операторам цикла

```

 $n := 0$ ;
while not eof(input) do
  begin  $n := n + 1$ ; read( $l[n]$ );  $l[n] := M(l[n])$  end

```

но мы выберем другое решение, так как во время построения макета можно найти порядковые номера первой и последней гласной слова:

```

 $n := 0$ ;  $k := 0$ ;
while not eof(input) do
  begin  $n := n + 1$ ; read( $l[n]$ );  $l[n] := M(l[n])$ ;
    if  $l[n] = 'з'$  then
      begin  $m := n$ ;
        if  $k = 0$  then  $k := n$ 
      end
    end
end

```

Переменная  $m$  последовательно принимает значения номеров всех гласных слова, а после выполнения этих операторов имеет значение номера последней гласной. При обнаружении гласной переменная  $k$  получает значение номера этой гласной только в том случае, когда до этого выполнялось  $k = 0$ . Предполагается, что в слове, которое содержит не менее четырех букв, имеется хотя бы одна гласная; таким образом, после выполнения выписанных операторов переменные  $k$  и  $m$  имеют ненулевые значения:  $k$  — номер первой гласной,  $m$  — последней.

Полученные предварительные значения  $k$  и  $m$ , возможно, придется еще несколько подправить: во-первых, нельзя оставлять на строке одну букву и нельзя переносить одну букву, поэтому добавим операторы

```

if  $k = 1$  then  $k := 2$ ;
if  $m = n$  then  $m := n - 2$  else  $m := m - 1$ 

```

во-вторых, каждую из букв *й*, *б*, *э* нельзя отрывать при переносе от предшествующей буквы, поэтому включим еще операторы

```
if (l[k]='й') or (l[k]='б') or (l[k]='э')
  then k:=k+1;
if (l[m]='й') or (l[m]='б') or (l[m]='э')
  then m:=m-1
```

Пусть, например, первоначально было задано слово *альбом*. Его макет—*гсьсгс*, в процессе построения макета будет получено  $k=1$ ,  $m=5$  (номера первой и последней гласной). Далее, значениями  $k$  и  $m$  станут числа 2 и 4, но так как третья буква макета—это *б*, то значением  $k$  станет 3. Итак,  $k=3$ ,  $m=4$ , т. е. перенос невозможен раньше, чем после *аль* и позже, чем перед *ом*. После этих вычислений значений  $k$  и  $m$  можно приступить к применению правил 1'), 2'), 3'), как это указано в схеме программы.

Детализируем применение правила 1'):

```
for i:=k to m do
  if ((l[i]='э') or (l[i]='й')) and (l[i+1]='э')
    then begin t:=i; goto 0 end
```

правило 2'):

```
for i:=k to m do
  if ((l[i]='с') or (l[i]='б')) and (l[i+1]='с')
    then begin t:=i; goto 0 end
```

и, наконец, правило 3'):

```
for i:=m downto k do
  if (l[i]='й') or (l[i]='э')
    then begin t:=i; goto 0 end
```

Бросается в глаза, что при выполнении последнего оператора цикла выбранная часть макета просматривается в поисках буквы *г* или *й* от конца к началу. Это удобно, по крайней мере, в том отношении, что при встрече буквы *г* отпадает необходимость дополнительного сравнения следующей за ней буквы с *й* (при используемом порядке просмотра эта буква уже проверена). Этим путем мы приходим к программе:

```
program перенос(input, output);
  label 0;
  type v=array[1..25] of char;
  var l: v; i, t, n, k, m: integer;
  function M(x:char):char;
    begin if (x='й') or (x='б') or (x='э')
      then M:=x
      else if (x='а') or (x='е') or (x='у') or
```

```

        (x='o') or (x='y') or (x='bl') or
        (x='s') or (x='ю') or (x='я')
    then M:='e' else M:='c'
end;
begin n:=0; k:=0;
    while not eof(input) do
        begin n:=n+1; read(l[n]); l[n]:=M(l[n]);
            if l[n]='e' then
                begin m:=n;
                    if k=0 then k:=n
                end
            end;
        t:=0; if n < 4 then goto 0;
        if k=1 then k:=2;
        if m=n then m:=n-2 else m:=m-1;
        if (l[k]='ü') or (l[k]='b') or (l[k]='o') then k:=k+1;
        if (l[m]='ü') or (l[m]='b') or (l[m]='o') then m:=m-1;
        for i:=k to m do
            if ((l[i]='e') or (l[i]='ü')) and (l[i+1]='e')
                then begin t:=i; goto 0 end;
        for i:=k to m do
            if ((l[i]='c') or (l[i]='b')) and (l[i+1]='c')
                then begin t:=i; goto 0 end;
        for i:=m downto k do
            if (l[i]='ü') or (l[i]='e')
                then begin t:=i; goto 0 end;
    0: write(t)
end.

```

Выше были даны примеры, когда применение этого алгоритма приводит к абсолютно правильному результату. Укажем теперь такие примеры, когда получаемый результат нельзя признать удачным (полезно самостоятельно разобраться, в чем именно состоит неудачность разделения слов): *прислать*, *пятиграммовый*, *поссорить*, *нововведение*.

Наконец, отметим еще один недостаток, связанный с оформлением алгоритма. Дело в том, что обычно такого рода алгоритмы оформляются не в виде законченных программ, а в виде процедур, или функций, так как по самому смыслу задачи этот алгоритм, скорее всего, будет играть лишь вспомогательную роль в некотором более общем алгоритме, таком, как, например, алгоритм печати текста при заданном формате страницы. Мы описали этот алгоритм в виде программы, поскольку не имеем возможности разбирать эти более сложные задачи.

## ЗАДАЧИ

485. Рассмотреть первое предложение настоящего параграфа («Преобразование текстов на естественных языках...») и применить разобранный алгоритм к каждому из слов этого предложения. Оценить качество результатов.

486. Переделать программу *перенос* так, чтобы выводилось не значение  $t$ , а само исходное слово, разделенное в нужном месте знаком переноса (дефис).

487. Описать в программе *перенос* процедуру без параметров для выполнения действий:  $t := i$ ; *goto* 0 и использовать обращения к этой процедуре в соответствующих местах программы.

488. Описать в программе *перенос* функцию *особ* ( $x$ ), значением параметра которой является буква, а значением самой функции — число 0 или 1, при этом *особ* ( $x$ ) = 1, если значением  $x$  является буква *й*, *ь* или *ѣ* и 0 в противном случае. Использовать обращения к этой функции в соответствующих местах программы.

489. Нетрудно заметить, что при работе с макетом слова буквы *ь* и *ѣ* можно не различать, поэтому и при построении самого макета можно, скажем, букву *ѣ* заменять на *ь*. Используя это соображение, упростить программу *перенос*.

490. В программу *перенос* внести изменение, снимающее предположение о том, что слово содержит хотя бы одну гласную букву. *Указание.* Воспользоваться тем, что если в слове нет гласных букв, то после выполнения оператора цикла *while not eof(input) do ...* значение переменной  $k$  будет равно нулю.

491. Переделать программу *перенос* так, чтобы место переноса разыскивалось между буквами с данными номерами  $r$  и  $s$  (задача поиска места такого переноса реально встает при печати текста с заданным форматом страницы).

492. Переделать программу *перенос* так, чтобы по возможности отыскивались все места, в которых можно сделать перенос в данном слове.

493. Усовершенствовать программу *перенос*, включив в нее небольшой словарь приставок. Установить, что приставка не может разбиваться знаком переноса, но если приставка содержит гласную, то перенос может быть сделан сразу после приставки.

494. Для большинства существительных, с суффиксами *енок* и *енок*, множественное число образуется от другой основы. Как правило, это происходит по образцу: *цыпленок* — *цыплята*, *мышонок* — *мышата* и т. д. (в новой основе перед последней буквой *т* пишется *а* или *я* в зависимости от предыдущей буквы: если это шипящая, то *а*, иначе — *я*). Имеются слова — исключения, из которых укажем следующие: *ребенок* — (*дети*), *бесенок* (*бесенята*), *опенок* (*опята*), *звонок* (*звонки*), *позвонок* (*позвонки*), *поденок* (*подонки*), *живо-*

ронок (жаворонки), бочонок (бочонки). Есть еще ряд малоупотребительных слов — исключений, которые мы не рассматриваем.

Дано слово, оканчивающееся на *онок* или *енок*. Получить это слово во множественном числе.

495. Рассмотрим существительные мужского рода, оканчивающиеся на *ок*: *кружок*, *масленок*, *брелок* и т. д. При склонении таких слов буква *о* может выступать как беглая гласная: *кружка*, *масленком* и т. д. При склонении некоторых таких слов гласная *о*, однако, сохраняется. Это, во-первых, слова из трех букв — *ток*, *сок* и т. д.; затем слова *скок*, *блок*, *волок*, *восток*, *шток* и слова, основа которых оканчивается на такие сочетания букв (т. е. *перескок*, *пищелок*, *юго-восток* и т. д.); наконец, имеется и еще ряд слов, среди которых укажем следующие: *брелок*, *щелок*, *войлок*, *челнок*, *зарок*. *срок*, *урок*, *знаток*, *поток*, *сток*, *артишок*.

Дан текст, среди символов которого имеется пробел. Группа символов, предшествующая первому пробелу, представляет собой русское слово — существительное мужского рода на *ок*; после первого пробела идет одна из букв *и*, *р*, *д*, *в*, *т*, *н*, указывающая падеж. Получить слово в указанном падеже.

496. Дано натуральное  $n$ , равное выраженной в копейках цене некоторого товара, например, 317, 5005, 100 и т. д. Выразить цену в рублях и копейках, например 3 рубля 17 копеек, 50 рублей 5 копеек, 3 копейки, 1 рубль (рубли или копейки могут не указываться, если их число — 0).

497. Дано натуральное  $n$  ( $n \leq 1000$ ). Записать это число русскими словами (семнадцать, двести пятьдесят три, тысяча и т. д.).

498. Дано натуральное  $n$ , символ  $s$  ( $n \leq 1000$ ,  $s$  — одна из букв *и*, *р*, *д*, *в*, *т*, *н*, указывающая падеж). Записать числительное, обозначающее  $n$ , в соответствующем падеже (ср. с предыдущей задачей).

