

Л.М.ФИНК

ПАПА, МАМА

9

И МИКРО-
КАРДКУЯТОР

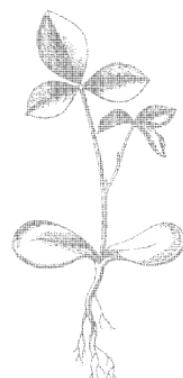
БИБЛИОТЕЧНАЯ СЕРИЯ

Л.М.ФИНК

ПАПА, МАМА
и МИКРО-
КАЛЬКУЛЯТОР



Москва
«Радио и связь»
1988



ББК 32.973
Ф 59
УДК 681.321.0

Р е ц е н з е н т докт. техн. наук, проф. Я. К. Трохименко

Редакция литературы по вычислительной технике

Финк Л. М.

Ф 59 Папа, мама, я и микрокалькулятор. —
М.: Радио и связь, 1988. — 272 с.: ил.

ISBN 5-256-00150-7

Задача книги — научить составлять программы для
микрокалькуляторов «Электроника Б3-34», «Электроника
МК-54» и «Электроника МК-56». В популярной форме опи-
саны методы и приемы программирования. Приведены при-
меры алгоритмов и программ решения задач различной
сложности.

Для широкого круга читателей.

Ф 2405000000-117
046(01)-88 КБ-27-20-87

ББК 32.973

ISBN 5-256-00150-7 © Издательство «Радио и связь», 1988

ПРЕДИСЛОВИЕ

В наши дни электронно-вычислительная техника стремительно проникает во все новые сферы человеческой деятельности. Овладение вычислительной техникой, навыками программирования становится необходимым людям самых разных специальностей и всех возрастов. Помочь в овладении этими навыками — цель книги. Задача книги — научить составлять программы для наиболее доступных вычислительных устройств — программируемых микрокалькуляторов «Электроника Б3-34», «Электроника МК-54» и «Электроника МК-56». В отличие от большей части книг, посвященных программируемым микрокалькуляторам, эта книга не является сборником прикладных программ и в еще меньшей степени сборником программ развлекательных игр. Наряду с примерами хороших программ здесь приводится немало программ, не совсем хороших и даже явно неудачных, с соответствующей критикой. Автор убежден, что научить искусству программирования только на положительных примерах невозможно. Преподнести эту критику в доходчивой форме помог принятый здесь беллетристированный стиль изложения.

Автор настоятельно рекомендует желающим научиться программировать читать эту книгу, имея под рукой микрокалькулятор и проделывая на нем все описываемые действия.

Большая часть программируемых здесь задач по своему содержанию не выходит за рамки программы средней школы. В тех немногих местах, где этот принцип нарушался, даны популярные пояснения, в частности, таких понятий, как предел, ряд и т. д. Автор надеется, что книга представит интерес для широкого круга читателей, желающих научиться работать с программируемым микрокалькулятором.

1

КАК ЭТО НАЧАЛОСЬ

Началось все очень просто. Родители в день моего рождения подарили мне микрокалькулятор «Электроника Б3-34». При этом папа сказал, что сейчас время вычислительной техники, что необходимо ликвидировать «компьютерную неграмотность» и что верным путем к этому является овладение программируемым микрокалькулятором и на это потребуется не больше месяца. Мама, которая всегда соглашается с папой, но не сразу, а после некоторого спора, сказала, что программируемый микрокалькулятор, конечно, вещь полезная, но всего лишь игрушка и компьютерной грамоте он не научит. Для этого нужно учить алгоритмические языки, например Фортран или хотя бы Бейсик, и что этому будут учить в школе — введен новый предмет «Информатика». Папа возмутился и сказал, что только программируемый микрокалькулятор позволил ему выработать верный подход к вычислительной технике, а всякие алгоритмические языки — это лишь способ переложить на машину часть чисто технической работы при составлении программы и лично он пока обходится без Фортрана. Важно, чтобы Миша научился понимать, что такое алгоритм и как нужно программировать. А Бейсики и Алголы он легко выучит, когда они ему понадобятся. (Миша, как Вы понимаете, это я.)

Чтобы поскорее прекратить их спор, я спросил:

— А чем отличается программируемый микрокалькулятор от обычного, непрограммируемого?

Тут они оба начали мне объяснять то, что я уже знал. Программируемый микрокалькулятор (далее я буду писать сокращенно ПМК) позволяет ввести в его память программу решения той или иной задачи и исходные данные, после чего решение производится автоматически и нужные результаты высвечиваются в определенном порядке.

— По существу, — сказал папа, — это карманная электронная вычислительная машина (ЭВМ). Она выпол-

няет почти все те же функции и имеет те же параметры, что и ЭВМ 50-х годов, занимавшая много десятков квадратных метров площади и потреблявшая десятки киловатт электропроизводства. Да и от современной ЭВМ он отстает не на много.

— Ну нет, — возразила мама, — от микрокалькулятора до большой ЭВМ — дистанция огромного размера. ЭВМ считает в несколько сотен раз быстрее, имеет значительно большую память, может использовать, если нужно, внешнюю память, а также память других ЭВМ и данные, вводимые из других устройств (периферийных). Большая ЭВМ может и выдавать результаты в виде распечаток на бумаге, а если нужно, и в виде графиков.

Вы, конечно, догадались, что мои родители имеют какое-то отношение к вычислительной технике, особенно мама. Она работает программистом и в разных алгоритмических языках, как говорится, собаку съела. Папа занимается радиоэлектроникой, работает в одном НИИ научным сотрудником. Он очень любит ПМК и старается, по возможности, решать свои задачи на нем.

Я не очень прислушивался к спору родителей, а с интересом рассматривал новый микрокалькулятор, стараясь понять назначение клавиш. Должен сказать, что я не впервые встретился с микрокалькулятором. Еще с 5-го класса у меня был очень хороший микрокалькулятор «Электроника Б3-18М», но непрограммируемый. Папа шутя называл его электронной логарифмической линейкой, хотя на самом деле он позволял производить более сложные вычисления, чем линейка, и со значительно большей точностью. Я пользовался им при решении всех задач по математике и физике, очень привык к нему. Но он ведь непрограммируемый. А сейчас я смогу сам программировать, отлаживать программы, в общем, делать все те таинственные операции, которыми так гордится мама.

Я не заметил, что родители уже давно перестали спорить о месте ПМК в ликвидации компьютерной неграмотности и обсуждают мои годовые отметки. Они были довольны — по физике и математике у меня были пятерки, да еще пара пятерок по другим предметам, которые меньше волнуют моих родителей. Но вот по русскому языку и литературе я маленько не дотянул — получил 4.

— Вот видишь, — сказала мама, — ты говоришь, что он легко овладеет Бейсиком и Алголем, когда они ему понадобятся, а он, оказывается, и своим родным русским языком еще не овладел как следует.

— Ну что ж, — ответил папа, — поможем ему и в этом. Вот что, Миша, начинай вести дневник. Записывай, как ты будешь знакомиться с ПМК и учиться программировать. Это поможет тебе добиться сразу двух целей — овладеть программированием и научиться излагать свои мысли. Кто знает, может быть, твой дневник поможет и другим научиться пользоваться ПМК.

Так это и началось. Теперь я каждый день записываю все, что мне удалось узнать о ПМК «Электроника Б3-34».

2

ЧТО МНЕ УДАЛОСЬ УЗНАТЬ СРАЗУ

Конечно, нужно было прочесть заводское руководство, приложенное к ПМК. Но это долго и нудно. Ведь этот микрокалькулятор у меня не первый. Клавиши здесь почти такие же, как и в моем старом Б3-18. Сначала надо попробовать, как он считает без программирования. Проверимка, чему равно дважды два. Для этого нажму клавишу 2, затем клавишу \times , снова 2 и, наконец, =, и тогда на индикаторе должно появиться 4.

Нажимаю 2 и вижу на индикаторе 2. Так и должно быть. Нажимаю клавишу \times , а на индикаторе почему-то появился 0. Я удивился, но продолжаю выполнять свой план. Снова нажимаю 2, и снова на индикаторе 2, что меня немного успокаивает. Теперь нужно нажать клавишу =. Ищу ее и не нахожу. Куда же делся знак равенства? Вероятно, его заменяет какая-то другая клавиша? Внимательно просматриваю все клавиши, но знака равенства не нахожу. Зато есть клавиши с такими-то непонятными обозначениями ШГ, В/О, С/П, К и многие другие. Все-таки придется прочесть заводское руководство. А может быть, лучше спросить у папы?

— Папа, а куда девался знак равенства?

— Его здесь нет, он не нужен, так как язык ПМК Б3-34 не такой, как у большинства непрограммируемых микрокалькуляторов.

— Как, папа, опять языки?

— Ну, конечно, если ты хочешь общаться с ПМК, то должен знать его язык. Не беспокойся, этот язык очень прост, ты его уже знаешь, только нужно научиться пра-

вильно его применять. А для этого прочти руководство. Ну, ладно, на первых порах я тебе немного помогу, но дальше рассчитывай на свои силы.

Итак, в каждом языке есть слова (лексика) и правила их соединения в предложения (синтаксис). «Словами» языка ПМК являются команды, подаваемые нажатием клавиш. Этот набор слов у одних ПМК богаче, у других беднее, но всегда содержит слова, отвечающие числам и основным математическим операциям над ними.

Посмотри на пульт твоего ПМК (рис. 2.1). На нем находятся индикатор, два переключателя и шесть рядов клавиш по пять в каждом ряду. Назначение индикатора и переключателей тебе ясно. Левый переключатель включает и выключает питание ПМК, правый устанавливает способ измерения углов — в градусах или радианах.

Рассмотрим для начала клавиши, расположенные в четырех нижних рядах. Они определяют основу лексики твоего ПМК. Для прямого набора чисел предназначены 11 черных клавиш: 10 цифр и запятая. С их помощью можно ввести любое положительное число, содержащее не более восьми цифр — от 0,0000001 до 99999999. Введенное число высвечивается на индикаторе в виде обычного десятичного числа или, как говорят, с *фиксированной запятой*.

Синяя клавиша $/-$ означает «Сменить знак». Если после ввода некоторого числа, например 243,456, нажать эту клавишу, то введенным окажется отрицательное число —243,456. Нажав клавишу $/-$ еще раз, снова получим число 243,456, положительное.

Синяя клавиша ВП означает «Ввод порядка», позволяет вводить числа в нормализованном виде или, как еще говорят, с *плавающей запятой*, т. е. в виде $\pm m \cdot 10^{\pm n}$, где первый сомножитель m — целое число или десятичная дробь, называемый мантиссой, а второй сомножитель 10 в некоторой степени, называемой порядком. На индикаторе твоего ПМК мантисса занимает восемь знакомест в левой части, а порядок — два знакоместа в правой части. Знак минус, если он нужен, стоит, как обычно, перед мантиссой или (и) перед показателем степени.

Ты, конечно, знаком с таким представлением. Оно очень удобно: позволяет расширить диапазон чисел, обрабатываемых в ПМК. Так, в твоем ПМК мантисса m может содержать до восьми знаков, а порядок n — до двух. Это позволяет вводить числа от $-9,999999 \cdot 10^{-9}$ до $9,999999 \cdot 10^{99}$, причем самое малое по абсолютному значению число, которое можно ввести в ПМК или которое

может различить ПМК в процессе вычисления, $1 \cdot 10^{-89}$. Таким образом, отношение самого большого по абсолютному значению числа к самому малому составляет почти 10^{199} . Это неимоверно большое число, значительно больше числа всех элементарных частиц во всей наблюдаемой части вселенной.

Тем не менее иногда при расчетах может возникнуть число, по абсолютному значению большее $9,9999999 \times$

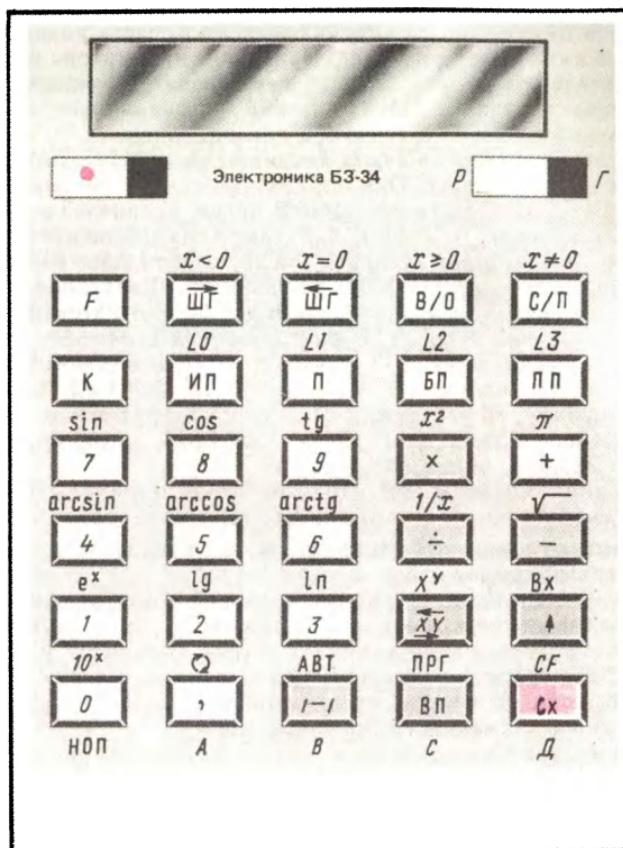


Рис. 2.1

$\times 10^{99}$. Такого числа ПМК не знает и выдает вместо него ЕГГОГ — от английского слова eggog — ошибка. В таких случаях говорят, что получилась «машинная бесконечность». Точно так же, если возникло число, меньшее по абсолютному значению $1 \cdot 10^{-99}$, то ПМК выдает вместо него 0. Это так называемый «машинный нуль». Впрочем, иногда вместо машинного нуля ПМК выдает сигнал ошибки (ЕГГОГ). Этот же сигнал ПМК выдаст всякий раз, когда ты попытаешься заставить его совершать не имеющие смысла или запрещенные в математике операции, например делить на нуль, вычислять логарифмы отрицательных чисел и т. д., а также некоторые операции, которым он не обучен, например возводить отрицательные числа в степень (кроме второй).

Возвращаясь к представлению чисел, хочу еще отметить, что вводить числа в ПМК можно в любой форме — с фиксированной или плавающей запятой, но ПМК выдаст результат в форме с фиксированной запятой, когда его абсолютное значение лежит между 1 и 10^8 . Если же результат меньше 1 или больше 10^8 , то он выдается в стандартной форме — с плавающей запятой, причем в мантиссе перед запятой всегда одна значащая (т. е. ненулевая) цифра. Ну как, все понятно?

— Нет, не все. Откуда взялись такие названия? Если я буду вводить числа, скажем, 2034,51; 8,63217; 32,132; 10000, в которых запятая стоит на самых разных местах — после четвертого знака, после первого, второго и даже вовсе отсутствует, то это называется фиксированной запятой, а если ПМК выдает результаты в виде $2,534 \times 10^{-7}$ или $3,111 \cdot 10^{10}$, где запятая всегда стоит после первого знака, то это называется плавающей запятой? Я бы называл совсем наоборот.

— Что ты, Миша, — возмутился папа. — Никогда не думал, что ты обращаешь внимание только на внешнюю сторону, а не на сущность. Ведь когда ты записал 2034,51, то определил некоторое число, в котором запятая находится на фиксированном месте, в данном случае после четвертого знака. Переместить ее никуда нельзя: изменится само число. А вот, если представить его в нормализованном виде (показательной форме), например $2,03451 \cdot 10^3$, то запятую можно перемещать куда угодно, изменения при этом, конечно, и показатель степени, например $20,3451 \times 10^2$, или $203.451 \cdot 10^1$, или $20345,1 \cdot 10^{-1}$, или даже $0,203451 \cdot 10^4$. Вот откуда название плавающей запятой. Теперь ясно?