## § 34. Дифференциальные уравнения

В современной науке и технике часто предпринимаются исследования протекающих во времени процессов. Эти процессы могут носить различный характер: физический (движение тела, жидкости, газа, изменение температуры, давления и т. д.), химический (изменение количества какого-либо вещества во время реакции), биологический (изменение численности конкурирующих популяций) и т. д. Первым этапом исследования таких процессов довольно часто является составление *дифференциального уравнения*, описывающего процесс, а вторым — поиск решения этого уравнения. При решении задач второго этапа в последние десятилетия активно используются компьютеры. Ниже в этом параграфе будет описан один классический алгоритм численного решения такого

рода уравнений. Предварительно потребуется ряд напоминаний и определений \*).

1) Пусть дана гладкая функция  $y = \varphi(t)$  \*\*, т. е. функция, график которой представляет собой непрерывную плавную линию

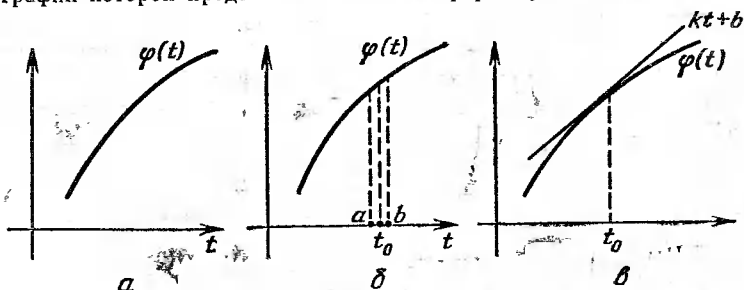


Рис. 91

без изломов (рис. 91а). Если взять некоторое значение аргумента  $t_0$  и рассмотреть данную функцию на маленьком окружающем  $t_0$  интервале  $(a, b)$ , то, ввиду малости интервала, график функции можно будет приближенно считать отрезком прямой (рис. 91б), которая проходит через точку с координатами  $(t_0, \varphi(t_0))$  и которая среди всех проходящих через эту точку прямых наиболее плотно прилегает к графику функции  $\varphi(t)$  (рис. 91в). Такая прямая называется касательной к графику в рассматриваемой точке. Сопоставив каждому значению аргумента  $t_0$  угловой коэффициент (тангенс угла наклона) касательной  $y = kt + b$ , проведенной к графику в точке  $(t_0, \varphi(t_0))$ , мы получим новую функцию, которая обозначается  $\varphi'(t)$  и называется производной функции  $\varphi(t)$ .

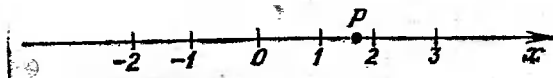


Рис. 92

2) Важность понятия производной для многочисленных приложений связана, в частности, с тем, что производная может рассматриваться как скорость изменения значения функции. Пусть, например, точка  $P$  движется по некоторой прямой линии (рис. 92). Рассмотрим эту прямую как координатную ось. То, что точка  $P$

\*) Для чтения этого параграфа желательно знакомство с понятием производной функции и с приемами нахождения производных (дифференцированием). Не предполагая, однако, у читателя прочных навыков в этом, мы сформулируем (бегло и не строго) некоторые начальные сведения.

\*\*) Для обозначения аргумента мы выбираем  $t$ , так как в дальнейшем аргумент будет интерпретироваться как время.

движется, означает, что координата  $x$  этой точки меняется со временем. Эта координата является, таким образом, функцией от времени  $t$ , которую мы обозначим через  $x(t)$ . Оказывается, что скорость точки  $P$  в момент времени  $t$  равна  $x'(t)$ . Если построен график зависимости координаты  $x$  точки  $P$  от времени  $t$  (рис. 93), то для вычисления скорости  $v(t_0)$  в момент  $t_0$  можно провести касательную в той точке графика, которая имеет абсциссу  $t_0$ , и определить угловой коэффициент — это и будет значение скорости.

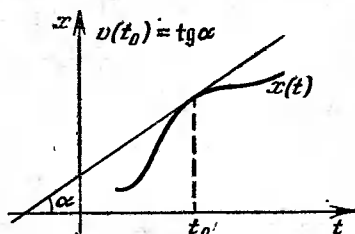


Рис. 93

3) Оказывается также, что ускорение  $a(t)$  точки  $P$ , движущейся по прямой (см. предыдущий пункт) является производной функции  $x(t)$ , т. е.  $x''(t)$  \*). Эта функция называется второй производной (или производной второго порядка). Иногда рассматриваются производные более высоких порядков: производная третьего порядка или третья производная  $x'''(t)$ , производная четвертого порядка или четвертая производная  $x^{(4)}(t)$  и т. д.



Рис. 94

4) Известны многочисленные правила дифференцирования функций, т. е. правила нахождения производных. Можно, например, показать, что производная постоянной функции равна нулю,  $t' = 1$ ,  $(t^2)' = 2t$ ,  $(\sin t)' = \cos t$  и т. д. На этих правилах, несмотря на их чрезвычайную важность, мы не имеем возможности останавливаться.

5) Положение секущей, проходящей через точку  $(t_0, \varphi(t_0))$  и через близкую к ней точку  $(t_0 + h, \varphi(t_0 + h))$ , где  $h$  — малое по модулю число (рис. 94), близко к положению касательной в точке  $(t_0, \varphi(t_0))$  (строго говоря, касательная — это предельное положение секущей при неограниченном приближении  $h$  к нулю). Угловым коэффициентом секущей равен, как нетрудно заметить,  $\frac{\varphi(t_0 + h) - \varphi(t_0)}{h}$ .

\*) Ускорение — это скорость изменения скорости.

равенство

$$\varphi'(t_0) \approx \frac{\varphi(t_0+h) - \varphi(t_0)}{h}.$$

б) Подчеркнем, что функция может не быть гладкой. Так, например, график функции  $|t|$  имеет излом при  $t=0$ , соответственно, производная этой функции не определена при  $t=0$ . В дальнейшем мы в этом параграфе без оговорок будем считать, что рассматриваемые функции имеют производные.

Обсуждение дифференциальных уравнений начнем с примера. Второй закон Ньютона — «сила, действующая на тело, равна произведению массы тела на сообщаемое этой силой ускорение», для силы изменяющейся со временем по известному закону и направленной вдоль фиксированной прямой, может быть записан в виде

$$F(t) = mx''(t),$$

где  $x(t)$  обозначает координату материальной точки массы  $m$ . Координата берется на рассматриваемой прямой. В свою очередь  $F(t)$  — это сила, действующая на точку в момент времени  $t$  (сила положительна, если она направлена в сторону возрастания координаты и отрицательна в противном случае). Если  $x(t)$  — неизвестная функция, а  $F(t)$  — известная, то закон Ньютона можно рассматривать как некоторое уравнение для  $x(t)$ . Это — пример дифференциального уравнения. Дифференциальным уравнением называется равенство, связывающее значения аргумента, неизвестной функции и некоторых ее производных. Наивысший порядок входящих в уравнение производных называется *порядком дифференциального уравнения*. В рассмотренном примере дифференциального уравнения второго порядка значение аргумента  $t$  связано со значением второй производной  $x''(t)$  с помощью известной функции  $F(t)$ . Уравнение, являющееся выражением второго закона Ньютона, может иметь и более сложный вид, если сила зависит не только от времени, но и от координаты и от скорости. Подчеркнем, что во всех случаях неизвестной в дифференциальном уравнении является функция, а не число.

Нами не случайно был проведен «ньютоновский» пример дифференциального уравнения. Именно Ньютон обогатил науку идеей, что любой достаточно сложный процесс, характеризующийся изменением некоторой величины, надо пытаться описывать с помощью такого уравнения, которое связывает время  $t$  не только со значением  $x(t)$ , но и с некоторыми из значений  $x'(t)$ ,  $x''(t)$ , ..., т. е. со скоростью, ускорением и т. д. изменения  $x(t)$ ; это соответствует, по Ньютону, характеру законов природы.

Ньютон использовал для обозначения производных по  $t$  (т. е. производных по времени) не штрихи, а точки, надписанные над буквенными обозначениями функции: производная  $x(t)$  по  $t$  запи-



сывалась им как  $\dot{x}(t)$  или  $\dot{x}$ , вторая производная—как  $\ddot{x}(t)$  или  $\ddot{x}$  и т. д. Эта символика часто используется и в настоящее время. В оставшейся части настоящего параграфа мы будем пользоваться именно этой символикой.

Взгляды Ньютона на характер законов природы оказали сильное влияние на дальнейшие исследования, и большое число фундаментальных законов было описано с помощью дифференциальных уравнений. Это же касается огромного количества конкретных процессов (величина  $x$ —это не обязательно координата, это может быть температура, давление, энергия, численность популяции, какой-либо экономический показатель и т. д.). Например, согласно закону излучения тепла, скорость изменения температуры тела в воздухе пропорциональна разности между температурой воздуха и тела:

$$\dot{x}(t) = \alpha (u(t) - x(t)),$$

где  $x(t)$ —температура тела в момент времени  $t$ ,  $u(t)$ —температура воздуха в этот же момент,  $\alpha$ —положительный коэффициент, определяемый свойствами тела. Если  $\alpha = 0.17$  и температура воздуха зависит от времени  $t$  как  $20 - 1/(1+t^2)$ , то уравнение запишется в виде

$$\dot{x}(t) = 0.17 \left( 20 - \frac{1}{1+t^2} - x(t) \right)$$

и т. д.

Решения некоторых из дифференциальных уравнений могут быть получены в виде достаточно простых формул, явно выражающих  $x(t)$  через  $t$ . Однако обнаруживается, что для многих уравнений таких формул не существует, или же эти формулы трудно найти. В этих случаях прибегают к численному решению уравнений: вместо формулы для функции  $x(t)$  получают приближенные значения функции  $x(t)$  для ряда значений аргумента  $t_0, t_1, \dots, t_n$ . Точно также приходится довольствоваться численным решением, когда значения некоторой функции, входящей в уравнение (например функции  $u(t)$ , входящей в уравнение, описывающее изменение температуры тела) известны только для отдельных значений аргумента.