— Ясно.

— Тогда пойдем дальше. Красная клавиша Сх служит для сброса только что введенного числа. Нажав ее, на индикаторе увидим 0. Четыре синие клавиши $+$, $-$, \times , \div соответствуют четырем арифметическим действиям.

И еще две синие клавиши $\overset{\leftarrow}{X}Y$ и \vec{Y} имеют непосредственное отношение к синтаксису языка ПМК Б3-34. Их назначение я объясню позже. А сейчас несколько слов хочу сказать об одной черной клавише в самом верхнем ряду F (от слова функция).

Это префиксная клавиша. Она сама по себе никаких операций не вызывает, но изменяет назначение нажатой после нее клавиши. Так, если ты нажмешь клавишу 7, то этим ты введешь в ПМК цифру 7. Если же нажмешь клавишу F, а затем 7, то никакой семерки ты никуда не введешь, а выдашь команду вычислить синус угла, значение которого указано на индикаторе. Вот смотри, я ввожу в индикатор число 15. При этом переключатель Р — Г (радианы — градусы) находится в положении Г. Затем нажимаю последовательно клавиши F и 7, на индикаторе число $2.5881903 \cdot 10^{-1}$, это и есть значение $\sin 15^\circ$. Над клавишей 7 красная надпись sin, которая и показывает, что после префиксной клавиши F эта клавиша вычисляет функцию синус от числа, высвечиваемого на индикаторе. Такие же красные надписи есть и над другими клавишами. Как видишь, они позволяют вычислять прямые и обратные тригонометрические функции от аргумента x , возводить x в квадрат, находить его обратную величину $1/x$, извлекать из него квадратный корень, вычислять показательные функции e^x и 10^x , а также десятичный (lg) и натуральный (ln) логарифмы от x . Кроме того, нажав на клавиши F +, мы можем вызвать на индикатор число $\pi = 3,1415926$. Остальные красные надписи ты освоишь, когда перейдешь к программированию.

Теперь обратимся к синтаксису. Прежде всего познакомимся с некоторыми терминами. Микрокалькулятор осуществляет определенные операции над числами. Эти числа называются *операндами*. Если операция осуществляется над одним операндом, то она называется *одноместной* (например, вычисление $\sin x$; x^2 ; \sqrt{x} ; $\ln x$), а если над двумя операндами (например, $x + y$; $x \div y$; x^y и т. д.), то *двухместной*. Из последовательности *операторов* — символов, определяющих операции, формируется програм-

ма решения той или иной задачи. В непрограммируемых микрокалькуляторах эти операторы вводятся вручную, нажатием соответствующих клавиш. В программируемых микрокалькуляторах операторы можно вводить автоматически, набрав соответствующую программу, которая «записывается» в программную память.

Но вернемся к работе ПМК в обычном режиме, без программирования. Операторы вводятся с помощью последовательности команд, а порядок ввода команд определяет синтаксис языка ПМК. В твоем старом микрокалькуляторе используется простая алгебраическая запись. Чтобы вычислить, например, чему равняется 2×5 , мы вводим операторы в том же порядке, в каком они записаны, т. е. сначала 2, затем \times , затем 5 и, наконец, оператор вывода, т. е. знак равенства $=$, на индикаторе читаем 10. Впрочем, иногда это правило нарушается: одноместные операторы вводятся после того операнда, к которому они относятся. Так, чтобы вычислить $2\sqrt{5}$, мы должны нажимать клавиши в таком порядке: $2 \downarrow 5 F \sqrt{-}$ и на индикаторе прочитаем 4,472136, или для вычисления $\sqrt{2} \lg 5$ нажимаем $2 F \sqrt{-} \times 5 F \lg =$ и получаем 0,9884928. Двухместные операторы вводятся так, как они записываются, например $1 - 2 + 3 - 4 + 5 =$, вычисляются нажатием клавиш в такой последовательности: $1 \downarrow - 2 \downarrow + 3 - 4 + 5 =$.

В твоем новом ПМК использован другой способ -- обратная запись. Это чуть-чуть сложнее запомнить, но зато очень сильно упрощает конструкцию ПМК. Двухместные операторы вводятся не между двумя operandами, к которым они относятся, а после них. Чтобы вычислить, чему равно 2×5 , мы нажимаем клавиши $2 \uparrow 5 \times$ и получаем ответ 10. В данном случае клавиша \uparrow является разделительным знаком. Если не нажать ее, то мы введем один operand 25, а не два операнда 2 и 5. Зато клавиша \downarrow здесь вовсе не нужна: после ввода двухместного оператора сразу получается результат. А чтобы вычислить $\sqrt{2} \lg 5$, мы нажимаем клавиши $2 F \sqrt{-} 5 F \lg \times$. Здесь никакого разделительного знака не нужно, потому что operandы 2 и 5 разделены одноместным оператором $\sqrt{-}$.

Усвоить эту систему очень просто, если учесть, что во всяком микрокалькуляторе для выполнения двухместных операций используется по крайней мере два операционных регистра, которые обычно обозначают X и Y.

Регистры используются для хранения одного числа в любой форме — и с фиксированной и с плавающей запятой. Таких регистров в любом микрокалькуляторе довольно много, а в программируемом — тем более. Но сейчас мы рассмотрим только регистры X и Y.

Когда число вводится, оно записывается в регистр X, который связан с индикатором, так что всегда можно видеть, что в этом регистре находится. Проще всего представлять себе регистр в виде строчки, в которой записано число. Над строчкой X находится строчка (регистр) Y, в которой тоже записано какое-то число. Записать число в регистр Y можно различными путями, в частности можно переписать число из регистра X, нажав клавишу \uparrow . Попробуем сложить или перемножить два многозначных числа, например 1284,56 и 312,8915. Как ты поступишь, если их нужно сложить на бумаге?

— Очень просто, я их запишу в столбик, одно под другим, и буду поразрядно складывать, соблюдая правило переноса.

— Вот точно так же действует и микрокалькулятор: он записывает их в регистры X и Y и складывает, а результат записывается в регистр X и поэтому может быть прочитан на индикаторе. Так осуществляются все двухместные операции, причем операнды располагаются всегда так, как и на бумаге при вычислении в столбик:

$$\begin{array}{r} 1284,56 \\ - 312,8915 \\ \hline 1597,4515 \end{array} \quad \begin{array}{r} 379,5 \\ - 211,27 \\ \hline 168,23 \end{array} \quad \begin{array}{r} 534,2 (Y) \\ \times \\ 16,1 (X) \\ \hline 8600,62 \end{array}$$

или при записи деления в виде дроби

$$1234,5 : 547,1 = \frac{1234,5 (Y)}{547,1 (X)} = 2,256443,$$

или при возведении в степень

$$X^Y = 25^{34} = 3,3881332 \cdot 10^{47}.$$

Как видишь, это легко запомнить.

— Да, это нетрудно. Но когда я делаю на бумаге, скажем, умножение, то оба сомножителя у меня сохра-

няются в записи. Здесь же в регистр X попадает произведение, так что сомножитель, который там был, пропадает. А что будет после выполнения операции в регистре Y?

— Это очень интересный и важный вопрос. Конструкторы ПМК решают его по-разному. Например, первый отечественный ПМК «Электроника Б3-21» устроен так, что операнд из регистра X после любой операции, одноместной или двухместной, не сохраняется, а заменяется результатом операции, а в регистре Y операнд, как правило, сохраняется, пока не будет заменен, например, оператором \uparrow . В нашем же ПМК Б3-34 по-другому. Здесь не два, а целых пять *операционных* регистров X, Y, Z, T и X1. Четыре из них связаны между собой в некоторую структуру, называемую *операционным стеком* *, а регистр X1 взаимодействует только с регистром X.

Регистр X1 является «хранителем истории»: в него после каждой операции записывается прежнее содержимое регистра X. И если нужно, можно вызвать эту «архивную» запись обратно в регистр X с помощью клавиши Bx (Возврат x), т. е. нажав префиксную клавишу F, а затем клавишу \uparrow , над которой имеется красная надпись Bx. А вот содержимое регистра Y после двухместной операции меняется содержимым регистра Z.

Взглянув на пульт микрокалькулятора, сразу можно определить, какой синтаксис он использует. Если это простая алгебраическая запись, то обязательно имеется клавиша вывода результата =, а очень часто и клавиши скобок. Если же это обратная запись, то ни знака =, ни скобок на клавиатуре нет, но обязательно имеется клавиша \uparrow и почти всегда $\overset{\leftarrow}{X} \vec{Y}$, хотя обозначение может быть и другое.

Чтобы грамотно обращаться с микрокалькулятором, а особенно чтобы составлять для него хорошие программы, нужно очень ясно представлять себе, как работает операционный стек. Но у меня уже больше нет времени, да и тебе нужно приучаться к самостоятельности. Возьми-ка руководство и почитай его. Пока не обращай внимания на программирование, а найди ответы на два вопроса: 1) как работает операционный стек и 2) какие возможности для запоминания данных (чисел) имеются в Б3-34.

Вот так я получил свое первое задание.

* Английское слово, означающее «куча, стог».

3

Я ВЫПОЛНЯЮ ПЕРВОЕ ЗАДАНИЕ

Я стал читать заводское руководство, и оказалось, что ничего сложного в нем нет. Я мог бы разобраться и сам, без помощи папы. Уже через час я подошел к папе и доложил о выполнении задания.

— В ПМК 14 регистров памяти. В каждый можно в любой момент записать любое число, находящееся в регистре Х, и в любой же другой момент переписать это число из регистра памяти в регистр Х (т. е. на индикатор). При этом в регистре памяти это число сохраняется до тех пор, пока в него не будет записано другое число. Регистры памяти имеют номера 0, 1, 2, 3 и т. д. до 9, а следующие четыре регистра обозначены буквами А, В, С, Д (написаны черным под клавишами , /—/ ВП и Сх). Почему-то А, В, С — латинские, а Д — русское (см. рис. 2.1).

Чтобы внести в регистр памяти какое-либо число, нужно ввести его в регистр Х (если его там нет), нажать клавишу П (память) и клавишу, соответствующую номеру выбранного регистра. Чтобы извлечь из регистра памяти хранящееся в нем число, нужно нажать клавишу ИП (из памяти) и клавишу с номером регистра, и тогда содержимое этого регистра окажется в регистре Х и выветится на индикаторе.

Таким образом, кроме префиксной клавиши F обнаружились еще две префиксные клавиши П и ИП. Есть еще и четвертая префиксная клавиша К, но я, откровенно говоря, не разобрался с ее назначением.

— Ладно, — сказал папа, — до клавиши К мы в свое время доберемся, а пока можем обойтись без нее.

— Итак, — продолжил я, — в нашем ПМК 19 регистров, из них 14 регистров памяти и 5 операционных.

— Ну, это ты уж много на себя берешь. Ты познакомился с 19 регистрами, в которые можно записывать операнды, но это вовсе не значит, что других регистров в ПМК нет. Их довольно много, но пользователь с ними непосредственно не общается. Они используются в процессоре, в который входит арифметическое устройство для выполнения микропрограмм, предназначенных для вычисления квадратов, квадратных корней, тригонометрических функций и других операторов, в частности вводимых с помощью префиксной клавиши F. Сами эти микропро-

граммы записаны раз навсегда в постоянном запоминающем устройстве (ПЗУ). Расскажи теперь, что ты узнал о работе операционного стека.

— Я очень долго разбирался в описании и рисунках, которые поясняют его работу, и в конце концов после нескольких экспериментов понял, что все это можно изложить в виде семи простых правил. Я их записал (для сокращения вместо слова «регистр» буду писать одну букву Р, а за ней буквы Х, Y, Z, Т).

1. Команда ↑ перемещает содержимое всех регистров операционного стека вверх на одну ступень, т. е. содержимое РХ переходит в РY (сохраняясь в РХ), содержимое РY — в РZ, а содержимое РZ — в РТ. То, что было перед этим в РТ, пропадает.

2. Если сразу после команд Сх или ↑ в РХ вводится число с помощью цифровых клавиш, то содержимое остальных операционных регистров не изменяется.

3. Если в РХ вводится число из регистра памяти (в том числе из РХ1) или с помощью цифровых клавиш, но не сразу после команд Сх или ↑, то старое содержимое РХ переходит в РY, содержимое РY — в РZ, содержимое РZ — в РТ, а содержимое РТ пропадает. Другими словами, содержимое всех регистров передвигается вверх.

4. После любой одноместной операции в РХ появляется результат этой операции, исходное число, бывшее в РХ, переходит в «архивный» регистр РХ1, а содержимое РY, РZ и РТ не изменяется.

5. После любой двухместной операции, в которой всегда участвуют операнды из РХ и РY, результат записывается в РХ, прежнее значение РХ переходит в «архивный» РХ1, содержимое РY пропадает и на его место переходит содержимое РZ, а в РZ переходит содержимое РТ, которое сохраняется в РТ. Другими словами, содержимое всех операционных регистров передвигается вниз на одну ступень, оставляя за собой след в РТ.

6. Команда $\overset{\leftarrow}{XY}$ меняет местами содержимое РХ и РY, в других регистрах ничего не меняется.

7. Есть еще одна команда, обозначенная красным кружком со стрелкой над клавишей с запятой. Она совершает кольцевую перестановку operandов во всем операционном стеке, т. е. содержимое РТ переходит в РZ, содержимое РZ — в РY, содержимое РY — в РХ, а содержимое РХ — в РТ. Для простоты буду обозначать эту команду стрелкой →.

Вот и все.

— Молодец. В общем ты довольно точно записал все правила обмена информацией, хранящейся в операционном стеке. Они тебе очень пригодятся при составлении программ. Ты только забыл отметить, что команды $\overset{\leftarrow}{X}Y$ и $\vec{Y}X$ вызывают также запись содержимого PX в $PX1$. Нужно еще заметить, что при операции X^Y стек ведет себя как при одноместной, а не как при двухместной операции, т. е. содержимое PY , PZ и PT сохраняется. Давай изобразим эти семь правил в виде схем. Это поможет тебе вспомнить, что происходит в том или другом случае. Тут папа взял листок бумаги и нарисовал такие схемки (рис. 3.1). Держи их при себе, пока не набьешь руку в программировании. Мне они не нужны: я хорошо все это помню. Вот ты и познакомился с входным языком твоего ПМК, по крайней мере с той его частью, которая нужна для непрограммируемых вычислений. Попробуем посчитать. Ты помнишь формулу, по которой определяется пройденный путь S при равноускоренном движении в течение времени t ? Она имеет такой вид

$$S = V_0 t + at^2/2, \quad (3.1)$$

где V_0 — начальная скорость; a — ускорение. Пусть, например, $V_0 = 3$ м/с, $a = 10$ м/с² и $t = 2$ мин. Вычисли пройденный путь.

— Ну, папа, для этого мне калькулятор и не нужен. Я легко подсчитаю это в уме. Трижды два будет 6, а 10, умноженное на 2² и деленное на 2, будет 20, так что всего 26.

— 26 чего?

— Метров, конечно.

— Считаешь в уме ты, конечно, здорово. В третьем классе ты бы, безусловно, получил пятерку. Но кроме прыти необходимо еще и внимание, и его никакой микрокалькулятор не заменит. Ты не заметил, что в размерностях скорости и ускорения у нас фигурировали секунды, а время t задано в минутах.

Я покраснел.

— Я не сомневаюсь, — продолжал папа, — что ты сможешь в уме перевести минуты в секунды и повторить свое вычисление тоже в уме. Но сейчас мы преследует другую цель — научиться обращаться с микрокалькулятором. Поэтому давай нажимай на клавиши и получай результат.