Мы опишем теперь алгоритм Эйлера численного решения дифференциальных уравнений первого порядка, имеющих, подобно уравнению излучения тепла, следующий вид:

$$\dot{x}(t) = f(t, x(t))$$

или, короче,

$$\dot{x} = f(t, x),$$

где  $f(t, x)$ —известная функция двух аргументов, значение которой плавно изменяется при изменении значений аргументов. Затем будут рассмотрены более сложные уравнения.

Сущность алгоритма Эйлера становится совершенно понятной, если посмотреть на задачу решения дифференциального уравнения с геометрической точки зрения.

Графики функций, удовлетворяющих дифференциальному уравнению, называются *интегральными кривыми* \*) этого уравнения. Геометрически задача решения дифференциального уравнения является задачей построения интегральных кривых. Уравнение  $\dot{x} = f(t, x)$  имеет прозрачный геометрический смысл: пусть некоторая интегральная кривая проходит через точку  $(t_0, x_0)$ , тогда в этой точке угловой коэффициент касательной к этой интегральной кривой равен  $f(t_0, x_0)$ . Если, например,  $f(t, x) = t - x$ ,  $t_0 = 1$ ,  $x_0 = 2$ , то  $f(t_0, x_0) = -1$  и мы можем указать на чертеже направление, в котором интегральная кривая проходит через точку  $(t_0, x_0)$  (рис. 95). Пусть теперь  $A$  — некоторая точка нашей плоскости. Проведем из этой точки небольшой прямолинейный отрезок  $AB$  в том направлении, в котором интегральная кривая проходит через точку  $A$  (рис. 96а); отрезок  $AB$  почти совпадает с дугой интегральной кривой, проходящей через  $A$ . Определяем направление интегральной кривой в точке  $B$  и проводим в этом направлении отрезок  $BC$  (рис. 96б), затем этим же способом проводим отрезок  $CD$  (рис. 96в) и т. д. Мы получим *ломаную Эйлера*

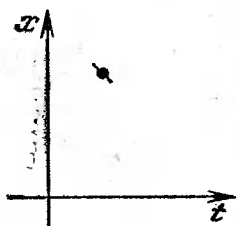


Рис. 95

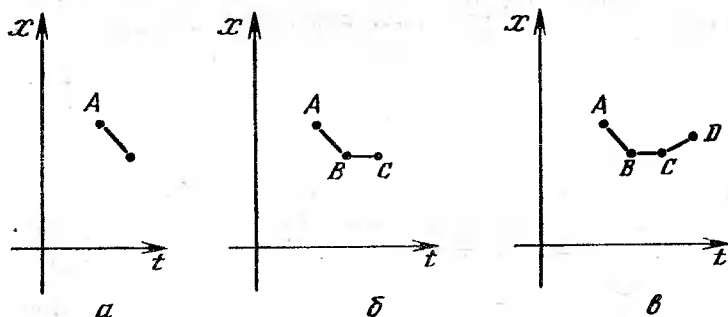


Рис. 96

$ABCD \dots$ , которая приближенно может быть принята за интегральную кривую. Естественно, что чем мельче отрезки, тем более точное приближение интегральной кривой мы получим.

\*) Это название связано с тем, что в простейших случаях решение дифференциальных уравнений сводится к интегрированию.

Перейдем теперь от геометрии к формулам, уточнив предварительно постановку задачи.

Прежде всего, следует отметить, что уравнение  $\dot{x} = f(t, x)$  имеет бесконечное множество решений (рис. 97) — через каждую точку плоскости проходит интегральная кривая (напомним, что мы строили ломаную Эйлера, выходя из произвольной точки  $A$ ). Чтобы выделить одну кривую, достаточно указать точку плоскости, через которую проходит кривая, т. е. указать значение  $x(t)$  для некоторого  $t = t_0$ .

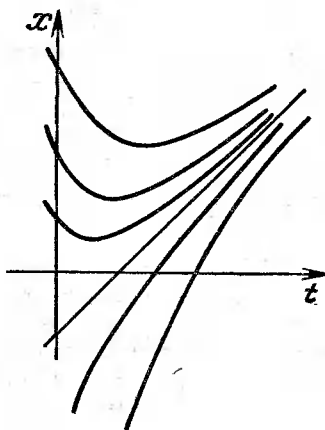


Рис. 97

Будем рассматривать уравнение  $\dot{x} = f(t, x)$  на некотором отрезке  $[a, b]$ , т. е. рассматривать его для значений  $t$ , удовлетворяющих неравенству  $a \leq t \leq b$ , где  $a$  и  $b$  — данные числа. Пусть  $n$  — натуральное число и  $h = \frac{b-a}{n}$ .

Обозначим  $t_0 = a$ ,  $t_1 = a + h$ ,  $t_2 = a + 2h$ , ...,  $t_n = a + nh = b$ . Пусть дано число  $x_0$ . Требуется найти (хотя бы приближенно)  $x(t_1)$ ,  $x(t_2)$ , ...,  $x(t_n)$ , где  $x(t)$  — такое решение уравнения  $\dot{x} = f(t, x)$ , для которого  $x(t_0) = x_0$  \*).

Обозначим искомые  $x(t_1)$ , ...,  $x(t_n)$  через  $x_1$ , ...,  $x_n$ .

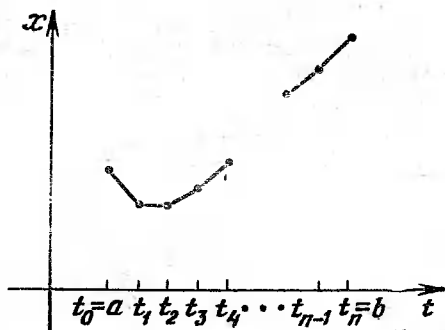


Рис. 98

Проведем ломаную Эйлера из данной точки  $(t_0, x_0)$  так, что  $t_0, t_1, \dots, t_n$  будут абсциссами вершин этой ломаной (рис. 98).

\*) Таким образом,  $x_0$  — это значение исследуемой величины в начальный момент времени. Равенство  $x(t_0) = x_0$ , где  $x_0$  — данное число, называют начальным условием.

Тогда значения ординат вершин ломаной дадут искомые значения функции  $x(t)$ .

Пусть уже известны  $x_0, x_1, \dots, x_{i-1}$ ,  $0 < i < n$ , покажем, как вычислить  $x_i$ . Уравнением прямой, имеющей угловой коэффициент  $k$  и проходящей через точку  $(t_{i-1}, x_{i-1})$ , будет, как нетрудно проверить, уравнение

$$x = k(t - t_{i-1}) + x_{i-1}.$$

При построении ломаной Эйлера берется угловой коэффициент, равный  $f(t_{i-1}, x_{i-1})$ , поэтому уравнение переписывается так:

$$x = f(t_{i-1}, x_{i-1}) \cdot (t - t_{i-1}) + x_{i-1}.$$

Для получения  $x_i$  подставим в это уравнение  $t = t_i$ ; так как  $t_i - t_{i-1} = h$ , получаем

$$x_i = f(t_{i-1}, x_{i-1})h + x_{i-1} \quad (i = 1, 2, \dots, n).$$

Это соотношение и определяет алгоритм Эйлера: зная  $x_0$ , мы вычисляем  $x_1$ , затем вычисляем  $x_2$  и т. д.

Заметим, что выведенную формулу для  $x_i$  можно получить и не прибегая к геометрическим рассуждениям. Для этого в вытекающем из данного дифференциального уравнения равенстве

$$\dot{x}(t_{i-1}) = f(t_{i-1}, x_{i-1}) \quad (i = 1, \dots, n),$$

достаточно заменить величину  $\dot{x}(t_{i-1})$  ее приближенным значением

$$\frac{x(t_{i-1} + h) - x(t_{i-1})}{h} = \frac{x_i - x_{i-1}}{h}; \text{ в результате получим}$$

$$\frac{x_i - x_{i-1}}{h} = f(t_{i-1}, x_{i-1}) \quad (i = 1, \dots, n),$$

или

$$x_i = f(t_{i-1}, x_{i-1})h + x_{i-1} \quad (i = 1, \dots, n).$$

Теперь мы можем заняться составлением программы. В программе можно обойтись без массива  $x_0, x_1, \dots, x_n$  — каждое значение функции может быть выведено сразу же после его вычисления. Имеет смысл описать в программе функцию, определяющую правую часть уравнения:

```

program Эйлер(input, output);
  var a, b, h, t, x: real; n, i: integer;
  function f(t, x: real): real
    begin ... end;
  begin read(a, b, n, x);
    h := (b - a)/n; t := a;
    for i := 1 to n do
      begin x := f(t, x)*h + x;
        t := t + h; writeln(x)
      end
    end.

```

Разумеется, в описании функции надо вместо многоточия подставить конкретные операторы, задающие вычисление  $f(t, x)$ . Например, при рассмотрении уравнения  $\dot{x} = t - x$  достаточно будет одного оператора  $f := t - x$ .

Довольно часто в условии задачи задается не  $n$ , а величина шага  $h$  и требуется вычислить значение  $x(t)$  для всех значений  $t$  вида  $a, a+h, a+2h, \dots$ , принадлежащих отрезку  $[a, b]$ . Оператор цикла и в этом случае написать несложно.

Кроме дифференциальных уравнений первого порядка на практике довольно часто возникают системы таких уравнений. Рассмотрим, например, систему

$$\begin{aligned}\dot{x}(t) &= F(t, x(t), y(t)), \\ \dot{y}(t) &= G(t, x(t), y(t)),\end{aligned}$$

где  $F$  и  $G$  — известные функции трех переменных  $t, x$  и  $y$ . Системы такого вида встречаются, в частности, при изучении численности двух конкурирующих популяций в ситуации «хищник — жертва» (лисы и зайцы, щуки и караси и т. д.). Для решения таких систем может быть применен алгоритм, являющийся несложным обобщением алгоритма Эйлера. В вытекающих из приведенной системы равенствах

$$\begin{aligned}\dot{x}(t_{i-1}) &= F(t_{i-1}, x_{i-1}, y_{i-1}), \\ \dot{y}(t_{i-1}) &= G(t_{i-1}, x_{i-1}, y_{i-1}) \quad (i=1, \dots, n)\end{aligned}$$

заменим  $\dot{x}(t_{i-1})$  и  $\dot{y}(t_{i-1})$  их приближенными значениями  $(x_i - x_{i-1})/h$  и  $(y_i - y_{i-1})/h$ . Это даст нам

$$\begin{aligned}x_i &= F(t_{i-1}, x_{i-1}, y_{i-1})h + x_{i-1}, \\ y_i &= G(t_{i-1}, x_{i-1}, y_{i-1})h + y_{i-1} \quad (i=1, \dots, n).\end{aligned}$$

Если первоначально заданы  $x_0, y_0$  (т. е. заданы начальные условия  $x(t_0) = x_0$  и  $y(t_0) = y_0$ ), то можно вычислить  $x_1, y_1$ , затем  $x_2, y_2$  и т. д.