Ничего не поделаешь, приходится подчиниться. Я сначала нажал клавишу 3. Этим я ввел значение V_0 в РХ. Потом нажал клавишу \uparrow и тем самым перевел значение скорости V_0 в РY. Затем нажал 1, 2 и 0 и тем самым ввел t (в секундах) в РХ, а потом дошла очередь до оператора умножения, я нажал клавишу \times и увидел сразу на индикаторе 360. Теперь нужно вычислить второй член, а для этого нужно запомнить первый результат. Это можно свободно сделать в уме, но я решил показать, что хорошо освоился с микрокалькулятором, и ввел этот результат в регистр памяти с номером 0, нажав клавиши Π и 0. Дальше я решил, что сначала лучше ввести t и вычислить его квад-

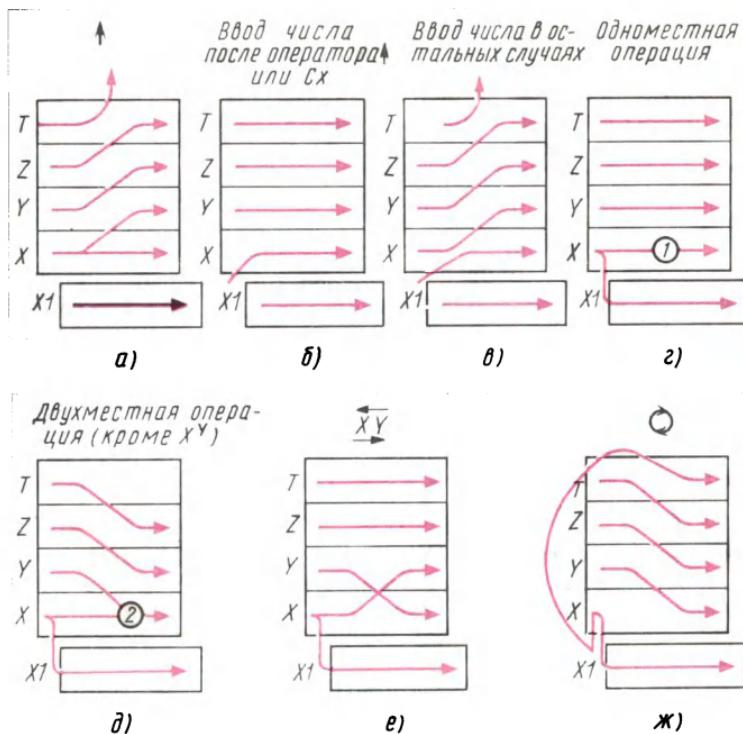


Рис. 3.1

рат. Набрал опять число 120, а потом нажал префиксную клавишу F и клавишу \times , над которой записан оператор x^2 . На индикаторе я с удовлетворением увидел $120^2 = 14\ 400$. Хотел было перевести это число в PY, нажав клавишу \uparrow , но вовремя спохватился, взглянув на только что нарисованные схемы. Ведь если я сейчас введу число a , то содержимое RX само перейдет в PY. Поэтому я набрал 10, нажал на клавишу \times и получил $at^2 = 144\ 000$. Далее ввел в RX число 2, в результате прежнее содержимое переместилось в PY, и я смог сразу произвести деление, нажав клавишу \div . К полученному значению второго члена 72 000 нужно прибавить первый член, который хранится в регистре памяти. Нажимаю клавиши ИП 0, вводя значение первого члена в RX, значение второго члена при этом перешло в PY. Наконец, нажав клавишу $+$, я сложил оба эти члена, получил результат 72 360 м и гордо посмотрел на папу.

— Что ж, для начала неплохо, но не очень экономно. Ты зря залезал в регистр памяти 0 и не использовал другие операционные регистры, кроме X и Y. Есть много способов более экономного решения этой задачи. Я бы, например, сделал так.

Набираю число $t = 120$ с, затем нажимаю клавишу \uparrow , в результате значение t оказывается в RX и PY. Затем нажимаю клавиши F и x^2 — в RX окажется t^2 , а в PY по-прежнему t . Далее ввожу $a = 10$ в RX. Теперь t^2 будет в PY, а t — в PZ. Нажимаю клавишу \times и получаю в RX произведение at^2 , а значение t опустилось из PZ в PY. Ввожу в RX число 2. Величина at^2 поднялась в PY, а t — в PZ. Нажимаю клавишу \div . Теперь at^2 разделилось на 2 и результат находится в RX, а значение t снова опустилось в PY. Нажимаю клавишу \overleftarrow{XY} , в результате $at^2/2$ переместилось в PY, а t — в RX. Ввожу в RX значение $V_0 = 3$, при этом t переместилось в PY, а $at^2/2$ — в PZ. Нажимаю клавишу \times . Теперь V_0t находится в RX, а $at^2/2$ — в PY. Остается нажать клавишу $+$, и я вижу на индикаторе результат.

— По-моему, это не на много проще моего способа, — сказал я. — Ты только не использовал регистр памяти, а операций у тебя все равно много.

— Это тебе только показалось. Я очень подробно рассказывал, что происходит в разных регистрах. А ты по-

пробуй подсчитай, сколько раз ты нажимал клавиши и сколько я.

Хотя я несколько раз сбивался в счете и все не хотел признать свое поражение, пришлось с папой согласиться. При моем способе пришлось нажимать клавиши 21 раз, а при папином — всего 15 раз.

— Вот видишь, я сэкономил почти 30 %. Теперь для решения этой задачи у нас есть две программы. Обе дают верное решение, но одна лучше другой, так как использует меньше операций.

— Какие программы? Разве это уже программы?

— Ну, конечно. Ведь программа для ЭВМ — это последовательность операций. Ты составил свою программу, а я — свою.

— И что же по этим программам ПМК может считать сам, без моего участия?

— Безусловно. Нужно только ввести ее в программную память. Но сегодня у меня нет времени. Вот тебе следующее задание. Изучи по руководству, как можно записать программу и как ввести ее в память ПМК, а завтра расскажешь и продемонстрируешь. Для этого тебе нужно будет обе программы ввести в ПМК. Но только давай не будем задаваться определенными значениями для начальной скорости, ускорения и времени. Будем исходить из предположения, что время в секундах записано в РД, ускорение a — в РА, а начальная скорость V_0 — в РВ. Ясно?

— Ясно.

И вот я с новым заданием.

4

Я СОСТАВЛЯЮ И ВВОЖУ СВОЮ ПЕРВУЮ ПРОГРАММУ

Я еще раз почитал руководство, и теперь все стало просто и ясно. ПМК может работать в двух режимах — режиме программирования и режиме автоматических вычислений. Для смены режимов используются клавиши $/—/$ и ВП, нажимаемые после префиксной клавиши F. Это сразу видно на пульте: над этими клавишами написано АВТ и ПРГ.

Чтобы ввести программу в память ПМК, нужно нажать клавиши F и ВП — на индикаторе в правой части высвешиваются два нуля. Это значит, что программная память готова принять программу, точнее, ее первый шаг, которому присвоен номер 00. Затем я должен нажимать те же клавиши, которые я нажимал, когда решал задачу вручную. При этом в правой части индикатора каждый раз номер ожидаемого шага увеличивается на единицу. Этот номер называется *адресом команды*. Всего таких адресов может быть 98. Больше в памяти ПМК места нет, и если пробовать вводить больше шагов, то они будут записываться на места ранее записанных начальных шагов, так что в общем получится ерунда.

Слева от номеров шагов на индикаторе высвечиваются двузначные номера — коды, соответствующие каждому оператору, введенному в программу. А как эти коды связаны с операторами? С вводом цифр, конечно, все просто: ввожу 1 и вижу 01, ввожу 2 и вижу 02, ввожу 9 и вижу 09. Но с операторами сложнее: я ввожу +, вижу 10, ввожу —, вижу 11. В некоторых случаях вместо цифр в код входят буквы. Вероятно, все это нужно будет запомнить.

Когда я закончил ввод последней операции программы, нужно ввести еще одну, самую последнюю — остановить ПМК. Для этого служит клавиша С/П — Стоп/Пуск. Ее нужно нажимать и для того, чтобы запустить вычисления по программе. Впрочем, перед запуском нужно еще нажать клавишу В/О — Возврат и очистка, чтобы калькулятор выполнял программу с самого начала, а не откуда придется.

Но прежде чем вводить программу, нужно ее записать на бумаге в виде таблицы, указав номера шагов (адреса команд), название вводимой операции, нажимаемые клавиши и код. Коды приведены в таблице, которая дана в руководстве. Нужно, конечно, записать, какие величины заранее должны быть введены в регистры памяти и где хранятся результаты вычисления.

Итак, я составляю первую свою программу. Вот она.

Программа 1. Вычисление пройденного пути S за время t при равноускоренном движении по формуле

$$S = V_0 t + at^2/2. \quad (4.1)$$

где V_0 — начальная скорость; a — ускорение (мой вариант).

Предварительный ввод: V_0 — в РВ, a — в РА, t — в РД, В/О С/П.

Адрес команды	Операция	Нажимаемые клавиши	Код
00	Ввод V_0 из РВ	ИП /—/	6L
01	Ввод t из РД	ИП Сх	6Г
02	Умножение	×	12
03	Запись промежуточного результата в Р0	П 0	40
04	Ввод t из РД	ИП Сх	6Г
05	Возведение в квадрат	F ×	22
06	Ввод a из РА	ИП ,	6—
07	Умножение	×	12
08	Ввод числа 2	2	02
09	Деление	÷	13
10	Вывод промежуточного результата из Р0	ИП 0	60
11	Сложение	+	10
12	Стоп	С/П	50

Я вводил V_0 и t из регистров памяти, а не с помощью цифровых клавиш. Это позволило обойтись без разделительной команды \uparrow .

Введя программу, я нажал клавиши F /—/ и перешел тем самым в режим АВТ. Затем ввел значения $V_0 = 3 \text{ м/с}$, $a = 10 \text{ м/с}^2$ и $t = 120 \text{ с}$ в соответствующие регистры памяти, нажал клавиши В/О и С/П, и ПМК заработал. На индикаторе что-то мелькало, и примерно через 5 с выступил результат 72360 м. Я очень обрадовался. Моя программа работает! А интересно, что будет через 3 мин, т. е. 180 с? Я ввел 180 в РД, снова запустил ПМК и через 5 с прочел результат 162540 м. Значит, за третью минуту пройден путь, больший, чем за первые две. Но это и не удивительно, ведь движение равноускоренное.

Интересно, будет ли папин вариант программы работать так же хорошо? Попробуем ввести его.

Программа 2. То же самое (папин вариант).

Предварительный ввод данных: V_0 — в РВ, a — в РА, t — в РХ.

Адрес команды	Операция	Нажимаемые клавиши	Код
00	Ввод t в РУ	\uparrow	0E
01	Возведение в квадрат	F X	22
02	Вывод a из РА	ИП А	6-
03	Умножение	X	12
04	Ввод числа 2	2	02
05	Деление	\div	13
06	Обмен между РХ и РУ	XY	14
07	Ввод V_0 из РВ	ИП /-/	6L
08	Умножение	X	12
09	Сложение	+	10
10	Стоп	C/P	50

Оказывается, папин вариант программы всего на два шага короче моего. Когда я ввел ее в ПМК и запустил его, то, конечно, получил тот же самый результат. Время вычисления оказалось таким же — 5 с (в пределах точности секундомера), ну, может, на полсекунды меньше.

Вечером я обо всем рассказал папе. Он довольно улыбнулся, но заметил, что на самом деле ПМК работает не в двух режимах, как это сказано в руководстве, а в трех: 1) обычного (непрограммированного или ручного) вычисления; 2) ввода программы, 3) автоматического вычисления по введенной программе. Оператор ПРГ переводит ПМК из первого режима во второй, а оператор АВТ — из второго режима в первый (а не в третий, как можно было бы подумать по его названию). Переход же в третий режим происходит из первого при нажатии клавиши С/П. Обратный переход из третьего режима в первый происходит автоматически после окончания автоматического вычисления по программе, или при наличии ошибки в программе, или перегрузке разрядной сетки ПМК («аварийный стоп»). Правда, иногда при неправильно составленной программе может произойти «зацикливание», т. е. ПМК продолжает автоматические вычисления и никак не хочет остановиться. Тогда приходится его останавливать искусственно, нажав клавишу С/П, а затем искать ошибку в программе.

Относительно моего сравнения двух вариантов программ папа сказал:

— Все-таки мой вариант немного лучше твоего. Во-первых, он на два шага короче, а бывают случаи, когда один шаг решает, можно ли задачу поручить микрокалькулятору или нужно выходить на большую ЭВМ. Во-вторых, я не занимаю лишний регистр памяти (нулевой). Но это мелочи. Набери, пожалуйста, любой из вариантов нашей программы. Попробуй ответить на такой вопрос. Как ты, вероятно, знаешь, Галилей, изучая законы падения тел, бросал разные вещи с Пизанской башни и обнаружил, что все они достигали земли через 3,4 с. Какова высота этой башни?

— Я не помню.

— Я знаю, что не помнишь. Но на этот вопрос может ответить микрокалькулятор, если ты ему поможешь.

— А, понимаю. Нужно вычислить пройденный путь для тела, движущегося равноускоренно под воздействием земного тяготения при нулевой начальной скорости. Ввожу ускорение силы тяжести, равное $a = 9,81 \text{ м/с}^2$, в РА, 0 в РВ и 3,4 с в РД. Запускаю микрокалькулятор и через 5 с читаю 56,7018. Высота Пизанской башни 56,7018 м.

— Да, примерно га... Конечно, за миллиметры и сантиметры здесь поручиться нельзя: само время t измерялось с небольшой точностью — секундомеров у Галилея не было и величина a введена с точностью порядка 1 %. Поэтому наш результат следует округлить: высота башни 56 ... 57 м. Посмотрим в энциклопедию. Действительно, высота Пизанской башни около 56 м.

Другой вопрос. Ты подбросил мяч вертикально вверх, сообщив ему начальную скорость 12 м/с. Будет ли он на земле через 2,5 с?

— Я оставляю то же значение a в РА, ввожу 12 в РВ и 2,5 в РД. Запускаю ПМК и читаю 60,657 ... Что-то странное. За 2,5 с мяч достигнет высоты больше 60 м? Значит, я его легко переброшу через Пизанскую башню? Нет, здесь что-то не так. Микрокалькулятор наврал.

— Конечно, здесь что-то не так, — подтвердил папа.

— Но подумай, может быть ошибся не ПМК, а ты?

Мне не пришлось долго думать. Я сразу сообразил, что ввел ускорение и начальную скорость в одном направлении. Это было бы верно, если бы я бросил мяч вниз где-то на большой высоте, сообщив ему начальную скорость 12 м/с. Тогда бы он действительно под действием начальной скорости и силы тяжести прошел бы более 60 м за 2,5 с. В нашем же случае, если я ввел $V_0 = 12 \text{ м/с}$, считая на-

правление вверх положительным, то ускорение $a = g$, направленное вниз, равно $-9,81 \text{ м/с}^2$ и ввести его в РА я должен со знаком минус. Я быстро внес исправление и тут же получил ответ $S = -6,5625 \cdot 10^{-1} = -0,65625 \text{ м}$.

— Что же это означает? — спросил папа.

— Это значит, что в рассматриваемый момент мяч окажется на 65 с небольшим сантиметров ниже той точки, с которой его подбросили, и, значит, почти достиг земли.

Смотри, папа, я могу с помощью этой программы проследить весь путь мяча, вводя разные значения t , и построить таблицу:

$t, \text{ с}$	0,2	0,4	0,6	0,8	1,0	1,2	1,4
$S, \text{ м}$	2,20	4,02	5,43	6,46	7,10	7,34	7,19
$t, \text{ с}$	1,6	1,8	2,0	2,2	2,4	2,6	.
$S, \text{ м}$	6,64	5,71	4,38	2,66	0,55	-1,96	

Цифры я, конечно, округлил.

— Правильно, Миша. Теперь я вижу, что ты прочувствовал, какие большие возможности дает ПМК, даже при решении такой простенькой задачи. По этой таблице можно построить график движения мяча — это будет парабола. Последнее число в твоей таблице $S = -1,96 \text{ м}$, конечно, не соответствует нашей задаче; мяч до момента 2,6 с уже упадет на землю и отскочит от нее. В нашей формулировке задачи (или, как теперь принято говорить, в нашей математической модели) земля как препятствие на пути мяча не учитывается.

Сейчас попрощаемся с этой программой и я дам тебе еще одно задание. Только что мы определили, что высота Пизанской башни примерно 56 м. Вероятно, Галилею было нелегко на нее взбираться. Лифтов тогда не было. Но вот он уже наверху и может приступить к своему опыту. Конечно, нужно отдохнуться. И, вероятно, в эти несколь-

ко минут отдыха он не только обдумывал свой эксперимент, но и рассматривал открывающийся с вершины башни вид. А какова дальность прямой видимости на такой высоте? Сравни ее с дальностью видимости, например, с Исаакиевского собора в Ленинграде, с Останкинской телевизионной башни в Москве, с Эйфелевой башни в Париже. И пусть тебе в этом поможет твой ПМК и, конечно, твои знания геометрии.

5

НАСЛЕДСТВО АЛЬ-ХОРЕЗМИ

На следующее утро я занялся задачей о дальности прямой видимости с высоты. Построил простенький чертеж (рис. 5.1), где R — радиус земного шара; h — высота наблюдателя над поверхностью земли; d — длина касательной от наблюдателя к поверхности земли. Мы ищем длину дуги r — радиус прямой видимости. Так как в нашей задаче $h \ll R$, длина дуги r (дальность по земле) практически не отличается от длины касательной d (дальность по воздуху). Поскольку угол между касательной и радиусом, проведенным к точке касания, прямой, здесь можно применить теорему Пифагора и сразу получить равенство

$$d = \sqrt{(R+h)^2 - R^2}. \quad (5.1)$$

При такой простой формуле и программа должна получиться простой. Только нужно вспомнить, чему равен радиус Земли. Мы это проходили очень давно, а старых учебников у меня под рукой нет. Можно было бы посмотреть в энциклопедии, но она в другой комнате, где сейчас спит мама. (Всю эту неделю она отлаживала программу по ночам: днем ее ЭВМ занята на других работах.) И тут меня осенило. Я вспомнил, что, когда вводили метрическую систему, за единицу длины была принята одна десятимилионная часть четверти парижского меридиана. Правда, тогда эту длину измерили не очень точно, но этой неточностью в нашем случае, вероятно, можно пренебречь. К тому же на самом деле Земля не шар, а нечто близкое к эллипсоиду, но мы не сильно ошибемся в расчетах, если будем считать ее шаром, у которого длина четверти окруж-

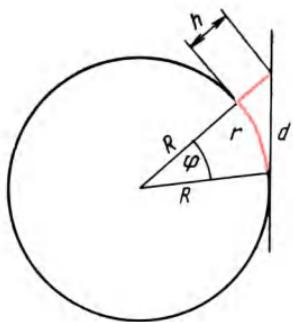


Рис. 5.1

ности большого круга равна 10 млн. метров. Отсюда легко вычислить и радиус Земли $R = 4 \cdot 10^7 / 2 \pi = 2 \cdot 10^7 / \pi$.

На микрокалькуляторе это сосчитать очень просто — 6 366 197,8 м. Можно, конечно, перевести в километры, но не стоит; ведь высоту d удобнее будет вводить в метрах. Значение R , конечно, нужно записать в каком-либо регистре памяти, например в Р1. Теперь можно и программу составить. Но в этот момент вошла мама и спросила, чем я занимаюсь. Я ей ответил, что составляю программу. Ее это очень развеселило.

- Как, ты уже программируешь? А что именно?
- Я показал ей мою формулу.
- А где же твой алгоритм?
- Какой алгоритм?
- Но нельзя же составлять программу без алгоритма.
- Разве папа тебе не сказал об этом?
- Нет. У нас даже разговора об алгоритме не было.
- А что такое алгоритм, ты знаешь?
- Ну, конечно. Мы это проходили. Слово «алгоритм» произошло от фамилии арабского ученого.

— Не совсем арабского, — возразила мама. — Он только писал на арабском языке, а по национальности принадлежал к древним узбекам и родился в IX в. в городе Хорезме. Поэтому его прозвали аль-Хорезми (что значит «хорезмиец»). В XII в. его трактат перевели на латинский язык, и европейцы впервые познакомились с описанием десятичной системы счисления и алгоритмами вычислений столбиком. Из искаженного имени аль-Хорезми и образовалось слово «алгоритм».

— Я вспомнил, есть алгоритм Евклида для нахождения общего наибольшего делителя. А вообще алгоритм — это что-то вроде правила?

— Алгоритм — это не просто правило. Это описание способа решения конкретной задачи в виде конечной последовательности однозначных описаний выполнимых операций, приводящих к нужному результату. Здесь слово

«выполнимый» означает, что исполнитель алгоритма может выполнить операцию имеющуюся у него средствами, т. е. каждая операция должна быть выполнима на твоем ПМК. В противном случае ее нужно разбить на последовательность более простых выполнимых операций.

-- А разве программа — это не описание способа решения задачи в виде последовательности операций?

— Конечно. Программа — один из способов представления алгоритма. Но для сложной задачи программу сразу составить трудно, лучше сначала составить алгоритм в виде словесного описания или схемы. Существуют определенные приемы составления и изображения алгоритмов. От них уже довольно просто перейти к программе.

— Но я уже составил две программы просто по формуле.

И я показал их маме. Мама внимательно просмотрела и сказала:

— Конечно, для такой простейшей задачи можно и не записывать алгоритм. Но по существу ты этот алгоритм составляешь в уме еще до того, как начинаешь записывать программу. А здесь у тебя даже два разных алгоритма для одной задачи. Первый из них такой:

1. Умножить V_0 на t и обозначить произведение x_0 .
Перейти к пункту 2.

2. Возвести t в квадрат. Перейти к пункту 3.

3. Умножить t^2 на a и обозначить произведение x_1 .

Перейти к пункту 4.

4. Разделить x_1 на 2 и обозначить частное x_2 . Перейти к пункту 5.

5. Сложить x_0 и x_2 и обозначить сумму S . Перейти к пункту 6.

6. Конец.

Второй алгоритм несколько отличается от первого:

1. Возвести t в квадрат. Перейти к пункту 2.

2. Умножить t^2 на a и обозначить результат x_1 . Перейти к пункту 3.

3. Разделить x_1 на 2 и обозначить результат z . Перейти к пункту 4.

4. Умножить V_0 на t и обозначить результат x . Перейти к пункту 5.

5. Сложить x и z и обозначить результат S . Перейти к пункту 6.

6. Конец.

— Но ведь это те же операции, только в другом порядке, — закричал я.

— И все же, — спокойно сказала мама, — это две различные последовательности операций и поэтому два разных алгоритма. Ты ведь сам видишь, что программы получились разные.

— А зачем после каждого пункта писать, что нужно перейти к следующему? Разве это и так не ясно?

— Пожалуй, в данном случае можно было бы и не писать: алгоритм очень простой по своей структуре, одна операция неизбежно следует за другой. Давай изобразим эти алгоритмы в виде схем. Начало и конец изображаются овалами, а все арифметические операции — прямоугольниками. Вот что у нас получилось (рис. 5.2). Видишь, все блоки этих алгоритмов выстроились по одной прямой линии. Поэтому такие алгоритмы называются линейными.

— А что это за странные знаки у тебя в прямоугольниках: знак деления и знак равенства рядом?

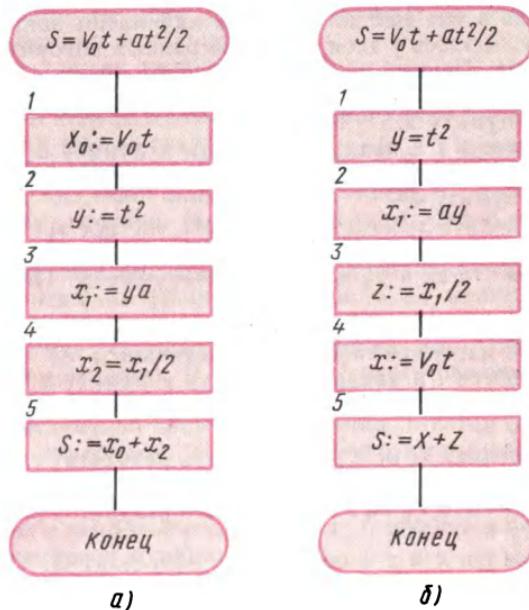


Рис. 5.2

— Это не знак деления, а знак присваивания, которое в алгоритмах имеет смысл «положим» или «установим». Когда я пишу $x_0 := V_0 t$, то это значит «обозначим через x_0 произведение $V_0 t$ » или «присвоим x_0 значение $V_0 t$ ». А чтобы не путать двоеточие с делением, для него придумали такой знак \div . Вот на микрокалькуляторе клавиша деления обозначена именно так. Очень часто в алгоритмах ты увидишь формулы вроде таких: $k := k + 1$ или $n := 2n$. Если бы здесь были обычные знаки равенства, то это можно было бы понимать только как уравнения относительно неизвестных k и n , причем первое уравнение корней не имеет, а второе имеет корень $n = 0$. А в таком виде они означают «увеличить значение k на единицу» или «увеличить значение n в 2 раза». Понятно?

— Более или менее. Постараюсь ко всему этому привыкнуть.

— Итак, ты видишь, что алгоритм можно записывать по-разному. Однако желательно использовать одинаковые способы записи алгоритмов, причем такие, которые позволяют легко по алгоритму строить программу. К сожалению, такого единого алгоритмического языка, пригодного для всех случаев жизни, нет. Собственно говоря, создать такой язык можно, но он был бы очень сложным. Им было бы трудно овладеть программисту, а еще труднее разработать транслятор, т. е. такую программу, которая преобразует алгоритм, записанный на данном языке, в программу для данной ЭВМ. Поэтому существует довольно много алгоритмических языков, предназначенных для различных классов задач и различных ЭВМ. Ты о них, конечно, слышал, это — Алгол, Фортран, Кобол, Паскаль, ПЛ/1, Фокал, Бейсик и др. И я уверена, что рано или поздно ты с некоторыми из них познакомишься.

А пока тебе надо освоить самый простой алгоритмический язык, который изучают в школе. Это упрощенный Алгол, в известной степени похожий на все остальные алгоритмические языки. Поэтому, зная его, ты легко освоишь и Бейсик, и Фортран, и вообще тот алгоритмический язык, который тебе потребуется, в зависимости от того, чем ты будешь заниматься в будущем: техникой, экономикой, медициной, музыкой или математикой. Вычислительная техника нужна везде.

— Но мы еще в школе не проходили алгоритмический язык.

— Конечно! Сейчас ты скажешь, «это мы не проходили, это нам не задавали». Надо самому приложить усилия. Ты ведь выписываешь журнал «Квант». Там начиная с № 9 1985 г. академик А. П. Ершов подробно описал суть этого языка. А кроме того, он рассказал о нем в журнале «Наука и жизнь» (№ 11 и 12 за 1985 год). А я тебе помогу. На первых порах для простых линейных алгоритмов тебе достаточно знать очень немногое, а остальное ты постепенно выучишь, когда будешь составлять более сложные алгоритмы. Вот основные положения:

1. В алгоритмическом языке около 15 служебных слов, которые обязательно употреблять во вполне определенном смысле. Они записываются сокращенно и подчеркиваются. в печатном тексте выделяются полужирным шрифтом.

2. Алгоритм должен иметь условное название. Вовсе не обязательно, чтобы это название точно определяло, какая задача решается, достаточен намек на ее сущность.

3. Запись алгоритма начинается с заголовка. В заголовке сначала записывается служебное слово алг (алгоритм), а за ним — название алгоритма прописными буквами. Далее нужно указать имена переменных, которые входят в алгоритм в качестве исходных данных (аргументов) и которые должны быть вычислены (результаты). При этом используются служебные слова, определяющие, какого типа значения эти переменные принимают: nat (натуральные числа), цел (целые), дроб (дробные), вещ (вещественные, т. е. принимающие любые целые и нецелые, в том числе иррациональные значения), лит (литерные, т. е. нечисловые данные, например «задача решения не имеет»). Переменные и тип значения их записывают в скобках. Затем нужно указать уже без скобок, какие из этих данных являются аргументами (арг) и какие должны быть выданы в качестве результатов (рез).

4. После заголовка нужно написать служебное слово нач (начало), а вслед за ним промежуточные (вспомогательные) величины, если они будут использоваться при выполнении алгоритма. В следующей строке записываются операции (разделяются точкой с запятой) в том порядке, в котором они должны выполняться. Эта последовательность операций называется серией. Если операции записываются в разных строчках, то можно между ними точки с запятой не ставить. Окончание алгоритма обозначается служебным словом кон (конец), оно записывается под словом нач.

Для начала тебе хватит и этих сведений. Теперь запиши твои два варианта алгоритма для вычисления прой-

денного пути на алгоритмическом языке. Для первого варианта

```
алг ПУТЬ 1 (вещ  $S$ ,  $V_0$ ,  $a$ ,  $t$ )
арг  $V_0$ ,  $a$ ,  $t$ 
рез  $S$ 
нач вещ  $y, x_0, x_1, x_2$ 
 $x_0 := V_0 t; y := t^2; x_1 := ya; x_2 := x_1/2$ 
 $S := x_0 + x_2$ 
кон
Для второго варианта
алг ПУТЬ 2 (вещ  $S$ ,  $V_0$ ,  $a$ ,  $t$ )
арг  $V_0$ ,  $a$ ,  $t$ 
рез  $S$ 
нач вещ  $y, x_1, z, x$ 
 $y := t^2; x_1 := ay; z := x_1/2; x := V_0 t$ 
 $S := x + z$ 
кон
```

Вот и все. Ясно?

— Да. Все очень просто, только не понятно, для чего нужно указывать, какие величины вещественные, а какие — целые или натуральные.

— Пока это, конечно, особого значения не имеет. И вообще, для микрокалькулятора это не очень существенно. Но если ты захочешь по твоим алгоритмам составить программы для большой ЭВМ, то это очень важно; в ней целые и натуральные числа записываются не в те регистры, в которых хранятся вещественные числа.

— Большое спасибо, мама, что ты меня научила записывать алгоритм. Но сейчас мне нужно составить программу по такой формуле. Я показал маме формулу (5.1). Когда ты вошла, я уже собирался записать эту программу без всякого алгоритма. Мне кажется, что здесь все так ясно, что никакой алгоритм не нужен.

— Тебе лучше было бы сказать, что ты уже составил алгоритм в уме и хочешь его сразу преобразовать в программу. В этом ничего плохого нет, иногда можно сэко-

номить время и усилия, если не записывать алгоритм в явном виде, а сразу преобразовать его в программу. Но я почти уверена, что ты составил в уме не очень хороший алгоритм.

— Почему ты в этом уверена?

— Я сказала «почти» уверена. Да потому, что такой плохой алгоритм составит любой начинающий программист. Попробуй записать свой алгоритм на алгоритмическом языке.

— Пожалуйста.

алг ВИДИМОСТЬ (вещ R, h, d)

арг R, h

рез d

нач вещ x_1, x_2

$$x_1 := R + h; x_2 := x_1^2 - R^2; d := \sqrt{x_2}$$

кон

Что в нем плохого?

— К сожалению, именно такой алгоритм я и ожидала увидеть. Этот алгоритм плох, так как его 2-я команда $x_2 := x_1^2 - R^2$ содержит опасную операцию — вычитание двух больших, очень мало отличающихся друг от друга величин. Ведь у тебя, как правило, $h \ll R$? Значит, x_1 очень мало отличается от R , а x_1^2 — от R^2 . Здесь есть опасность, что на индикаторе эти числа не будут различаться, и ты окажешься в зоне машинного нуля. Но даже если этого не произойдет, то результат такого вычитания ты можешь получить с большой относительной погрешностью. Понятно?