Оператор цикла для последовательного вычисления  $x_1, y_1, \dots, x_n, y_n$  может быть написан, например, так:

```
for i:=1 to n do
  begin
    xx:=F(t, x, y)*h+x;
    yy:=G(t, x, y)*h+y;
    x:=xx; writeln(x, y)
  end
```

Вспомним, что первым примером дифференциального уравнения, который мы рассматривали в этом параграфе, было уравнение второго порядка

$$\ddot{x}(t) = \frac{1}{m} F,$$

где указывалось, что  $F$  может зависеть от некоторых величин из числа  $t, x(t), \dot{x}(t)$ . Скажем несколько слов о решении уравнений второго порядка.

Для того, чтобы получить численное решение уравнения, имеющего вид

$$\ddot{x}(t) = H(t, x(t), \dot{x}(t))$$

перейдем от этого уравнения к системе двух уравнений первого порядка. Введем новую неизвестную функцию  $y(t) = \dot{x}(t)$ ; это нам позволит написать систему

$$\dot{x}(t) = y(t),$$

$$\dot{y}(t) = H(t, x(t), y(t))$$

относительно  $x(t)$  и  $y(t)$ . Если первоначально заданы  $x_0$  и  $y_0$ , (т. е. если заданы начальные условия  $x(t_0) = x_0$  и  $\dot{x}(t_0) = y_0$ ), то  $x_1, y_1, x_2, y_2, \dots, x_n, y_n$  можно будет найти, в соответствии со сказанным выше о решении систем первого порядка, по формулам

$$x_i = y_{i-1} \cdot h + x_{i-1},$$

$$y_i = H(t_{i-1}, x_{i-1}, y_{i-1}) \cdot h + y_{i-1} \quad (i = 1, 2, \dots, n).$$

Итак, для того, чтобы найти на отрезке  $[a, b]$  некоторое решение уравнения второго порядка, нам потребуется кроме  $x(a)$  знать еще  $\dot{x}(a)$ . Начальные условия в этой задаче состоят из двух равенств. Это не выглядит неожиданным в связи с уравнением, которым записывается второй закон Ньютона: разумеется, для того, чтобы найти  $x(b)$  — координату точки в момент времени  $t = b$ , мало знать силу и начальную координату  $x(a)$ , надо знать еще начальную скорость  $\dot{x}(a)$  этой точки.

## ЗАДАЧИ

499. Переписать программу Эйлера, предполагая, что вместо натурального  $n$  задается величина шага — действительное число  $h$ .

500. В чем смысл введения дополнительной переменной  $x$  в приведенном фрагменте программы решения системы двух дифференциальных уравнений первого порядка?

501. С помощью алгоритма Эйлера найти решение дифференциального уравнения  $\dot{x} = f(t, x)$ . После уравнения записаны исходные данные:  $x_0$ , отрезок, на котором решается уравнение, шаг  $h$ :

а)  $\dot{x} = t + x, x_0 = 0, [0, 1.4], h = 0.02$ ;

б)  $\dot{x} = t^2 + x^2, x_0 = -2, [-1, 2], h = 0.1$ ;

в)  $\dot{x} = 2tx, x_0 = 1, [0, 1], h = 0.025$ ;

г)  $\dot{x} = \cos x, x_0 = 0, [1, 2], h = 0.1$ .

502. Дифференциальные уравнения часто применяются для описания процессов изменения численности популяций. Если зайцы

обитают на большом пространстве, их не поедают хищники и корма им хватает, то прирост оказывается пропорциональным количеству зайцев в популяции. Изменение числа зайцев  $x(t)$  описывается уравнением

$$\dot{x} = kx \quad (k > 0),$$

(число особей  $x$  и время  $t$  измеряются в подходящих для этого исследования единицах, например: зайцы исчисляются в сотнях, время — в годах). Если же пространство обитания не велико, то конкуренция из-за пищи вызывает уменьшение скорости прироста; процесс описывается уже более сложным уравнением:

$$\dot{x} = (a - bx)x \quad (a > 0, \quad b > 0)$$

(вывод уравнения мы опускаем). Положим в обоих случаях, для простоты, коэффициенты  $k$ ,  $a$ ,  $b$  равными 1. Это приведет к уравнениям  $\dot{x} = x$  и  $\dot{x} = (1 - x)x$ . Отметим, что в обоих случаях правая часть уравнения задается функцией, зависящей только от  $x$ .

а) Найти с помощью алгоритма Эйлера решение первого уравнения, беря  $x(0) = 0.25$ ,  $0 \leq t \leq 2$ ,  $n = 0.1$ . (Для этого уравнения известно точное решение. При  $x(0) = 0.25$  этим решением будет  $x(t) = 0.25e^t$ . Сравнить значения функции  $0.25e^t$  с результатами вычислений.)

б) Найти с помощью алгоритма Эйлера два решения второго уравнения, беря в первый раз  $x(0) = 0.25$ , во второй —  $x(0) = 2$ ; в обоих случаях  $0 \leq t \leq 3$ ,  $h = 0.1$ . Построить графики полученных решений. (Из этих графиков должно быть видно, что численность популяции стремится к некоторому положению равновесия. Положение равновесия определяется имеющимся количеством корма).

**503.** В 20-х годах нашего столетия для описания явлений межвидовой конкуренции. Примером является ситуация «хищник — жертва» — изменение численности популяций животных — хищников и животных — жертв (скажем, лис и зайцев). Уменьшение числа лис приводит к увеличению числа зайцев, но увеличение числа зайцев вызывает рост числа лис — возникают колебания численности той и другой популяции. Этот процесс описывается системой уравнений Лотки — Вольтерра

$$\begin{aligned}\dot{x} &= -ax + bxy, \\ \dot{y} &= cx - dxy,\end{aligned}$$

где  $x$  — число животных-хищников,  $y$  — число животных-жертв,  $a$ ,  $b$ ,  $c$ ,  $d$  — положительные постоянные. (Правые части уравнений не зависят от  $t$ .) Положим, для простоты  $a = b = c = d = 1$ . Найти с помощью алгоритма Эйлера решение системы Лотки — Вольтерра, беря  $x(0) = 0.5$ ,  $y(0) = 1$ ,  $0 \leq t \leq 24$ ,  $h = 0.3$ . Построить графики изменения популяций.

504. Для уравнения  $x=f(t, x)$  требуется указать точки плоскости, в каждой из которых проходящая через нее интегральная кривая имеет горизонтальное направление (точнее, горизонтальное направление имеет касательная к кривой). Рассмотреть уравнения, перечисленные в задаче 501.

505. Найти с помощью алгоритма Эйлера решение дифференциального уравнения

$$\ddot{x} = t\dot{x} + \frac{t^2 + x^2}{2},$$

удовлетворяющее начальным условиям  $x(0)=0$ ,  $\dot{x}(0)=1$ ,  $0 \leq t \leq 1$ ,  $h=0.2$ .

506. Вернемся к уравнению, описывающему изменение температуры тела в зависимости от температуры воздуха. Пусть для рассматриваемого тела  $\alpha=0.12$ :

$$\dot{x}(t) = 0.12(u(t) - x(t)).$$

Пусть в момент  $t=0$  температура воздуха была равна  $30^\circ$ , а впоследствии опускалась на  $0.01^\circ$  за каждую секунду. Таким образом, при  $t \geq 0$  выполнено  $u(t) = 30 - 0.01t$ . Требуется выяснить, сколько потребуется времени для того, чтобы тело, имевшее в момент времени  $t=0$  температуру  $100^\circ$ , остыло до температуры  $90^\circ$ . (Построить по алгоритму Эйлера решение уравнения с начальным условием  $x(0)=100$ , вычисляя значения неизвестной функции  $x(t)$  до тех пор, пока не получится значение, меньшее или равное  $90$ .)

507. Вновь рассмотрим уравнение изменения температуры тела и вновь, как и в предыдущей задаче, будем считать, что  $\alpha=0.12$ ; будем, далее, считать, что при  $t \geq 0$  выполнено  $u(t) = 30 - 0.1t + 5 \sin t$ . Таким образом, мы имеем дело с уравнением

$$\dot{x}(t) = 0.12(30 - 0.1t + 5 \sin t - x(t)).$$

Пусть  $x(0)=100$  и пусть нас не интересуют значения  $x(t)$  при  $0 < t < 5$ , а интересует только  $x(5)$ . Написать вариант программы Эйлера для решения этой задачи, предприняв некоторые меры по контролю точности результата. Первоначально вычисления проводятся с шагом  $h=1$ , затем они повторяются с шагом  $h=1/2$ , затем с шагом  $h=1/4$  и т. д. до тех пор, пока модуль разности двух последних приближений  $x(5)$  не окажется меньшим, чем данное число  $\varepsilon$ . Последнее приближение  $x(5)$  объявляется результатом.

508. Уравнение маятника (рис. 99)  $\ddot{x} = -\frac{g}{l} \sin x$  в случае малых

колебаний обычно заменяют более простым уравнением  $\ddot{x} = -\frac{g}{l} x$  (при малых по модулю  $x$  имеет место приближенное равенство  $x \approx \sin x$ ). Для последнего уравнения при  $x(0)=x_0$ ,  $\dot{x}(0)=0$  реше-



нием будет  $x(t) = x_0 \cos \sqrt{g/l} t$ . Для достаточно больших  $x_0$  (когда колебания нельзя считать малыми) функция  $x_0 \cos \sqrt{g/l} t$  не дает правильного описания колебаний. В предположении, что  $g/l=1$ , решить с помощью алгоритма Эйлера уравнение маятника, т. е. уравнение  $\ddot{x} = -\sin x$  для  $x(0) = \pi/2$ ,  $\dot{x}(0) = 0$ ,  $0 \leq t \leq 2\pi$ ,  $h=0.1$  и сравнить результаты со значениями функции  $\cos t$ .



509. В качестве уравнения маятника (см. предыдущую задачу) с трением рассмотрим

$$\ddot{x} = -\sin x - k\dot{x}$$

Рис. 99 ( $k$ —коэффициент трения). Решить по алгоритму Эйлера выписанное уравнение, полагая  $x(0) = \pi/2$ ,  $\dot{x}(0) = 0$ ,  $0 \leq t \leq 2\pi$ ,  $h=0.1$  для  $k=1.5$  и  $k=2.5$ . Сравнить результаты с решением предыдущей задачи.

510. Согласно формуле Ньютона—Лейбница  $\int_a^b f(t) dt = F(b) - F(a)$ , где  $F(t)$ —любая функция такая, что  $\dot{F}(t) = f(t)$ . Последнее равенство можно рассматривать как дифференциальное уравнение относительно неизвестной функции  $F(t)$ . Если взять в качестве начального условия  $F(a) = 0$ , то для определяемого этим условием решения будет по формуле Ньютона—Лейбница выполнено  $\int_a^b f(t) dt = F(b)$ . Можно приближенно вычислить  $F(b)$  с помощью алгоритма Эйлера и получить значение интеграла. Показать, что это использование алгоритма Эйлера в точности совпадает с алгоритмом левых прямоугольников приближенного интегрирования функций (см. задачу 171).

511. Из сказанного в предыдущей задаче можно сделать вывод, что алгоритм Эйлера является обобщением алгоритма левых прямоугольников. Обобщением алгоритма средних прямоугольников является следующий вариант алгоритма Эйлера—алгоритм добавочного полушага. Пусть выбрана величина шага  $h$ . Для исходной точки  $(t_0, x_0)$  находим  $f(t_0, x_0)$ —угловой коэффициент касательной к интегральной кривой. Откладываем в этом направлении отрезок до точки с абсциссой  $t_0 + \frac{h}{2}$  (добавочный полушаг). Вычисляем в этой точке угловой коэффициент касательной. Откладываем в этом направлении от точки  $(t_0, x_0)$  отрезок до точки с абсциссой  $t_0 + h$ . Это даст точку  $(t_0 + h, x_1)$ . Далее продолжаем построение этим же способом: добавочный полушаг, а затем—шаг, приводящий в точку  $(t_0 + 2h, x_2)$  и т. д. Значения  $x_1, x_2, \dots$ —приближенные значения неизвестной функции. Этот алгоритм в широком

классе случаев дает более точное решение, чем обычный алгоритм Эйлера с шагом  $\frac{h}{2}$ .

Написать формулы, выражающие  $x_i$  через  $x_{i-1}$ . Составить программу и испытать ее на уравнении  $\dot{x}=x$  с начальным условием  $x(0)=1$ . Сравнить результаты с теми, которые получаются с помощью обычного алгоритма Эйлера (принять во внимание, что точным решением уравнения  $\dot{x}=x$  с начальным условием  $x(0)=1$  является  $x(t)=e^t$ ).

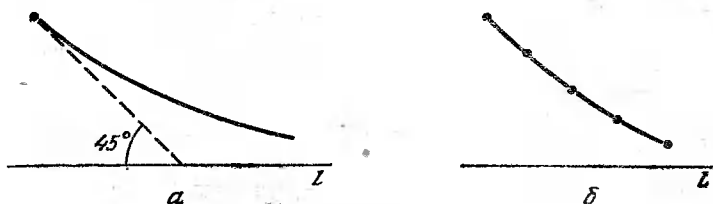


Рис. 100

512. В точке  $P$  находится собака, а в точке  $Q$  — заяц. Расстояние от  $P$  до  $Q$  равно 100 м (рис. 100а). Заяц бежит вдоль прямой  $l$  с постоянной скоростью 5 м/с. Собака бежит все время в направлении зайца со скоростью 10 м/с. Найти траекторию собаки, соответствующую первым 10 с погони. Для приближенного решения предлагается заменить кривую ломаной линией. Считается, что в первую секунду заяц пробегает отрезок  $QQ'$ , собака —  $PP'$ , во вторую секунду заяц пробегает отрезок  $Q'Q''$  и т. д., собака принимает решение о направлении погони в начале каждой секунды (рис. 100б).

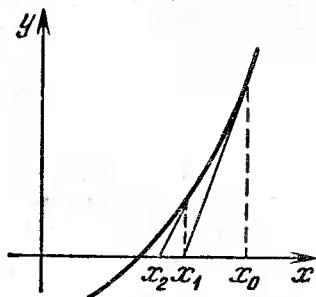


Рис. 101

513. Замена части графика некоторой функции касательной используется не только при решении дифференциальных уравнений, но и при нахождении корней алгебраических и трансцендентных уравнений вида  $f(x)=0$ , где  $f(x)$  — гладкая функция переменной  $x$ . Пусть  $v$  — истинное значение корня, т. е.  $v$  таково, что  $f(v)=0$ , и пусть  $x_0$  — приближение к этому корню. Если провести через точку  $(x_0, f(x_0))$  касательную к графику функции  $f(x)$ , то точка пересечения  $x_1$  этой касательной с осью абсцисс будет в широком классе случаев расположена ближе к  $v$ , чем  $x_0$  (рис. 101).

Уравнением касательной будет  $y = f'(x_0)(x - x_0) + f(x_0)$  \*); приравняв правую часть нулю, получаем, что  $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$ . Процесс улучшения приближений можно продолжить, воспользовавшись формулой  $x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$  ( $i = 1, 2, \dots$ ). Этот алгоритм улучшения приближений к корню называется алгоритмом касательных или алгоритмом Ньютона. Закон-

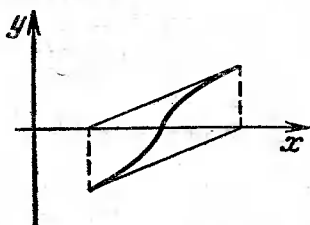


Рис. 102

чить процесс построения  $x_1, x_2, \dots$  можно в тот момент, когда для выбранного заранее малого числа  $\varepsilon$  будет выполнено  $|f(x_i)| < \varepsilon$  или когда  $|x_i - x_{i-1}| < \varepsilon$ .

Этот алгоритм не всегда дает сходящуюся к корню последовательность (рис. 102). Но геометрически очевидно, что если (как

на рис. 101) график функции  $f(x)$  сохраняет выпуклость вниз для всех  $x$ , лежащих между  $x$  и  $x_0$ , и при этом  $f(x_0) > 0$ , то последовательность  $x_0, x_1, \dots$  будет быстро сходиться (также как и в случае, когда сохраняется выпуклость вверх и при этом  $f(x_0) < 0$ ).

Дано действительное  $\varepsilon > 0$ . Требуется найти корень уравнения по алгоритму Ньютона, начиная построение последовательности с указанного приближения и заканчивая его в тот момент, когда  $|x_i - x_{i-1}| < \varepsilon$ , где  $x_i$  — последнее из вычисленных приближений. Рядом с уравнением вида  $f(x) = 0$  выписано начальное приближение и  $f'(x)$ :

- а)  $x^3 - 2x^2 + x - 3 = 0$ ,  $x_0 = 2.2$ ,  $3x^2 - 4x + 1$ ;
- б)  $x^3 - 6x^2 + 20 = 0$ ,  $x_0 = 2.31$ ,  $3x^2 - 12x$ ;
- в)  $x^4 - 3x^2 + 75x - 10\,000 = 0$ ,  $x_0 = -11$ ,  $4x^3 - 6x + 75$ ;
- г)  $1.8x^4 - \sin 10x = 0$ ,  $x_0 = 0.22$ ,  $7.2x^3 - 10 \cos 10x$ ;
- д)  $\lg x - x = 0$ ,  $x_0 = 4.67$ ,  $\frac{1}{\cos^2 x} - 1$ .

514. Используя правило дифференцирования степенной функции, согласно которому, в частности,  $(x^n)' = nx^{n-1}$  для любого натурального  $n$ , убедиться, что описанный в тексте § 6 алгоритм извлечения квадратного корня и описанный в задаче 87 алгоритм извлечения корня степени  $n$  являются частными случаями алгоритма Ньютона решения уравнения  $f(x) = 0$  (см. предыдущую задачу), при этом в качестве  $f(x)$  выступают  $x^2 - u$  и  $x^n - u$  соответственно.

\*) В этой и следующей задачах рассматриваются функции не от  $t$  как в тексте, а от  $x$ . Соответственно рассматриваются производные по  $x$ .

515. В ходе отыскания приближенного решения уравнения  $f(x)=0$  по алгоритму Ньютона получить на экране соответствующую иллюстрацию (рис. 101). График функции, касательные и вертикальные отрезки окрашиваются в разные цвета.

### § 35. Некоторые задачи поиска

Рассмотрим некоторый массив  $a_1, \dots, a_n$ . Этот массив обладает  $2^n$  подмассивами: один из них — пустой подмассив, который не содержит ни одного элемента, а все остальные подмассивы — это наборы элементов, имеющие вид  $a_i, a_j, \dots, a_p$  ( $1 \leq i < j < \dots < p \leq n$ ). В некоторых случаях приходится решать задачу выбора из данного массива одного или нескольких подмассивов, удовлетворяющих определенному условию. Можно столкнуться с условием такого рода, что окажется трудным найти решение более искусное, чем полный перебор всех подмассивов, который сопровождается проверкой условия для каждого из них. Так обстоит дело, например, когда из данного массива целых чисел  $a_1, \dots, a_n$  нужно выбрать подмассив  $a_i, a_j, \dots, a_p$  такой, что  $a_i + a_j + \dots + a_p = z$ , где  $z$  — данное целое число.