— Не совсем.

— Ну, возьмем, может быть, не очень реальный для твоей задачи случай, когда $h \approx 2 \cdot 10^{-9} R$. Тогда $(R + h)^2 = R^2 (1 + 2 \cdot 10^{-9} + 4 \cdot 10^{-18}) \approx R^2 (1 + 4 \cdot 10^{-9})$. Эта величина отличается от R^2 только в девятом десятичном знаке. Но твой ПМК округляет все числа до восьми значащих цифр, так что на индикаторе $(R + h)^2$ и R^2 будут представлены одинаковыми числами и их разность получится равной нулю, хотя в действительности она ненулевая. Ты оказался в зоне машинного нуля, определяемого разрядностью процессора и индикатора ПМК. Но даже и при значительно большем отношении h/R , когда ты полу-

чишь ненулевой результат, его относительная погрешность может оказаться весьма значительной. Этот вопрос заслуживает более подробного разговора, но сейчас не будем отвлекаться от нашей темы. Запомни такое правило. При составлении алгоритма старайся избегать операций, в которых из одного числа вычитается другое, очень мало от него отличающееся.

— А как это сделать?

— В каждом случае по-разному. Обычно всегда удается преобразовать формулу или алгоритм так, что эта операция вычитания заменяется другими более надежными операциями. В твоем случае это очень просто сделать. Подумай, как.

— Я, кажется, уже знаю. Раскроем в формуле (5.1) скобки в подкоренном выражении и получим

$$(R+h)^2 - R^2 = R^2 + 2Rh + h^2 - R^2 = 2Rh + h^2.$$

Таким образом,

$$d = \sqrt{2Rh + h^2}. \quad (5.2)$$

Здесь уже нет никаких вычитаний и нет опасности попасть в область машинного нуля или приблизиться к ней. Теперь алгоритм будет иметь такой вид:

```
алг ГОРИЗОНТ (вещ R, h, d)
    арг R, h
    рез d
    нач вещ x, y
        y := 2Rh; x := h^2; d := √x+y
    кон
```

Название алгоритма я изменил, чтобы не было двух разных алгоритмов с одинаковым названием.

— Вот это другое дело, — сказала мама. — По такому алгоритму вычисление будет значительно более точным. Составь программы по этим двум алгоритмам. Они дадут не совсем совпадающие результаты при малых отношениях h/R . Между прочим в этом алгоритме, как и в предыдущем, есть одна неточность. Радиус Земли R не следует включать в перечень данных, в частности в число аргументов. Ведь

аргументы — это переменные величины, которые могут принимать разные значения и которые мы задаем в каждой конкретной задаче. Здесь же у тебя R — постоянная, одинаковая для всех задач. Поэтому ее не следует упоминать вслед за названием алгоритма.

Я все же не согласился с мамой:

— Конечно, если считать, что мы привязаны к Земле, то R — постоянная. Но я, например, собираюсь когда-нибудь слетать на Марс и, может быть, построю там башню. Тогда мне придется подставлять в качестве значения R радиус Марса.

— Ну, ты и хитрец. Ладно, пусть будет по-твоему. Тем более, что на виде программы это не скажется. Но в дальнейшем постоянные величины не включай ни в аргументы, ни тем более в результаты.

Я взялся за дело и очень быстро составил два варианта программы для вычисления дальности прямой видимости. Вот они.

Программа 3. Вычисление дальности прямой видимости по формуле $d = \sqrt{(R + h)^2 - R^2}$.

Предварительно $R = 6\ 366\ 197,8$ м вводится в Р1. Величина h (в метрах) вводится в РХ. В/О С/П.

Адрес команды	Операция	Нажимаемые клавиши	Код
00	Ввод R в РХ и перевод h в РУ	ИП 1	61
01	Сложение R и h	+	10
02	Возведение в квадрат	F X	22
03	Ввод R в РХ и перевод $(R + h)^2$ в РУ	ИП 1	61
04	Возведение R в квадрат	F X	22
05	Вычитание	—	11
06	Извлечение квадратного корня	F —	21
07	Стоп	C/P	50

Результат d [м] высвечивается на индикаторе.

Введя эту программу в ПМК, я рассчитал значения d для разных h : для Останкинской телевизионной башни ($h = 500$), для Пизанской башни ($R = 56$), а также для случая, когда я стою в поле и смотрю на горизонт ($h \approx 1,6$ —

это высота моих глаз над поверхностью земли). А по маминому совету я еще ввел h для Мурзика (это наш котенок), считая, что он стоит на лапах и тоже смотрит на горизонт (в этом случае $h \approx 0,1$ м). На все это с начала ввода программы и до окончания всех вычислений ушло около 3 мин. Вот результаты:

h	500	56	1,6	0,1
d	79 793,483	26 702,059	4 582,5756	1 414,2135

На вычисление одного значения d уходит 3 с.

Затем я ввел программу по другому алгоритму.

Программа 4. Вычисление дальности d прямой видимости по формуле $d = \sqrt{2Rh + h^2}$.

Предварительный ввод такой же, как в программе 3.

Адрес команды	Операция	Нажимаемые клавиши	Код
00	Ввод h в РY		0E
01	Ввод R в РX и перевод h в РY и РZ	↑ ИП 1	61
02	Умножение h на R	×	12
03	Ввод числа 2 в РX и перевод Rh в РY и h в РZ	2	02
04	Умножение 2 на Rh	×	12
05	Обмен содержимым РX и РY	XY	14
06	Возведение h в квадрат	F X	22
07	Сложение	+	10
08	Извлечение корня	F —	21
09	Стоп	C/P	50

— Видишь, мама, эта программа получилась на два шага длиннее предыдущей. И я не знаю, как ее сократить.

— Сократить ее можно, если переписать (5.2) так: $d = \sqrt{(2R + h)h}$. Но сейчас не в этом дело. Давай про-

ведем по этой программе те же вычисления, что и по предыдущей, и найдем, насколько они отличаются друг от друга.

Это все заняло опять несколько минут, и результаты оказались такими:

h , м	500	56	1,6	0,1
d , м	по программе 3	79793,483	26702,059	4582,5756
	по программе 4	79790,023	26702,383	4513,5169
Разность Δ , м	3,46	-0,324	69,0587	285,8344
Относительная погрешность $ \Delta /d$	$4,34 \cdot 10^{-5}$	$1,21 \cdot 10^{-5}$	$1,53 \cdot 10^{-2}$	$2,53 \cdot 10^{-1}$

Здесь Δ — разность между результатами, полученными по программам 3 и 4. Для высоких башен ($h \geq 56$) эта разность ничтожно мала (тысячные доли процента) и вызвана обычной вычислительной (операционной) погрешностью ПМК. Но при малых h разность оказывается заметной и определяется методической погрешностью (т. е. погрешностью алгоритма) программы 3. Относительная погрешность 1,5 % при $h = 1,6$ и 25% при $h = 0,1$.

— Вот видишь, — сказала мама, — при очень маленьких h погрешность программы 3 становится весьма заметной, хотя для высоких башен можно было бы спокойно считать по ней.

— А если я захочу подсчитать дальность видимости с высоко летящего самолета или с космического корабля, то уж, конечно, можно воспользоваться более короткой программой 3.

— Как насчет самолета, не знаю. Все зависит от высоты полета. А вот для космического корабля ни программа 3, ни программа 4 не обеспечит хорошей точности, но уже по другой причине. При больших h длину дуги r (см.

рис. 5.1) нельзя заменить длиной касательной d . Подумай, как построить такой алгоритм, чтобы он обеспечивал хорошую точность при больших R , а я займусь своими делами.

Я опять сел думать. Взглянув на рис. 5.1, я заметил, что длина дуги r выражается через центральный угол φ . Ведь угол φ (в радианах) равен отношению длины дуги r к радиусу R , а тангенс угла φ равен отношению d к R . Отсюда следует, что $r = R\varphi = R \operatorname{arctg} (d/R)$. Поэтому в алгоритм нужно внести добавление — после того как найдено d , нужно d разделить на R , взять арктангенс от этой величины и умножить его на R . Это значит, что к программе 4 нужно после шага 08 добавить следующие шаги (продолжение программы 4):

09	Ввод R в РХ	ИП 1	61
10	Деление d на R	\div	13
11	Вычисление $\operatorname{arctg} \varphi$	F 6	1L
12	Ввод R в РХ	ИП 1	61
13	Умножение R на $\operatorname{arctg} \varphi$	\times	12
14	Стоп	C/P	50

Проходившая мимо меня мама взглянула на мою продолженную программу и покачала головой: она ей показалась длинной — целых 15 шагов.

— А разве для определения угла φ нужно вычислять d ? — спросила мама.

— Нет, конечно. Угол φ можно определить как $\arccos [R/(R+h)]$. Тогда алгоритм примет такой вид:

```

алг КРУГОЗОР (вещ  $R$ ,  $h$ ,  $r$ )
    арг  $R$ ,  $h$ 
    рез  $r$ 
нач вещ  $\varphi$ ,  $x$ 
     $x := R/(R+h)$ ;  $\varphi := \arccos x$ ;  $r := R\varphi$ 
кон

```

Из него следует такая короткая программа.

Программа 5. Вычисление дальности прямой видимости по формуле $r = R \arccos R/(R + h)$.

Ввод такой же, как в программах 3 и 4. Переключатель Р — Г в положении Р.

Адрес команды	Операция	Нажимаемые клавиши	Код
00	Ввод R в РХ и перевод h в РY	ИП 1	61
01	Сложение R с h	+	10
02	Ввод R в РХ и перевод $R+h$ в РY	ИП 1	61
03	Обмен содержимым РХ и РY	XY	14
04	Деление R на $R+h$	÷	13
05	Вычисление \arccos	F 5	1—
06	Ввод R в РХ и перевод ϕ в РY	ИП 1	61
07	Умножение R на ϕ	×	12
08	Стоп	C/P	50

— Вот видишь, — сказала мама, — вместо 15 шагов осталось 9.

Я уже научился быстро вводить программу и через несколько минут получил интересные результаты. Оказывается, что при $h < 30\ 000$ м разница между d и r не превышает 0,3 %, так что для самолетов можно и не вводить поправку и оценивать дальность видимости по d . Между прочим, для самолета, летящего на высоте 10 000 м, $r \approx 357$ км, так что, если этот полет происходит в районе станции Бологое, которая находится точно посредине между Москвой и Ленинградом, то пассажиры могут видеть на горизонте в один иллюминатор Ленинград, а в другой — Москву.

Для космических кораблей различие между d и r может оказаться заметным даже в пределах околоземного пространства. Так, при $h = 100$ км $d \approx 1133$ км, а $r \approx 1121$ км, так что разница больше 1%, а при $h = 1000$ км $d \approx 3706$ км, а $r \approx 3356$ км — уже больше 10 %.

По этой программе можно подсчитать и другие значения h , например при расположении наблюдателя на Луне.

6

КАК ЗАПИСЫВАТЬ ПРОГРАММУ, РЕДАКТИРОВАТЬ И ОТЛАЖИВАТЬ

В субботу утром я показал папе мои программы.

— Что ж, Миша, ты понемногу растешь. Пора сбросить детские штанишки, в которые заводское руководство одевает потребителей ПМК. Я имею в виду способ записи программ в виде таблицы, в которой имеется очень много лишнего или, как мы говорим, избыточного. На первых порах это, может быть, и полезно: облегчает самоконтроль при вводе программы и отыскание ошибок. Некоторые авторы сборников прикладных программ издают их в виде таблиц, похожих на те, которые рекомендуются в руководстве. Один из авторов такого сборника придумал свой метод представления программы, разграфленной на клетки, в одном углу которой указано, какие клавиши нажимаются, а в другом — код операции, высвечиваемый на индикаторе при наборе программы. При этом клавиши обозначаются той надписью, которая стоит на них, а не над ними. Так, например, он пишет не e^x , а F1, не $\sqrt{-}$, а F — , не Ig, а F2.

Конечно, читателю при этом трудно понять сущность алгоритма. Такой способ записи рассчитан только на потребителя, который не хочет ни о чем задумываться, не интересуется идеями,ложенными в основу программы, и уж во всяком случае не хочет сам учиться программировать. Предполагается, что он будет бездумно нажимать в указанном порядке клавиши и следить за высвечиваемыми кодами. Я не завидую такому читателю. Мне это напоминает ситуацию, как если бы меня заставили перепечатать текст, написанный, например, на венгерском языке, в котором я ни одного слова не знаю. Мне, скажем, предоставили для этого машинку с латинским шрифтом, буквы которого мне известны. По всей вероятности, я в конце концов выполнил бы это задание. Но сколько бы времени у меня отняло такое побуквенное копирование, сколько бы случайных ошибок я наделал и как бы я устал от этого.

Значительно более совершенная и простая система записи используется в журнале «Наука и жизнь». Однако в ней тоже много лишнего. Я сейчас познакомлю тебя с той системой, которой я пользуюсь уже несколько лет. Эта

Электроника МК-56

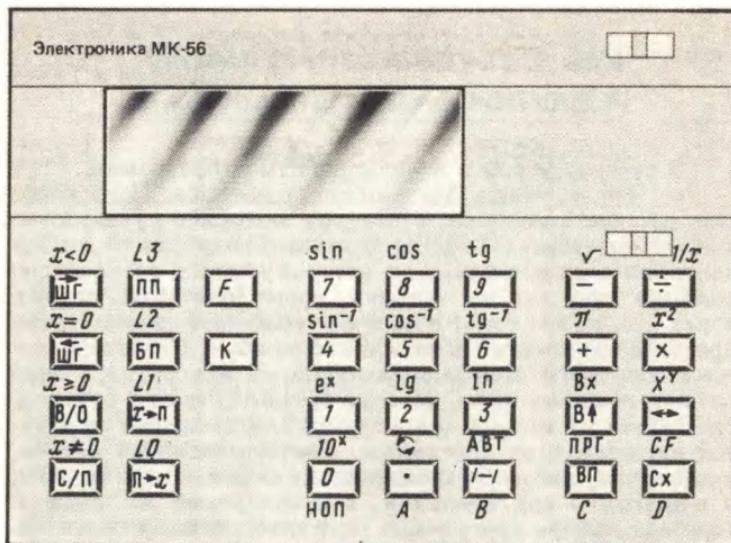


Рис. 6.1

система предложена в книге по программированию для ПМК Я. К. Трохименко и Ф. Д. Любича. По фамилиям авторов будем называть эту систему записи программ системой ТЛ. Постараюсь изложить ее систематически применительно к ПМК «Электроника Б3-34» и к другим, в которых используется тот же входной язык, хотя обозначения на клавишиах могут быть и иные.

— А какие это ПМК? Ты мне о них ничего не говорил.

— В настоящее время широко используются «Электроника МК-54» (карманный ПМК с питанием от сухих батарей и от сети) и «Электроника МК-56» (настольный небольшой ПМК с питанием только от сети). Вот на этих рисунках ты можешь видеть, как выглядят их пульты управления (рис. 6.1, 6.2).

По своей схеме и операциям они практически ничем не отличаются от Б3-34, но некоторые обозначения на клавишиах и над клавишиами у них другие. В частности, вместо arcsin , arccos и arctg используются обозначения \sin^{-1} , \cos^{-1} и \tg^{-1} , принятые во многих зарубежных странах.



Рис. 6.2

Это, к сожалению, вызывает недоразумения, так как у нас $\sin^{-1} x$ понимают обычно как $1/\sin x$. Вместо обозначения префиксной клавиши П в них использовано X → П (перевод из регистра X в регистр памяти), а вместо ИП использовано П → X (перевод из регистра памяти в регистр X). Вместо обозначения ↑ использовано B ↑ (какой смысл этому придан, не знаю), вместо $\overset{\leftarrow}{X} \vec{Y}$ — более простое обозначение ↔.