В связи с перебором всех подмассивов данного массива  $a_1, \dots, a_n$  рассмотрим задачу перебора всех массивов  $b_1, \dots, b_n$  из нулей и единиц. На эти массивы можно смотреть как на указатели подмассивов массива  $a_1, \dots, a_n$ . Если, например,  $b_1 = 1$ , то  $a_1$  включается в подмассив, если же  $b_1 = 0$ , то не включается. Пусть  $n=6$ , тогда 0, 1, 0, 1, 1, 0 указывает на подмассив  $a_2, a_4, a_5$ . На подмассив  $a_1, a_2, a_4, a_6$  будет указывать 1, 1, 0, 1, 0, 1. На пустой подмассив будет указывать 0, 0, 0, 0, 0, 0, а на подмассив, совпадающий с исходным массивом будет, в свою очередь, указывать 1, 1, 1, 1, 1, 1.

Общее количество  $n$ -элементных массивов из нулей и единиц равно  $2^n$ . В дальнейшем мы укажем алгоритм, который позволяет перебрать все  $2^n$   $n$ -элементных массивов из нулей и единиц в таком порядке, который можно охарактеризовать как словарный. Например, для  $n=3$  порядка будет таким:

0, 0, 0  
0, 0, 1  
0, 1, 0  
0, 1, 1  
1, 0, 0  
1, 0, 1  
1, 1, 0  
1, 1, 1

(если рассматривать массивы как слова в алфавите из двух цифр 0 и 1, считая, что 0 предшествует 1, то именно в таком порядке

массивы попадут в словарь). Но прежде, чем непосредственно заниматься этим алгоритмом, заметим, что иногда оказывается возможным некоторое сокращение перебора (значение  $2^n$  при увеличении  $n$  растет слишком быстро для того чтобы считать решение, основанное на рассмотрении всех возможных вариантов, приемлемым). Чтобы определить  $n$ -элементный массив из нулей и единиц, который выделяет подмассив массива  $a_1, \dots, a_n$ , нужно про каждый элемент  $a_1, \dots, a_n$  решить, принимается он в подмассив или нет. Может возникнуть следующая ситуация: относительно элементов  $a_1, \dots, a_k$  ( $k < n$ ), приняты какие-то решения, после этого обнаружилось, что как бы мы ни распоряжались остальными элементами, нам все равно не удастся получить подмассив, удовлетворяющий поставленному условию. В этом случае можно исключить из рассмотрения все подмассивы, первые элементы которых выбраны из числа  $a_1, \dots, a_k$  в соответствии с принятыми решениями. Например, дан массив целых положительных чисел  $a_1, \dots, a_n$  и целое положительное  $z$ . Требуется построить подмассив  $a_i, a_j, \dots, a_p$  ( $1 \leq i < j < \dots < p \leq n$ ) такой, что  $a_i + a_j + \dots + a_p = z$ . Если уже по каким-то соображениям выбраны начальные элементы  $a_i, a_j, \dots, a_r$  этого подмассива и сумма этих элементов больше  $z$ , то попытка подобрать к уже выбранным элементам еще несколько из  $a_{r+1}, a_{r+2}, \dots, a_n$  так, чтобы общая сумма была равна  $z$ , заведомо обречена на неудачу.

Вновь перейдем от подмассивов массива  $a_1, \dots, a_n$  к  $n$ -элементным массивам из нулей и единиц. Пусть среди всех  $n$ -элементных массивов из нулей и единиц требуется отыскать удовлетворяющие фиксированному условию. Пусть условие таково, что некоторые более короткие массивы из нулей и единиц являются тупиками — они не могут быть превращены в искомые массивы дописыванием нескольких элементов. Пусть имеется алгоритм, который позволяет распознавать тупики, т. е. имеется алгоритм, который может быть применен к любому массиву из нулей и единиц, содержащему менее  $n$  элементов и который обладает тем свойством, что результатом применения всегда является значение 1 или 0 («да» или «нет»), и значение 1 гарантирует, что массив является тупиком. Требуется использовать имеющийся алгоритм для возможно большего сокращения перебора. Решение этой задачи опирается на словарный порядок массивов. Обычно в словарях встречаются слова разной длины, и наряду с выписанной выше последовательностью трехэлементных массивов можно рассмотреть последовательность

0  
0, 0  
0, 0, 0

0, 0, 1  
 0, 1  
 0, 1, 0  
 0, 1, 1  
 1  
 1, 0  
 1, 0, 0  
 1, 0, 1  
 1, 1  
 1, 1, 0  
 1, 1, 1

Короткие массивы (т. е. содержащие менее  $n$  элементов) тоже интересуют нас, так как именно к ним следует применять алгоритм распознавания тупиков.

Словарный порядок полезен для нас тем, что массивы, начинающиеся с одной и той же группы элементов, идут подряд. Поэтому, если какая-то группа элементов признана тупиком, то из последовательности массивов могут быть исключены несколько подряд идущих массивов. Если какой-то из коротких массивов не оказался тупиком, то к нему приписывается 0. Так, если 1, 0 — не тупик, то дальше рассматривается 1, 0, 0. Если зашли в тупик или столкнулись с массивом длины  $n$ , то дальше надо заняться массивом, который:

- 1) не является удлинением рассмотренного массива;
- 2) расположен в словаре дальше, чем рассмотренный массив;
- 3) из всех массивов, удовлетворяющих 1) и 2) встречается в словаре первым.

Такой массив, если он существует, может быть построен из  $b_1, \dots, b_k$  просмотром элементов в обратном порядке, т. е. в порядке  $b_k, b_{k-1}, \dots$ , до первого  $b_m$ , равного 0; элементу  $b_m$  присваивается значение 1, и после этого присваивания группа элементов  $b_1, \dots, b_m$  удовлетворяет условиям 1)–3). Если  $b_1 = b_2 = \dots = b_k = 1$ , то перебор заканчивается. Например, если 1, 0, 1 — тупик, то надо перейти к 1, 1. От тупика 1, 1 переход к следующему массиву невозможен; перебор прекращается. Пусть  $n=3$  и известны два тупика:

0, 0  
 1

Тогда предложенный принцип перебора диктует последовательное рассмотрение массивов:

0  
 0, 0  
 0, 1

0, 1, 0  
0, 1, 1  
1

Отсеиваются только заведомо невыгодные варианты; нужные  $n$ -элементные массивы следует искать среди двух массивов: 0, 1, 0 и 0, 1, 1.

Уточним стратегию поведения в ситуации, когда попался тупиковый массив. Пусть тупиком является начальная часть  $b_1, \dots, b_k$  массива  $b_1, \dots, b_n$ . В соответствии со сказанным выше можно предложить следующую процедуру:

```

procedure B
  label 0;
  begin
    while  $b[k]=1$  do
      begin  $k:=k-1$ ;
      if  $k=0$  then go to 0
      end;
     $b[k]:=1$ ;
    0: end;
  
```

В результате обращения к этой процедуре без параметров могут быть изменены значения переменной  $k$  и некоторых элементов массива  $b_1, \dots, b_k$ . После обращения к процедуре *B* группа элементов  $b_1, \dots, b_k$  является той новой группой, которую следует рассмотреть за обнаруженным ранее тупиком. Значения элементов  $b_{k+1}, \dots, b_n$  не представляют интереса. Если тупиковая группа — это 1, 1, ..., 1, то следующую группу построить нельзя, сигналом об этом будет равенство  $k=0$ .

Итак, наряду с добавлением новых цифр при продвижении от начала к концу, время от времени возникает обратное проследивание цифр. Английское слово *бектрекинг* (backtracking — обратное проследивание) дало название алгоритму перебора в словарном порядке некоторых слов или массивов, при котором не рассматриваются слова или массивы, являющиеся удлиннениями замеченных тупиков.

Перебор массивов с помощью бектрекинга может производиться, в зависимости от цели перебора, либо до исчерпания всех массивов, либо до возникновения первого массива, удовлетворяющего поставленному условию.

Задача поиска в массиве целых положительных чисел  $a_1, \dots, a_n$  какого-нибудь подмассива с суммой элементов, равной целому положительному числу  $z$ , позволяет прекратить перебор с появлением первого такого подмассива. Если бы требовалось найти все такие подмассивы, или подсчитать их количество, то перебор продолжался бы до исчерпания всех возможностей.

Напишем последовательность операторов, представляющую общий алгоритм поиска среди  $n$ -элементных массивов из нулей и единиц такого, который обладает некоторым фиксированным свойством. Будем предполагать, что в нашем распоряжении имеется процедура без параметров  $T$ , с помощью которой исследуются  $k$  первых элементов ( $k \leq n$ ) массива  $b_1, \dots, b_n$ . В результате обращения к  $T$  переменная  $t$  получает значение 1, если  $k < n$  и  $b_1, \dots, b_k$  — тупик или если  $k = n$  и  $b_1, \dots, b_n$  не обладает нужным свойством; в остальных случаях  $t$  получает значение 0. Кроме процедуры  $T$  мы будем еще использовать указанную ранее процедуру без параметров  $B$  (обратное прослеживание) и процедуру без параметров  $A$ , которую мы пока (как и процедуру  $T$ ) оставим неопределенной, но про которую будем считать, что она задает необходимые действия в ситуации, когда найден  $n$ -элементный массив из нулей и единиц, для которого процедура  $T$  дает  $t = 0$ :

```

 $k := 1; b[1] := 0; T;$ 
while  $k > 0$  do
  if  $t = 1$  then begin  $B; T$  end
  else
    if  $k = n$  then  $A$ 
    else
      begin
         $k := k + 1; b[k] := 0; T$ 
      end

```

Выполнение оператора цикла завершается, когда значение переменной  $k$  оказывается равным 0, т. е. когда все  $n$ -элементные массивы из нулей и единиц уже обследованы. Но выполнение этого оператора может оборваться и раньше, если процедура  $A$  содержит внутри себя переходы к меткам, расположенным вне оператора цикла.

Используя бектрекинг, напишем целиком программу нахождения такого подмассива данного массива положительных целых чисел  $a_1, \dots, a_n$ , сумма элементов которого равна данному положительному целому  $z$ . Процедуру  $T$ , о которой шла речь выше, можно описать, например, так:

```

procedure  $T;$ 
  var  $i, s$ : integer;
  begin  $s := 0;$ 
    for  $i := 1$  to  $k$  do if  $b[i] = 1$  then  $s := s + a[i];$ 
    if  $(s > z)$  or  $((k = n) \text{ and } (s < z))$  then  $t := 1$ 
    else  $t := 0$ 
  end;

```

Однако при таком решении слишком много вычислений сумм элементов подмассивов массива  $a_1, \dots, a_n$ . Полезно включить в про-



грамму целочисленный массив  $s_0, s_1, \dots, s_n$  такой, что  $s_i$  ( $1 \leq i \leq k$ ) представляет собой сумму тех элементов  $a_j$  из числа  $a_1, \dots, a_i$ , для которых  $b_j = 1$ . Назначение  $s_0$  — вспомогательное. Описание процедуры  $T$  существенно упростится:

```

procedure T;
  begin
    if ( $s[k] > z$ ) or (( $k = n$ ) and ( $s[k] < > z$ )) then  $t := 1$ 
                                     else  $t := 0$ 
  end

```

Мы можем теперь написать всю программу целиком (считаем, что  $n = 10$ ):

```

program выбор(input, output);
  label 1;
  const  $n = 10$ ;
  type  $u = \text{array}[1..n]$  of integer;
        $v = \text{array}[0..n]$  of integer;
  var  $a, b, u; s: v; k, t, i, z: \text{integer}$ ;
  procedure B;
    label 0;
    begin while  $b[k] = 1$  do
      begin  $k := k - 1$ ;
        if  $k = 0$  then goto 0
      end;
    0: end;
  procedure T;
    begin
      if ( $s[k] > z$ ) or (( $k = n$ ) and ( $s[k] < > z$ ))
        then  $t := 1$  else  $t := 0$ 
    end;
  procedure A;
    var  $i: \text{integer}$ ;
    begin
      for  $i := 1$  to  $n$  do
        if  $b[i] = 1$  then writeln( $a[i]$ );
      goto 1
    end;
  begin read( $z$ );
    for  $i := 1$  to  $n$  do read( $a[i]$ );
     $k := 1; b[1] := 0; s[0] := 0; T$ ;
    while  $k > 0$  do
      begin
        if  $b[k] = 1$  then  $s[k] := s[k - 1] + a[k]$ 
          else  $s[k] := s[k - 1]$ ;

```

```

    if  $t = 1$  then begin  $B$ ;  $T$  end
    else
        if  $k = n$  then  $A$ 
        else
            begin  $k := k + 1$ ;
                 $b[k] := 0$ ;  $T$ 
            end
        end;
1: write('поиск завершен')
end.

```

Если поиск оказался успешным, то будут выведены элементы соответствующего подмассива, а затем — текст *поиск завершен*. Иначе будет выведен только указанный текст.

Обратимся теперь к задаче, которая была сформулирована в самом начале параграфа — тоже к задаче выбора подмассива с заданной суммой элементов, но среди элементов исходного массива допускаются не только положительные, но и отрицательные числа. Изменение программы выбор сводится к такому изменению описания процедуры  $T$ , при котором никакие короткие массивы не объявлялись тупиками. Новое описание будет иметь вид

```

procedure  $T$ ;
begin if  $(k = n)$  and  $(s[k] < > z)$ 
    then  $t := 1$ 
    else  $t := 0$ 
end;

```

Получившаяся программа предписывает перебор всех подмассивов данного массива  $a_1, \dots, a_n$ . В этой программе предполагается, что сумма элементов пустого массива равна нулю. Если не считать это предположение правомерным, то можно начать перебор всех  $n$ -элементных массивов из нулей и единиц с массива  $0, 0, \dots, 0, 1$ . Для этого достаточно заменить операторы

```
 $k := 0$ ;  $t := 0$ ;  $s[0] := 0$ ;  $T$ 
```

входящие в программу сразу после операторов, задающих ввод данных, на операторы

```

 $k := n$ ;  $s[0] := 0$ ;
for  $i := 1$  to  $n - 1$  do begin  $b[i] := 0$ ;  $s[i] := 0$  end;
 $b[n] := 1$ ;  $T$ 

```

В результате этой замены пустой подмассив в ходе выполнения программы рассматриваться не будет.

Если потребовать, чтобы перебор продолжался не до первого подходящего подмассива, а до исчерпания всех подмассивов, удовлетворяющих поставленному условию, то достаточно будет изме-

нить процедуру  $A$ , которая, как говорилось, задает необходимые действия в ситуации, когда найден  $n$ -элементный массив из нулей и единиц, для которого процедура  $T$  дает  $t=0$ . Применительно к двум рассмотренным вариантам программы *выбор* это применение может заключаться в том, что после вывода элементов соответствующего подмассива вместо оператора *goto 1* будет выполняться оператор  $t := 1$  — такое присваивание инициирует поиск следующего подмассива:

```

procedure  $A$ ;
  var  $i$ : integer;
  begin
    for  $i := 1$  to  $n$  do
      if  $b[i] = 1$  then writeln( $a[i]$ );
      writeln('-----');
     $t := 1$ 
  end;

```

Вывод ряда минусов предпринимается для того, чтобы отделить подмассивы друг от друга.

При выборе подмассива данного массива  $a_1, \dots, a_n$  мы относительно каждого элемента массива принимаем решение — брать этот элемент или нет. Два возможных исхода этого решения мы обозначали 1 и 0. Встречаются задачи, которые, как и задача выбора подмассива, требуют многократного принятия решений, но возможных исходов каждого решения оказывается больше двух.

Рассмотрим задачу о восьми ферзях. На обычной шахматной доске надо расставить восемь ферзей так, чтобы они не угрожали друг другу. Если ферзи расставлены и они не угрожают друг другу, то в каждой вертикали находится ровно один ферзь. Для того, чтобы задать некоторое такое расположение, надо для каждой вертикали решить, на каком именно из ее полей помещается ферзь. Априори каждое такое решение может иметь восемь исходов.

Удовлетворяет ли некоторое расположение ферзей условию задачи, имеет смысл проверять последовательным рассмотрением вертикалей. Для каждой вертикали нужно выяснить, не угрожает ли ее ферзь какому-нибудь из ферзей в предыдущих вертикалях. Как только обнаруживается, что ферзь в  $i$ -й вертикали угрожает какому-нибудь из ферзей в вертикалях  $1, 2, \dots, i-1$ , проверку можно закончить и дать отрицательный ответ. Если дошли до восьмой вертикали и не обнаружили угрожающих друг другу ферзей, то дается положительный ответ.

Основой для применения бектрекинга является наличие легкообнаруживаемых тупиков: коль скоро замечено, что ферзь в  $i$ -й вертикали угрожает ферзю в какой-то вертикали с меньшим номером, то можно не рассматривать все расположения, которые

предполагают эту же самую расстановку ферзей в вертикалях 1, 2, ...,  $i$ .

Если уже установлены не угрожающие друг другу ферзи в первых  $i-1$  вертикалях и при добавлении ферзя в  $i$ -ю вертикаль на нижнюю (первую) горизонталь выяснено, что новый ферзь угрожает какому-то из предыдущих, то продвигаем ферзя в  $i$ -й вертикали на одно поле вверх и вновь проверяем, получилось ли допустимое расположение. Если нет—передвигаем ферзя еще на одно поле вверх и т. д. до последней горизонтали. Пусть для этого ферзя не нашлось ни одного подходящего места. Тогда удаляем этого ферзя и начинаем подыскивать новое место для ферзя из  $i-1$ -й вертикали, передвигая его вверх. Если места для него не нашлось, то снимаем и его и переходим к вертикали с номером  $i-2$  и т. д.

Ферзи добавляются по возрастанию номера вертикали, а когда дело заходит в тупик, вертикали рассматриваются обратным прослеживанием. Таким образом, эта задача действительно решается бектрекингом.

По аналогии с задачей о ферзях можно рассмотреть задачу о ладьях. Разумеется, найти какое-нибудь одно расположение не угрожающих друг другу ладей не составляет труда—можно взять, например, расположение 1, 2, 3, 4, 5, 6, 7, 8 (записываем номера горизонталей, в которых располагаются ладьи в первой, второй, ..., восьмой вертикалях—см. рис. 103). Но наша задача будет состоять в том, чтобы найти все расположения.

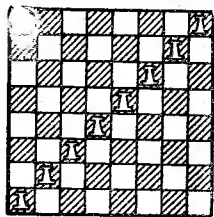


Рис. 103

Бектрекинг позволяет сделать это—найти очередное расположение и зарегистрировав его, можно продолжать тот же самый процесс дальше—попытаться подвинуть вверх ладью в последней вертикали, а если это невозможно, то снять эту ладью с доски и заняться ладьей в предпоследней вертикали и т. д. Так, после нескольких проб, вслед за расположением, которое проведено на рис. 103, будут получены расположения, приведенные на рис. 104а—в, которые можно записать как

1, 2, 3, 4, 5, 6, 8, 7

1, 2, 3, 4, 5, 7, 6, 8

1, 2, 3, 4, 5, 7, 8, 6

Поясним, например, переход от расположения, приведенного на рис. 103 к расположению, приведенному на рис. 104а. Ладью, расположенную в последней вертикали, невозможно подвинуть вверх, так как она занимает самое верхнее поле, поэтому снимаем эту

ладью и продвигаем вверх ладью в предпоследней вертикали (рис. 105). Затем устанавливаем ладью на нижнюю горизонталь

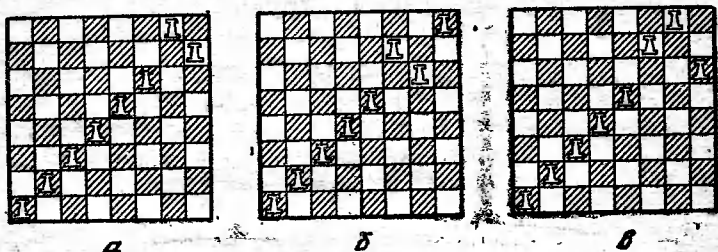


Рис. 104

последней вертикали, и, двигая ее вверх, находим для нее подходящее место в седьмой горизонтали. Получаем расположение, приведенное на рис. 104а.