Теперь перехожу к описанию системы записи программы ТЛ, пригодной для Б3-34, МК-54 и МК-56, а также для МК-61 и МК-52.

1. Программа записывается по строчкам, причем в каждой строке 10 шагов и общее число строк не превышает

10. Строки и столбцы нумеруются от 0 до 9, но номера эти не записывают. По ним легко установить адрес любого записанного в программе оператора. Например, оператор, записанный в нулевой строке и пятом столбце, имеет адрес 05, а записанный в четвертой строке и втором столбце — адрес 42.

2. Записывается в программу не порядок нажатия клавиш, а порядок выполнения операций. Правда, большая часть операторов обозначается так же, как вводящие их клавиши. Однако, например, одноместные операции обозначаются соответствующими надписями над клавишами Б3-34 без префиксной клавиши F. Используемая многими авторами запись префикса F ничего конкретного не добавляет, является избыточной. Так, если в программе записан sin, то ввести его можно, только нажав префиксную клавишу F. Поэтому записывать ее ни к чему, если не рассчитывать на самого невнимательного вычислителя, которому нужно каждую минуту подсказывать, куда ткать пальцем. Конечно, префиксные клавиши П и ИП записывать необходимо: без них получится совсем другая операция. Это же относится и к префиксной клавише K, которой мы займемся несколько позже.

3. Для облегчения записи программ в системе ТЛ два оператора обозначаются не так, как они записаны на пульте Б3-34: вместо \overleftarrow{XY} — просто XY, без стрелок, а вместо обозначения оператора циклической перестановки операционного стека — кружка со стрелками — будем использовать \rightarrow .

4. Перед программой или после нее записывается инструкция, в которой отмечается предварительный ввод данных, порядок запуска программы, нахождение результатов вычисления, а также необходимые дополнительные указания, например положение переключателя Р — Г, если это имеет значение для данной программы.

При записи инструкции используются условные обозначения Р0, Р1, ..., Р9, РА, ..., РД. Это имена переменных, хранящихся соответственно в регистрах 0,1, ..., 9, А,...,Д. Чтобы указать, например, что значение *a* вводится в регистр А, а число 3,5 — в регистр 5, записываем так: *a* ← РА, 3,5 ← Р5. Если некоторое значение *h* вводится прямо перед вычислением *n*, следовательно, оказывается в регистре X, то это записывается так: *h* ← RX. Если некоторый результат вычисления *S* находится после остановки

ПМК в регистре X (т. е. высвечивается на индикаторе), а другой результат t — в регистре 0, то это обозначается так: $RX = S, R0 = t$.

Если программа (как это чаще всего бывает) предназначена для многократного использования с различными исходными данными, то в инструкции необходимо отметить, какие операции по вводу и запуску программы необходимо делать при каждом вычислении, а какие — только перед первым вычислением. С этой целью последние заключаются в скобки. Например, запись $(0 = P1 = P2, u = P3) z = RX (B/O) C/P RX = P, PY = \alpha$ означает, что перед первым вычислением нужно 0 ввести в регистры 1 и 2 (это часто называют так — очистить регистры 1 и 2), а значение u — в регистр 3. Затем нужно набрать значение z , которое окажется в регистре X, и нажать клавиши B/O и C/P. После окончания каждого вычисления мы видим на индикаторе вычисленное значение P . Другая вычисленная величина α находится в регистре Y, ее можно назвать, нажав клавишу XY. При последующих вычислениях нужно вводить только значения z и нажимать клавишу C/P (без B/O).

5. Когда программа составлена, отлажена, проверена и записана, необходимо привести простой пример вычисления и указать его результат и длительность вычисления. Это нужно для того, чтобы после нового ввода этой программы тобой или кем-либо другим можно было бы проверить правильность ввода. Если это «тестовый» пример вычислен правильно, то можно рассчитывать, что и другие вычисления по этой программе будут верны.

Вот так мы в дальнейшем и будем записывать программы. Давай-ка перепиши теперь свои готовые программы по этим правилам.

Я взял программы и переписал их. Конечно, мне не удалось это сделать с первого раза, но к четвертой программе я уже освоился с новой системой и убедился, что она гораздо проще старой. Вот что у меня получилось.

Программа 1. Вычисление пройденного пути S за время t при равноускоренном движении по формуле $S = V_0t + at^2/2$, где V_0 — начальная скорость; a — ускорение (мой вариант)

ИПВ ИПД \times П0 ИПД x^2 ИПА \times 2 \div

ИП0 $+ C/P$

Инструкция. ($V_0 = \text{PB}$, $a = \text{PA}$) $t = \text{РД В/С}$
 $\text{С/П PX} = S$.

Пример. $V_0 = 3 \text{ м/с}$, $a = 10 \text{ м/с}^2$, $t = 120 \text{ с}$. $S = 72\ 360 \text{ м}$. Время вычисления — около 5 с.

Программа 2. То же (папин вариант)

$\uparrow x^2 \text{ ИП1} \times 2 \div XY \text{ ИПВ} \times +$

C/P

Инструкция. ($V_0 = \text{PB}$, $a = \text{PA}$) $t = \text{PX B/O}$
 $\text{С/П PX} = S$.

Пример. Тот же, что в программе 1.

Программа 3. Вычисление дальности прямой видимости d по формуле $d = \sqrt{(R+h)^2 - R^2}$, где R — радиус Земли; h — высота точки наблюдения

$\text{ИП1} + x^2 \text{ ИП1} x^2 - \sqrt{C/P}$

Инструкция. ($R = 6\ 366\ 197,8 \text{ м} = P1$)
 $h = \text{PX B/O C/P PX} = d$. Точность 1,5 % обеспечивается при $1,6 \text{ м} < h < 100 \text{ км}$.

Пример. При $h = 56 \text{ м}$ $d = 26\ 702,059 \text{ м}$, при $h = 300 \text{ м}$ $d = 61\ 806,148 \text{ м}$. Время вычисления — около 3 с.

Программа 4. Вычисление дальности прямой видимости d по формуле $d = \sqrt{2Rh + h^2}$.

$\uparrow \text{ИП1} \times 2 \times XY \text{ x}^2 + \sqrt{C/P}$

Инструкция. ($R = 6\ 366\ 197,8 = P1$) $h = \text{PX B/O C/P PX} = d$.

Пример. При $h = 56 \text{ м}$ $d = 26\ 702,383 \text{ м}$, при $h = 300 \text{ м}$ $d = 61\ 804,6 \text{ м}$, при $h = 0,1 \text{ м}$ $d = 1\ 128,3791 \text{ м}$. Время вычисления — около 3,5 с. Точность порядка 0,01 % сохраняется при сколь угодно малых h .

Программа 5. Вычисление радиуса прямой видимости r по формуле $r = R \arccos [R/(R+h)]$

$\text{ИП1} + \text{ИП1} XY \div \arccos \text{ ИП1} \times C/P$

Инструкция. Та же, что и к программе 4. Программа пригодна для больших значений h , вплоть до космических, и обеспечивает погрешность меньше 0,01 %.

Пример. При $h = 1000 \text{ км} (1 \cdot 10^6 \text{ м}) r = 3355,936 \text{ км}$. Время вычисления — около 4 с.

Папа мои программы в основном одобрил, но только заметил, что в программах 4 и 5 я на глазок оценил точность в 0,01 %.

— Конечно, алгоритмы в этих программах очень точные, количество операций четыре-пять, так что если даже на каждой операции относительная погрешность будет порядка 10^{-7} , то общая погрешность вычислений не превысит 10^{-6} или 0,0001 %. Но здесь нужно учесть, что исходные данные имеют большую погрешность. Радиус Земли R дается с абсолютной погрешностью до $\pm 12\ 000$ м, потому что Земля — не шар, а эллипсоид. Если посмотришь в энциклопедию, то увидишь, что экваториальный радиус $6\ 378\ 245$ м, а полярный — $6\ 356\ 863$ м. Ты же считал $6\ 366\ 198$ м, так что относительная погрешность составляет уже $\pm 0,2\%$.

Вероятно, еще больше относительная погрешность для высоты башни h . Так что в лучшем случае здесь можно обеспечить точность $\pm 0,5\%$. Вообще же точность программы определить не так просто. Это следует делать в процессе ее отладки.

— Кстати, папа, ты ведь обещал рассказать мне, что такое редактирование и отладка программы.

— Да ведь ты уже редактировал программу. Вот сегодня, когда ты в первый раз набрал свою программу 5, она у тебя сначала не пошла. Помнишь, когда ты ввел $h = 1 \cdot 10^6$ м, у тебя получилось $r = 1$ м, ты сразу понял, что здесь что-то не так.

— Да, я понял, что неправильно ввел программу.

— И что же ты сделал?

— Я выключил питание и тем самым стер введенную программу, а затем набрал ее снова, более внимательно, и все пошло нормально.

— Вот видишь, ты и отредактировал программу, но только самым примитивным образом, полностью переписал ее. Конечно, для такой короткой программы в 9 шагов это разумно. Но если бы у тебя была программа шагов на 80, то такое редактирование не привело бы быстро к цели.

— А как же нужно было поступить?

— Нужно было проверить введенную программу и найти в ней ошибку. Ведь ты и сейчас не знаешь, в чем она состояла. А я знаю. Ты при вводе программы пропустил одну операцию — ИП1 с адресом 06. Сейчас я тебе расскажу, какие меры следует применять при вводе и редактировании программы. Между прочим, редактированием программы называется отыскание и устранение ошибок, совершенных при вводе программы в программную память ПМК или при вводе данных в регистры памяти.

Что же касается отладки, то под ней понимают выявление и устранение ошибок, допущенных при составлении программы.

Начнем с методов редактирования программы, о которой мы знаем, что она верна, т. е. в свое время была отлажена. Ошибки при вводе программы могут быть трех видов: 1) пропуск какой-то команды; 2) ошибочный ввод не той команды, которая нужна; 3) ввод лишней команды. Опыт показывает, что первый вид ошибок встречается значительно чаще, чем остальные два. Поэтому важно уметь обнаруживать их уже в процессе самого ввода программы.

Для этого нужно помнить: при записи программы по системе ТЛ в конце каждой строчки высвечивается адрес, оканчивающийся нулем. Если не так, то в этой строчке какая-то команда пропущена или вставлена лишняя. Нужно найти ошибку и исправить. Чтобы найти ошибку,

пользуются клавишами \vec{W} и \vec{G} , продвигающими программу на один шаг вперед и на один шаг назад соответственно. Продвигая таким образом набранный фрагмент программы, необходимо сравнивать высвеченные коды операторов с операторами, записанными в программе. На первых порах придется использовать таблицу команд из заводского руководства. Однако очень скоро ты запомнишь коды наиболее часто встречающихся команд и сможешь отыскивать ошибки, не прибегая к таблице.

Если в конце каждой строки программы высвечивается адрес, оканчивающийся нулем, а проверка программы по известному примеру дает неверный результат, приходится искать ошибку в вводе какого-то оператора по всей программе. Этот поиск может облегчить клавиша ПП, с которой ты еще не встречался. Она выполняет две различные функции в зависимости от режима работы. В режиме ПРГ эта клавиша вводит команду «Переход к подпрограмме». С этим мы познакомимся позже. А сейчас же нас интересует клавиша ПП в режиме АВТ, когда ее обозначение можно расшифровать как «Пошаговое прохождение». Если при введенной программе нажать клавишу В/О, а затем вместо клавиши С/П — клавишу ПП, то ПМК выполнит первую команду программы. Следующее нажатие той же клавиши заставит ПМК выполнить вторую команду, и так можно по шагам пройти всю программу до конца. После каждого шага на индикаторе высвечивается его результат, и, следя за программой, обычно удается легко выявить тот шаг, после которого результат отличается от ожи-

даемого. Очевидно, либо этот шаг, либо один из предшествующих шагов набран неправильно. Конечно, при таком редактировании программы надо немного пошевелить мозгами, но это полезно.

Чтобы исправить обнаруженную неправильную команду, необходимо, нажимая клавиши ШГ в режиме ПРГ, выйти на адрес ошибочной команды и ввести правильный оператор. Запись новой команды стирает команду, которая находилась по этому адресу ранее. Лишнюю введенную команду можно заменить оператором НОП (Нет операции), чтобы не набирать заново остальные команды.

Для отладки новой программы используются те же средства, что и при редактировании. Но здесь мы не уверены в правильности записанной на бумаге программы и не имеем проверенного примера. Поэтому такой пример нужно создать, рассчитав его вручную. Задаемся самыми простыми исходными данными, чтобы пример можно было легко решить даже в уме. Например, чтобы проверить программу 5, положим $R = 4$ и $h = 1$. Тогда получится (см. рис. 5.1) пифагоров треугольник с гипотенузой 5 и катетами 4 и 3. Следовательно, $d = 3$, и $\cos \Phi = 0,8$, откуда в режиме ручного вычисления находим $\Phi \approx 0,6435$ и $r \approx 2,574$. Введя в Р1 значение $R=4$ и в РХ $h=1$, запускаем программу. Если получим те же результаты, считаем программу отлаженной. В противном случае нужно было бы искать ошибку в самой программе. Этому помогает пошаговое прохождение вычисления с помощью клавиши ПП. В дальнейшем тебе не раз придется отлаживать программы и ты приобретешь необходимую сноровку.

7

ВИТЯЗИ НА РАСПУТЬЕ

Сегодня выходной, и мы накануне решили отправиться гулять. Вот только никак не могли решить куда. Папа предлагал поехать в Петродворец. Мама же сказала, что, если будет идти дождь, как в последние дни, вряд ли мы там получим удовольствие. Поэтому она предложила пой-

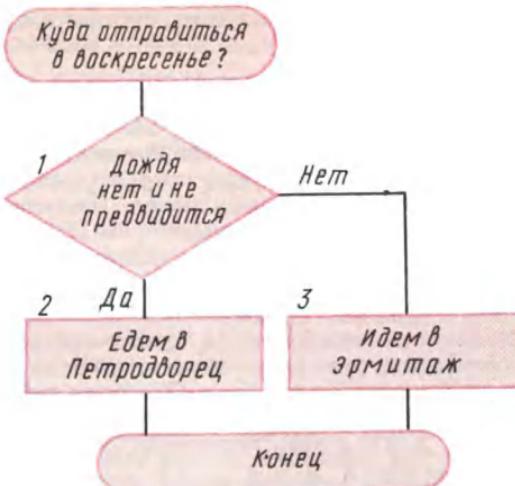


Рис. 7.1

ти в Эрмитаж. Я возразил: нужно рано встать, чтобы занять очередь. Мы долго спорили, пока папа не сказал, что утро вечера мудренее. Завтра проснемся, увидим какая погода, послушаем прогноз по радио и тогда примем решение.

— Значит, алгоритм такой, — сказала мама. — Если дождя нет и не предвидится, едем в Петродворец. В противном случае идем в Эрмитаж.

— Какой же это алгоритм? Это же не правило, а условие, — возразил я.

— Конечно это алгоритм, — ответила мама, — только не линейный, а алгоритм с ветвлением. Он содержит логический блок, в котором записано условие. Если это условие выполняется, мы идем дальше по одному пути, а если не выполняется — по другому. В схеме такие логические блоки изображаются ромбом, из которого исходят две стрелки, указывающие путь: Да — когда условие выполняется и Нет — когда оно не выполняется. В данном случае наш алгоритм имеет такую схему (рис. 7.1). А на алгоритмическом языке этот алгоритм с ветвлением записывается так:

алг ВОСКРЕСЕНЬЕ (лит. А, В)

арг А

рез В

нач если А == «дождя нет и не предвидится»

то В := «едем в Петродворец»

иначе В := «едем в Эрмитаж»

все

кон

Ты видишь здесь новые служебные слова: **если**, **то**, **иначе**, **все**. Последнее обозначает окончание ветвления. Слова **если** и **все** являются «скобками», обрамляющими ветвление.