Рассматриваемая нами задача о ладьях является, по существу, известной задачей о перестановках, общая формулировка которой такова: расположить в ряд всеми возможными способами  $n$  различных предметов. Известно, что таких расположений существует  $n! = 1 \cdot 2 \cdot \dots \cdot n$ . В нашем случае надо расположить в ряд числа 1, 2, 3, 4, 5, 6, 7, 8.

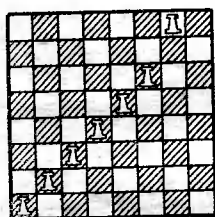


Рис. 105

Каждое расположение в ряд чисел 1, 2, ..., ...,  $n$  называется перестановкой  $n$ -й степени. Задача о получении всех перестановок  $n$ -й степени является важной для обширного ряда комбинаторных задач, решаемых на компьютерах. Задачу о ладьях можно для при-

дания алгоритму большей общности рассматривать на доске  $n \times n$ , считая, что ладей  $n$  штук. Записывая каждое расположение ладей в виде  $i_1, i_2, \dots, i_n$ , где  $i_1, i_2, \dots, i_n$  — различные целые числа от 1 до  $n$ , мы получим в процессе решения задачи о  $n$  ладьях все перестановки  $n$ -й степени.

Можно взять за образец операторы, выписанные на стр. 248, но при этом необходимо заменить присваивание  $b[1] := 0$  (расположенное до оператора цикла) и присваивание  $b[k] := 0$  (расположенное в теле оператора цикла) соответственно присваиваниями  $b[1] := 1$  и  $b[k] := 1$ , так как нижняя горизонталь имеет у нас номер 1:

```

k := 1; b[1] := 1; T;
while k > 0 do
  if t = 1 then begin B; T end
  else
    if k = n then A

```

```

else
begin
 $k := k + 1$ ;  $b[k] := 1$ ;  $T$ 
end

```

Процедуры  $B$  и  $T$  тоже нуждаются в изменениях. Эти процедуры применяются к  $b_1, \dots, b_k$  в таких случаях, когда  $b_1, \dots, b_{k-1}$  задают расположение не угрожающих друг другу ладей в первых  $k-1$  вертикалях. Это позволяет довольно просто описать обе процедуры:

```

procedure B;
begin
 $k := k - 1$ ;
if  $k > 0$  then  $b[k] := b[k] + 1$ 
end;
procedure T;
label 0;
var i: integer;
begin t := 0;
for i :=  $k - 1$  downto 1 do
if  $b[k] = b[i]$  then
begin
 $t := 1$ ; goto 0
end;
0: end;

```

Выполнение процедуры  $A$  будет заключаться в выводе полученной перестановки  $n$ -й степени и в присваивании  $t := 1$ . Получаем программу (считаем, что  $n = 8$ ):

```

program перестановки(output);
const n = 8;
type u = array[1..n] of integer;
var b: u; k, t, i: integer;
procedure B;
begin  $k := k - 1$ ;
if  $k > 0$  then  $b[k] := b[k] + 1$ 
end;
procedure T;
label 0;
var i: integer;
begin t := 0;
for i :=  $k - 1$  downto 1 do
if  $b[k] = b[i]$  then
begin t := 1; goto 0 end;
0: end;

```

```

procedure A;
  var i: integer;
  begin
    for i:=1 to n do write(b[i]);
    writeln(' '); t:=1
  end;
  begin k:=1; b[1]:=1; T;
    while k > 0 do
      if t=1 then begin B; T end
      else
        if k=n then A
        else
          begin
            k:=k+1; b[k]:=1; T
          end
        end
      end
    end.

```

Программа не содержит операторов ввода, и поэтому в ее заголовке не упоминается входной файл *input*.

С помощью этой программы перестановки получаются в словарном порядке (мы считаем, что 1 предшествует 2, 2 предшествует 3 и т. д.). Первой будет выведена перестановка 1, 2, 3, 4, 5, 6, 7, 8, последней — 8, 7, 6, 5, 4, 3, 2, 1. Можно также сказать, например, что перестановка 6, 3, 2, 5, 7, 8, 1, 4 будет выведена раньше, чем перестановка 6, 3, 2, 7, 1, 8, 4, 5. Всего будет выведено  $8! = 40\,320$  перестановок. Если  $n=3$ , то перестановок будет  $3! = 6$ , если  $n=4$ , то будет получено  $4! = 24$  перестановки и т. д.

### ЗАДАЧИ

516. Изменить программу *выбор* так, чтобы:

- а) вместо последовательности элементов выбранного подмассива получалась последовательность номеров (индексов) этих элементов;
- б) текст *поиск завершен* выводился только в том случае, когда не существует подмассива с требуемым свойством.

517. Пусть в процессе выполнения программы *выбор* группа нулей и единиц 0, 1, 1, 1, 1 признана тупиковой. Какая следующая группа должна быть рассмотрена?

518. Изменить программу *перестановки* так, чтобы выводились перестановки чисел 0, 1, ...,  $n-1$  (считать, что нижняя горизонталь на шахматной доске имеет номер 0, затем идет горизонталь с номером 1 и т. д.).

519. Показать, что в процедуре *A*, используемой в программе *перестановки*, можно после присваивания  $t:=1$  выполнить присваивание  $k:=n-1$ , что приведет к некоторому сокращению действий.

520. Пусть в программе *перестановки* значение константы  $n$  равно 6. Какая из двух перестановок будет выведена раньше: 5, 2, 3, 6, 4, 1 или 5, 2, 1, 4, 6, 3?

521. Можно ли обращение к процедуре  $T$ , непосредственно предшествующее оператору цикла *while*  $k > 0$  *do* ..., заменить оператором присваивания  $t := 0$ :

а) в программе *выбор*;

б) в программе *перестановки*?

522. Получить все  $n$ -элементные последовательности из нулей и единиц, содержащие ровно  $m$  единиц ( $m \leq n$ ) (эти последовательности тесно связаны с сочетаниями из  $n$  элементов по  $m$  элементов в каждом —  $m$ -элементными подмножествами данного  $n$ -элементного множества; число этих последовательностей  $n$ , одновременно, подмножеств равно  $\frac{n!}{m!(n-m)!}$ ).

523. Получить все размещения из  $n$  элементов 1, 2, ...,  $n$  по  $m$  элементов в каждом ( $m \leq n$ ). Каждое такое размещение — это  $m$  элементов множества  $\{1, 2, \dots, n\}$ , расположенные каким-то образом в ряд. Число сочетаний равно  $n(n-1)\dots(n-m+1)$ . Эта задача может рассматриваться как задача о  $m$  неугрожающих друг другу ладьях на прямоугольной шахматной доске, имеющей  $n$  горизонталей и  $m$  вертикалей.

524. Изменить программу *выбор* так, чтобы выбирался подмассив, сумма элементов которого равна  $z$  и который имеет

а) наименьшее

б) наибольшее

число элементов из всех подмассивов с такой суммой. Если этими условиями массив определяется не однозначно, то взять любой из массивов, удовлетворяющих этим условиям.

525. Изменить программу *выбор* так, чтобы вычислялось количество всех подмассивов с данной суммой элементов.

526. Даны действительные  $a_1, \dots, a_n$  ( $n$  — некоторая константа). Найти наименьшее из всех значений вида  $\sin(a_i a_j \dots a_p)$ , где  $1 \leq i < j < \dots < p \leq n$ .

527. Даны действительные  $a_1, \dots, a_n$  ( $n$  — некоторая константа). Ни одно из чисел  $a_1, \dots, a_n$  не равно нулю. Найти наименьшее по модулю из всех значений вида  $a_i + a_j + \dots + a_p$ , где  $1 \leq i < j < \dots < p \leq n$ .

Потребуется ли производить перебор подмассивов, если нужно получить подмассив с наибольшей по модулю суммой элементов?

528. Получить все полные перестановки элементов 1, 2, ...,  $n$  (перестановка  $a_1, \dots, a_n$  элементов 1, ...,  $n$  называется полной, если  $a_i \neq i$  для  $i = 1, \dots, n$ ; например, перестановка 3, 1, 2 элементов 1, 2, 3 является полной, а перестановка 2, 1, 3 — нет).



529. Подробно рассмотреть задачу о  $n$  ферзях. Представляет интерес как программа, позволяющая получить одно-единственное расположение, так и программа, позволяющая получить все расположения ферзей.

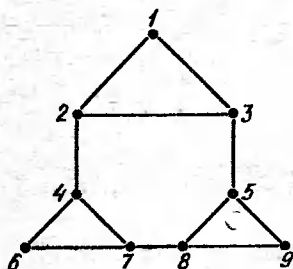


Рис. 106

530. Графом называется совокупность точек (вершин), некоторые из которых соединены между собой линиями. Графу, содержащему  $n$  вершин, можно сопоставить матрицу соединений размера  $n \times n$ . Элемент  $a_{ij}$  матрицы соединений равен единице, если  $i$ -я вершина соединена с  $j$ -й (ясно, что  $a_{ij} = a_{ji}$ ). Так, графу, изображенному на рис. 106, сопоставляется матрица соединений

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Дана матрица соединений графа, содержащего  $n$  вершин ( $n$  — некоторая константа). Требуется выяснить, существует ли такой замкнутый путь, проходящий по линиям графа, который проходит через все вершины по одному разу (так называемый гамильтонов цикл). Если такой путь существует, то построить его.

*Указание.* Перестановка  $a_1, \dots, a_n$  чисел  $1, \dots, n$  задает гамильтонов цикл, если вершина с номером  $a_i$  соединена линией с вершиной  $a_{i+1}$  для  $i=1, 2, \dots, n-1$  и, кроме этого, вершина  $a_n$  соединена с  $a_1$ . В сравнении с процессом построения перестановок здесь возникают дополнительные тупики.

531. Имеется  $m$  различных предметов. Известен вес каждого предмета и его стоимость. Определить, какие предметы надо положить в рюкзак, чтобы общий вес не превышал заданной границы, а общая стоимость была максимальной. Решить задачу для  $m$  предметов, веса которых в килограммах равны  $p_1, \dots, p_m$ , стоимость  $c_1, \dots, c_m$ . Вес рюкзака не должен превышать 50 кг.

532. Имеется  $n$  предметов, веса которых равны  $a_1, \dots, a_n$ . Разделить эти предметы на две группы так, чтобы общие веса двух групп были максимально близки.