— А можно из этого алгоритма сделать программу для ПМК?

— К сожалению, нет: первая операция для ПМК не выполняется, он не умеет определять и предсказывать погоду. — ответила мама.

— Значит, из алгоритма с ветвлением программы не сделать?

— Почему же? — вмешался папа. — Большая часть программ, которые я составлял, имеет ветвления, циклы, подпрограммы и т. д.

Наутро мы применили наш алгоритм, но не совсем удачно. Дождя с утра не было. В прогнозе по радио тоже ничего определенного о дожде не говорилось. Поэтому мы поехали в Петродворец, но не успели доехать, как хлынул дождь. Мы вернулись в Ленинград. В Эрмитаж ехать было уже поздно, и я решил заняться ПМК, в первую очередь выяснить, что это за алгоритмы с ветвлением и как их можно превратить в программу. С этим вопросом я обратился к маме.

— Это очень просто. Я покажу тебе на примере той задачи, с которой ты начал свой путь будущего программиста, — вычисления пройденного пути при равноускоренном движении. Возьмем в качестве исходного образца наш алгоритм ПУТЬ 2 (см. стр. 31), по которому у тебя была составлена программа 2. Несколько конкретизируем задание. Предположим, что ты подбрасываешь строго вертикально вверх камень, придавая ему начальную скорость V_0 (см/с), при этом начальная его высота над зем-

лей H_0 (м). На какой высоте H над землей он будет через время t (с)?

— Да ведь эту задачу я уже решал. И обходился без всякого ветвления. Правда, я подбрасывал не камень, а мяч.

— Я заменила мяч на камень, чтобы тебе было легче.

— А разве подбрасывать камень легче, чем мяч?

— Легче считать. Мяч упругий, отскакивает от земли, а камень как упадет на землю, так и останется лежать. Давай составим алгоритм, исходя из формулы (4.1), в которой положим a равным ускорению силы тяжести $a = -9,81 \text{ м/с}^2$ (считая положительным направление вверх). Высота камня над землей в данный момент t определяется как $H = H_0 + S$. Если это величина положительная, то камень еще не долетел до земли. Если же $H_0 + S < 0$, то $H = 0$, т. е. на самом деле камень уже на земле. Итак, алгоритм получится такой:

```
алг КАМЕНЬ (вещ  $H_0$ ,  $V_0$ ,  $t$ ,  $H$ )
    арг  $H_0$ ,  $V_0$ ,  $t$ 
    рез  $H$ 
нач вещ  $S$ ,  $x$ 
     $S := V_0 t - 9,81 t^2 / 2$ ;  $x = H_0 + S$ 
    если  $x \geq 0$ 
        то  $H := x$ 
        иначе  $H := 0$ 
    все
кон
```

— Попробуем немного усложнить задачу. Проследим весь путь камня, отмечая все его положения, скажем, через интервалы в 0,1 с. По этим данным ты сможешь составить таблицу положения камня в разные моменты времени, а также приблизительно определить, сколько времени будет продолжаться его полет. Обозначим $t [k]$ некоторый k -й момент времени, а через $H [k]$ — высоту камня в этот момент. Наш предыдущий алгоритм КАМЕНЬ войдет в новый алгоритм в качестве *вспомогательного*. На этом примере ты увидишь некоторые новые служебные слова и познакомишься с понятием цикла. Обозначим T длитель-

ность полета камня с точностью до 0,1 с. Вот какой будет алгоритм:

```
алг ПОЛЕТ (вещ  $H_0$ ,  $V_0$ ,  $T$ , таб  $H[k]$ )
    арг  $H_0$ ,  $V_0$ 
    рез  $H[k]$ ,  $T$ 
нач (нат  $k$ , таб вещ  $t[k]$ )
     $k := 0$ ;  $t[k] := 0$ ;  $H[k] := H_0$ 
пока  $H[k] \neq 0$ 
    иц  $k := k + 1$ ;  $t[k] := t[k] + 0,1$ 
    КАМЕНЬ ( $H_0$ ,  $V_0$ ,  $t[k]$ ,  $H[k]$ )
кц
     $T := t[k]$ 
ком
```

Здесь служебное слово **таб**, как ты сам понимаешь, обозначает таблицу, в данном случае из вещественных чисел, а натуральное число k называется индексом. Бывают таблицы и с несколькими индексами. Таблица $t[k]$ нам дана, а таблицу $H[k]$ мы можем вычислить. Результатом является также T — время до момента, когда $H[k]$ станет равным нулю.

Конечно, можно было бы найти это время более точно, решая уравнение, но этим ты займешься позднее. А пока отыщем T с точностью до 0,1 с по этому алгоритму. В дальнейшем ты будешь строить и более сложные программы с ветвлениями.

Далее, в этом алгоритме ты впервые сталкиваешься с циклом. Цикл — это многократно повторяемая последовательность операций. Перед описанием цикла в алгоритме ставится служебное слово **пока**, вслед за которым дается условие, при выполнении которого цикл выполняется. В нашем алгоритме условие $H[k] \neq 0$. Как только это условие окажется невыполненным, цикл прекращается (алгоритм «выходит из цикла»). Сам цикл представляет собой серию команд, обрамленных служебными словами («скобками»): иц (начало цикла) и кц (конец цикла). Они обязательно должны находиться на одной вертикали. Таким образом, когда в процессе выполнения алгоритма мы подходим к концу цикла, нужно вернуться к слову **пока** и

проверить, выполняется ли условие цикла. Если да, то цикл повторяется. Если условие не выполняется, цикл считается законченным, и мы переходим к команде, следующей за словом **кц** (в данном случае заканчиваем вычисления). Иногда вместо **иц** пишут **цикл**.

При составлении алгоритма с циклами необходимо тщательно проверить, всегда ли обеспечивается выход из цикла. Если алгоритм составлен неудачно, то может произойти «зацикливание» и программа будет работать «вечно», пока не будет искусственно остановлена. В нашем случае условие $H[k] \neq 0$ не может вечно выполнятся, поскольку при $a < 0$ величина S рано или поздно становится отрицательной и по абсолютному значению возрастает.

— А как мы будем считывать значения $H[k]$? Нужно предусмотреть каждый раз остановку микрокалькулятора?

— Ну, конечно, если бы калькулятор имел печатающее устройство, то мог бы, как ЭВМ, сразу распечатать таблицу $t[k]$, $H[k]$. А так тебе придется ограничиться нахождением общего времени полета T либо действительно каждый раз останавливать ПМК, чтобы списать результат. Только не говори «остановка». Остановка бывает у трамваев и автобусов, а программисты говорят «останов».

Я это запомнил. Но в тот же вечер, когда произнес это слово «останов», получил нагоняй от папы.

— Почему ты коверкаешь русский язык? Такого слова нет.

— Но все программисты говорят и пушут «останов».

— Подумаешь! Во времена моей молодости все электрики говорили «искrá» (с ударением на последнем слоге), но сейчас, слава богу, кажется, говорят правильно — искара. Железнодорожники долгое время считали слово «путь» существительным женского рода — «поезд отправляется с четвертой пути». Сейчас такого не услышишь. Надеюсь, что и слово «останов» скоро забудется. Но пока приходится слышать и читать страшные вещи. В одной инструкции к электронным часам я прочел «установ» вместо «установка». Я не удивлюсь, если прочитаю про «установ рекорда по заготовке кормов». Давай, все-таки, хранить чистоту русского языка: и говорить и писать «остановка», а не «останов».

Я целиком согласен с папой. Но как быть с мамой? Она иронически улыбается, когда я говорю «остановка», и переспрашивает «троллейбусная?» Поэтому я, чтобы не раздражать ни одну из сторон, решил применять нейтральный термин «стоп».

Возвращаясь к нашему утреннему разговору я должен сказать, что моя попытка привлечь маму к составлению программы по ее алгоритму потерпела неудачу..

— Нет уж уволь. Ты же знаешь, что я мало что смыслю в микрокалькуляторах. Обращайся с этим к отцу. А еще лучше, сам попытайся в этом разобраться. Ты ведь уже не маленький.

Последний мамин довод на меня подействовал. Действительно, неужели я не могу сам разобраться по руководству, как строить программу с ветвлениеми и циклами? И я засел за руководство.

Я, конечно, уже понимал, что ветвления и циклы осуществляются командами *условных и безусловных переходов*. По этим командам осуществляется переход на определенный адрес программы, не в обычном порядке следования адресов. Например, если по адресу 21 введена команда безусловного перехода (БП), а за ней, на шаге 22, — число 42, то это значит, что при прохождении программы после адреса 21 пойдут не адреса 22, 23 и т. д., а осуществляется переход на адрес 42 и будет выполняться та команда, которая там записана, а после нее — 43, 44 и т. д. Немного сложнее с условными переходами. Они осуществляются не всегда, а только в случае невыполнения некоторого условия. Таких условий в нашем ПМК предусмотрено четыре, и все они связаны с значением числа, находящегося в данный момент в РХ. Это условия $x < 0$, $x = 0$, $x \geq 0$ и $x \neq 0$. После ввода в программу какого-либо из этих условий нужно на следующем шаге указать адрес, на который переходит программа, если данное условие не выполнено. Если же указанное условие выполнено, то программа «перескакивает» через адрес и продолжает свой естественный ход. Например, программа

ИП1 ИП2 — $x \geq 0$ 25 ИП3 + ...

если разность между числами, хранящимися в Р1 и Р2, отрицательна (т. е. условие на шаге 03 $x \geq 0$ не выполнено), переходит на шаг 25, выполняет содержащуюся по этому адресу команду и продолжает выполнение команд на шагах 26, 27 и т. д. Если разность положительна или равна нулю, эта программа перескакивает через шаг 04 и извлекается число из Р3, которое затем прибавляется к разности.

В устройстве ПМК есть одна интересная хитрость. Если бы я захотел ввести в программу двузначное число 25 не после команды перехода, а после любой другой команды,

то мне пришлось бы потратить на это два шага и в программе в качестве кодов должно быть записано 02 05. Это значит, что в общем случае числа вводятся в программу по отдельным цифрам. Но если введена команда условного или безусловного перехода, после которой нужно ввести адрес перехода, то ПМК терпит только двузначные числа. Поэтому в моем кусочке программы число 25, введенное после $x \geq 0$, займет всего один шаг и будет записано в программе в обычной форме 25. А вот если я по ошибке на шаге 04 вместо адреса 25 набрал 24 и захотел потом это исправить, то во второй раз ввести двузначное число мне бы не удалось. Пришлось бы снова возвращаться на шаг 03, вторично набрать $x \geq 0$, после чего ПМК воспримет двузначный адрес 25. Разобравшись с этим, я вернулся к маминому алгоритму.

Сначала я решил составить программу по вспомогательному алгоритму КАМЕНЬ. Собственно говоря, она почти ничем не отличается от моей программы I (см. стр. 43). Нужно только добавить в конце ветвление. Таким образом, я получил следующее:

Программа 6. Вычисление положения брошенного камня

ИПВ	ИПД	\times	ИПД	x^2	9	,	8	1	\times
2	\div	—	ИПЮ	$+ x \geq 0$	19	ПА	С/П	0	
ПА СП									

Инструкция. $t = \text{РД}$, $V_0 = \text{РВ}$, $H_0 = \text{Р0}$
 $B/O \text{ С/П } RX = PA = H$.

Примеры. При $t = 2$ с, $V_0 = 12$ м/с, $H_0 = 1$ м
 $H = 5,38$ м; при $t = 3$ с, $V_0 = 12$ м/с, $H_0 = 1$ м $H = 0$.
Время вычисления 8 с.

В этой программе ветвление осуществляется на шаге 15. Перед этим я вычислил $x = S + H_0$ и сейчас проверяю условие $x \geq 0$. Если оно не выполнено, т. е. $x < 0$, то я пересекаю на шаг 19, т. е. ввожу 0 в RX и записываю его в PA. Если же условие выполнено и $x > 0$, то шаг 16 пропускаю, сразу записываю полученное значение x в PA и заканчиваю на этом расчет.

Итак, программа по алгоритму КАМЕНЬ готова, теперь нужно составить программу по алгоритму ПОЛЕТ. Для этого нужно только изменять значения t и при каждом из них вычислять H , пока не будет выполнено условие $H = 0$, и отсчитать после этого значение T . Впрочем,

я решил не останавливать ПМК для вывода значений $H [k]$ через каждую десятую долю секунды, а ограничиться вычислением общей длительности полета T .

Программа 7. Полет подброшенного камня

$$\begin{array}{rcl} \text{ИПД} & 0 & , \quad 1 \quad \div \quad \text{ПД ИПВ} & \cdot \quad \text{ИПД} \quad x^2 \\ & 9 & , \quad 8 \quad 1 & \quad 2 \quad \div \quad - \quad \text{ИПЮ} \quad - \\ x \geqslant 0 & 24 \quad \text{БП} \quad 00 \quad \text{С} \quad \text{П} \end{array}$$

Инструкция. $V_0 = \text{РВ}$, $H_0 = \text{Р0}$; $0 \quad \text{РД}$
 $\text{ВОСПРД} = T$.

Примеры. При $V_0 = 2 \text{ м/с}$, $H_0 = 1 \text{ м}$ $T = 0,7 \text{ с}$.
время вычисления — около 1 мин; при $V_0 = 6 \text{ м/с}$, $H_0 = 1 \text{ м}$ $T = 1,4 \text{ с}$, время вычисления — около 2 мин.

Я вначале удивился, что по короткой программе ПМК так долго считает, но потом сообразил: цикл ведь повторяется много раз, во втором примере 14 раз, так что ничего странного здесь нет.

8

ЧТО ТАКОЕ ХОРОШО И ЧТО ТАКОЕ ПЛОХО?

Когда папа вернулся с работы, я с гордостью показал ему свою программу 7. Папа отнесся к ней одобрительно.

— Молодец. За неделю ты научился составлять программы для более или менее простых задач. Теперь тебе надо учиться составлять *хорошие* программы.

— Как? — опешил я. — Разве эта моя программа плохая?

— Я этого не говорю. Но все же ей многое не хватает. Лично я, составив такую программу, не успокоился бы, а попытался ее улучшить.

— Что значит улучшить? Сделать ее короче или точнее?

— Нет, ни в коем случае. Что касается длины программы, то, конечно, твою программу можно укоротить. И, конечно, при прочих равных условиях более короткую программу нужно всегда предпочесть более длинной. Но

это только при прочих равных условиях. В данном же случае я предпочел бы иметь более длинную программу, но свободную от некоторых недостатков.

— В чем же ее недостатки? Я их не замечаю.

— Не замечаешь, потому что не хочешь заметить, да и опыта у тебя еще мало. Попробую пояснить тебе основной недостаток. Представь, на производстве установили очень умный станок с программным управлением, который очень точно и довольно быстро обтачивает тяжелые металлические детали. Но доставлять эти детали и устанавливать их на станке приходится вручную. Токари говорят, что лучше бы механизировали транспортировку детали, а обточить ее мы сами сможем. Вот и у тебя значительная часть времени уходит на ввод постоянных и на вывод результатов. И дело тут не только в затратах времени и внимания. При таких программах легко наделать много ошибок.

Конечно, если рассматриваемая программа является лишь частью большей программы, важно обеспечить ее краткость, чтобы иметь больше возможностей для ее пополнения. Но в общем случае есть более важные показатели качества, чем длина программы. И одно из основных качеств — это простота и удобство обращения с программой, в частности минимум ручных операций. Лишь в тех случаях, когда программа не умещается в отведенном количестве шагов, приходится ради сокращения числа команд жертвовать удобством.

В общем, нелегко дать точный перечень всех качеств, которыми должна обладать хорошая программа для ПМК. Это, конечно, и время вычисления, и точность результатов, и допустимые исходные данные, и способ вывода результатов. В дальнейшем, поднабравшись опыта, ты будешь легко отличать хорошие программы от плохих и даже начнешь понимать, что такое «красивая» программа, так же как шахматные мастера ощущают красоту партии, которая любителям недоступна. Так вот, твоя программа не красивая и не очень хорошая.

— Чем же она некрасива?

— Во-первых, неудобным способом ввода данных и вывода результата. Во-вторых, наличием внутри цикла операций, которые можно вынести за его пределы.

— Предложи тогда более красивую программу.

— Надо подумать: даже самый опытный программист не может сразу составить идеальную программу. Но попробуем нащупать пути улучшения твоей программы.

Если начать с конца, то непонятно, чего ради ты заставляешь вычислителя выводить результат, нажимая клавиши ИПД? Не проще ли поручить это самому микрокалькулятору? Перейдем теперь к началу. Перед каждым новым вычислением ты должен набрать значения V_0 и H_0 и ввести их вручную в нужные регистры. Более того, ты должен не забыть ввести 0 в РД. Лучше это поручить программе. Конечно, ввода таких данных, как V_0 и H_0 , избежать нельзя. Но не будем заставлять пользователя напрягать внимание перед каждым вычислением, чтобы поместить эти данные в нужные регистры. Это тоже введем в программу. Повышенное внимание потребуется только один раз при вводе самой программы.

Далее, константы 0,1 и 9,81 ты вводишь в программе, но при каждом прохождении цикла. Это занимает много времени. Введем их один раз, до начала цикла, и запишем в регистры памяти. И, конечно, нецелесообразно сначала умножать t^2 на 9,81, а затем делить на 2. Лучше сразу умножить t^2 на $9,81/2 = 4,905$. Вообще старайся всегда уменьшать количество и продолжительность операций внутри цикла, даже за счет существенного увеличения их вне цикла.

Исходя из этих соображений, я бы составил такую программу.

Программа 8. Продолжительность полета камня (па-
пин вариант)

ПВ	С/П	П0	0	ПД	0	,	1	П1	4
,	9	0	5	П2	ИПД	ИП1	+	ПД	ИПВ
×	ИПД	x^2	ИП2	×	—	ИП0	+	$x < 0$	15
ИПД	С/П								

Инструкция. $V_0 = RX \text{ B/O С/П}$, $H_0 = RX \text{ С/П } RX = T$.

— Как видишь, я выбрал более рациональное условие перехода $x < 0$, и поэтому мне удалось обойтись без безусловного перехода. Правда, моя программа длиннее твоей на 7 шагов, но я убежден, что считать она будет быстрее, не говоря уж о том, что пользоваться ею удобнее.

Я ввел эту программу и проверил те два примера, которые использовал с программой 7. Результаты получились, конечно, те же, но значительно быстрее — за 40 с для первого примера и за 1,5 мин — для второго.

— Вот видишь, можно по хорошему алгоритму составить хорошую программу, но нужно немного подумать. Если не думать, а просто вслепую переносить из алгоритма все операции в программу, то, как правило, эта программа будет работать (если, конечно, у тебя хватит программной памяти и регистров), но она может оказаться не очень хорошей. В одних случаях она будет считать очень медленно, а в других ею просто будет неудобно пользоваться. В этом нет ничего удивительного. Составляя алгоритм, ты учитывашь только сущность задачи, но не приспособливаешь ее к особенностям твоего ПМК. Поэтому записанный алгоритм годится для любой ЭВМ, но не для всякой из них составленная программа окажется хорошей.

Как правило, современные большие ЭВМ считают так быстро, что и по ненаилучшей программе выдают решение почти мгновенно. Поэтому для них вопрос о составлении оптимальных программ стоит не так остро. Но наши калькуляторы работают пока довольно медленно. Поэтому для них очень важно уметь составлять экономные программы. Я уже не говорю о том, что плохо составленная программа иногда просто не поместится в программной памяти. Поэтому я хочу тебе показать пару примеров, как с помощью небольших усилий можно сократить и улучшить программу.

Пример 1. Вычисление степенного многочлена. Возьмем достаточно простой случай. Нужно вычислить при различных значениях x многочлен

$$P(x) = ax^4 + bx^3 + cx^2 + dx + e. \quad (8.1)$$

Предположим, что коэффициенты a, b, c, d и e записаны соответственно в РА, РВ, РС, РД и Р0. Значение x мы вводим в РХ и нажимаем клавиши В/О С/П. Постарайся составить программу, занимающую минимум шагов.

— Сначала нужно значение x запомнить в какомнибудь регистре, например Р1, ведь оно нам будет нужно много раз. Затем, скажем, я умножу x на d и сложу с e . Это можно записать в тот же Р0, в котором был записан коэффициент e .

— Стоп! Ты забыл, что наш многочлен нужно будет вычислять при разных значениях x , а поэтому все коэффициенты, записанные в регистрах памяти, должны сохраняться нетронутыми.

— Хорошо, — согласился я. — Запишем сумму в Р2 и потом будем к ней прибавлять cx^2 , bx^3 и ax^4 . Зна-

чение x^3 проще всего вычислить как xx^2 , а значение x^4 — как $(x^2)^2$. В общем получится такая программа?

П1 ИПД × ИПЮ + П2 ИП1 x^2 ИПС ×
+ П2 ИП1 ...

Нет, — спохватился я. — Нужно было бы x^2 записать в память, чтобы легче вычислять x^3 и x^4 . Сделаем так:

П1 ИПД × ИПЮ + П2 ИП1 x^2 П3 ИПС
× + П2 ИП1 ИПЗ × ИПВ × + П2
ИПЗ x^2 ИПА × + С/П

Получилось 26 шагов.

— Теперь посмотри, много ли у тебя лишнего. Ты совершенно правильно заметил, что после шага 10, когда ты вычислил cx^3 , в РY находится ранее вычисленная сумма $dx + e$, которую ты записал в Р2, и поэтому ее вызывать из Р2 не нужно, а можно сразу производить сложение. Та же картина будет и дальше. Какой из этого можно сделать вывод?

— Можно Р2 вообще не использовать, суммы все будут храниться в операционном стеке. Значит, программа будет такая:

П1 ИПД × ИПЮ + ИП1 x^2 П3 ИПС ×
+ ИП1 ИПЗ × ИПВ × + ИПЗ x^2 ИПА
× + С/П

Ура! Я сэкономил три шага. Теперь программа занимает 23 шага.

— Ну, ура кричать рановато. Можно программу еще подсократить. Вот один из вариантов. Он основан на том, что наш многочлен можно записать в таком виде:

$$ax^4 + bx^3 + cx^2 + dx + e = (((ax + b)x + c)x + d)x + e. \quad (8.2)$$

Признаюсь, это придумал не я. Такой метод вычисления многочленов давно известен программистам и называется по фамилии автора схемой Горнера. Теперь легко записать программу так.

Программа 9. Вычисление многочлена 4-й степени (8.1)

П1 ИПА × ИПВ + ИП1 × ИПС + ИП1
× ИПД + ИП1 × ИП0 + С/П

И н с т р у к ц и я. ($a = \text{PA}$, $b = \text{PB}$, $c = \text{PC}$, $d = \text{PD}$, $e = \text{P0}$) $x = \text{RX}$ В/О С/П RX = P(x).

Осталось всего 18 шагов.

— А нельзя ли еще сократить программу?

— Почему же нельзя? Можно. У программистов в ходу такая шутка: всякую программу, если хорошо подумать, можно сократить на один шаг. Но сейчас не будем этим заниматься. Позднее, когда ты освоишь косвенную адресацию, мы сделаем это. А пока вот тебе другой пример.

В задачах по теории вероятности очень часто приходится вычислять значения w по формуле

$$w = e^{-x^2/2} / \sqrt{2\pi}. \quad (8.3)$$

Предположим, что нам нужно найти w для разных значений x , вводя их в RX и нажимая клавиши В/О С/П. Попробуй составить программу минимальной длины.

— Сейчас попробую составить программу без всяких хитростей, а потом постараюсь ее сократить. Итак,

x^2 2 ÷ /—/ e^x 2 π × √ ÷

СП

Как видишь, здесь всего 11 шагов и все очень просто. Зачем же еще сокращать?

— Прежде всего затем, что это вычисление может быть кусочком другой, достаточно сложной задачи, которая без этого сокращения не умещается в программную память ПМК. К тому же, если такой фрагмент вычисляется много раз, то, сокращая его длину, мы экономим время вычисления. Между прочим, в твоей программе есть один недостаток. Во многих экземплярах БЭ-34 операция смены знака /—/ в режиме автоматического вычисления работает неверно. Поэтому ее следует избегать. Для того чтобы вычислить $e^{-\alpha}$, имея α в RX, лучше использовать не /—/, e^x , а другую пару операций e^x , $1/x$. В нашем случае еще проще — вместо того чтобы умножать на $e^{-\alpha}$, можно разделить на e^α и этим сэкономить шаг. Но попробуй повозиться с формулой и получить сокращение хотя бы еще на шаг.

Я стал думать, но с ходу ничего не придумал. Тогда папа стал мне подсказывать:

— Что произойдет с числом e^{-x^2} , если мы показатель разделим на 2? Другими словами, возьмем $e^{-x^2/2}$?

— Это то же самое, что извлечь корень квадратный.

— Попробуй записать это в таком виде.

Я написал: $w = \sqrt{e^{-x^2}} / \sqrt{2\pi}$.

— А теперь вспомни, что мы говорили относительно умножения на отрицательную степень.

— Это деление на положительную степень. Значит,

$$w = \frac{1}{\sqrt{2\pi}} \left/ \sqrt{e^{-x^2}} \right. = \frac{1}{\sqrt{2\pi e^{x^2}}} \quad (8.4)$$

— Теперь формула совсем упростилась, и ты можешь составить очень короткую программу.

— Конечно, — ответил я и записал

$x^2 e^x \pi \times 2 \times \gamma 1 x C/P$

Итак, всего 9 шагов.

— Я хотел бы, чтобы ты выработал у себя привычку при составлении каждой программы добиваться ее сокращения, но, конечно, не за счет точности и удобства обращения. Сегодня мы рассмотрели способы сокращения и улучшения программ на базе тех операций, с которыми ты уже хорошо знаком. Завтра я покажу тебе, как для этого использовать еще один оператор.

9

ДЛЯ ЧЕГО В ПМК СЧЕТЧИКИ?

— Реши-ка простую задачу, — сказал папа, прияя с работы. — Чему равно 5^{10} ?

— А что в этом трудного? Я ввожу 10 в РY и 5 в РX и включаю команду X^Y (клавиши F и XY). И вот тебе сразу результат 9 765 623.

— Ты меня не понял. Я хотел узнать, чему в точности равно 5^{10} .

— А что, разве этот ответ, который дал ПМК, не точный? Какие у тебя основания сомневаться?

— Во-первых, команду X^Y ПМК выполняет с помощью логарифмов, с большой погрешностью. Во-вторых, я хорошо знаю, что число 5^n при любом целом n в десятичной записи заканчивается пятеркой: $5^2 = 25$, $5^3 = 125$, $5^4 = 625$ и т. д. Ты же мне выдал ответ, в котором последняя цифра тройка.

— Но тут легко внести поправку. Ошибка не может быть очень большой. Поэтому $5^{10} = 9\ 765\ 625$.

— Ладно, в этом случае ты выпутался. А если мне нужно вычислить 7^9 ? Что даст твой ПМК?

Я быстро ввел эти числа, подал команду X^Y и получил $7^9 = 40\ 353\ 591$.

— Как ты думаешь, это точный ответ?

— Не знаю. Вообще-то степень семерки может оканчиваться единицей.

— А ты попробуй прямо умножить девять раз семерку на самое себя. Для этого лучше всего ввести число 7 в РХ, РY, РZ и РT, т. е. нажать клавиши $7 \uparrow \uparrow \uparrow$. Теперь, нажимая клавишу \times , ты будешь каждый раз умножать число в РХ на 7.

Я так и сделал. Нажимаю первый раз клавишу \times и получаю $49 = 7^2$, следующее нажатие дает $7^3 = 343$, затем получаю $7^4 = 2401$ и так далее, пока не дошел до $7^9 = 40\ 353\ 607$.

— Вот видишь, — сказал папа, — оказывается команда X^Y дала абсолютную погрешность — 16.

— А почему ты уверен, что результат $40\ 353\ 607$ верен?

— А откуда здесь может взяться ошибка? Ведь тут обычные действия с целыми числами без всяких округлений. Теперь предположим, что тебе нужно вычислить точно значения $A = m^n$, где m и n — целые числа. Как бы ты записал алгоритм и построил программу?

— Алгоритм может быть такой:

алг СТЕПЕНЬ (цел m , n , A)

арг m , n

рез A

нач цел k

$k := 1; A := 1$

пока $k \leq n$

иц

$A := Am; k := k + 1$

кц

кон

— Очень хорошо. А теперь попробуй по этому алгоритму составить программу.

— Попробую. Я буду записывать m в Р0, n в РВ, A в РА и k в Р1. Все операции буду делать, как в алгоритме.

1 П1 ПА ИП0 \times ПА ИП1 1 + П1
ИПВ ИП1 — $x \geq 0$ 18 ИПА БП 03 ИПА С.П

Для проверки я подсчитал 5^{10} и получил верный ответ — 9 765 625. На это ушла минута.

— А не думаешь ли ты, что это чересчур расточительно тратить 20 шагов программы на целую минуту на такую операцию, как возведение целого числа в целую степень?

— Но я же все делал по алгоритму.

— Да, но, во-первых, ты уже знаешь, что по одному алгоритму можно составить программу по-разному. хорошо или плохо. А, во-вторых, может быть, у тебя не лучший алгоритм? Давай рассмотрим его с этой точки зрения. У тебя условие выхода из цикла $k > n$, а на языке ПМК такого условия нет и тебе приходится в программе изворачиваться, вычитать k из n . Может быть, лучше не увеличивать k от 1 до n , а уменьшать от n до 1? Тогда условием выхода из цикла будет просто $k = 0$. Поэтому перепишем алгоритм и дадим ему другое название, чтобы не путать с твоим.

алг ВОЗВЕДЕНИЕ (**цел** m , n , A)

арг m , n

рез A

нач **цел** k

$k := n; A := 1$

пока $k \neq 0$

иц

$A := Am; k := k - 1$

кц

кон

Теперь составим программу. Будем записывать A в РА, k в Р0, m в Р1 и n в РВ.

Программа 10. Возведение в степень $A = m^n$

1 ПА ИПВ П0 ИПА ИП1 \times ПА ИП0 1

— П0 х = 0 04 ИПА С/П

Инструкция. $m = \text{Р1}$, $n = \text{РВ}$ В/О С/П РХ =
= РА = m^n .

По новому алгоритму программа сократилась на 20 % — до 16 шагов. Кроме того, она работает быстрее, так как в ней нет безусловного перехода, который был в твоей, дополнительно к условному.

Я проверил эту программу на том же примере: $5^{10} = 9\ 765\ 625$. ПМК просчитал это за 35 с, т. е. почти вдвое быстрее, чем по моему алгоритму.

— Обрати внимание на шаги с 8 по 13 (я их подчеркнул). Такие фрагменты программ встречаются очень часто, когда необходимо организовать повторение цикла n раз. В некоторый регистр записывается целое число n , определяющее, сколько раз нужно повторить цикл. При каждом прохождении цикла из содержимого этого регистра (в наших примерах Р0) вычитается единица и проверяется, не уменьшилось ли оно до нуля. Если этого не произошло, то осуществляется переход к началу цикла, а если произошло, то программа выходит из цикла.

Так как к таким методам организации цикла приходится прибегать довольно часто, конструкторы ПМК предложили выполнять указанные операции с помощью двухшагового оператора. С этой целью в язык ПМК Б3-34 и другие введены четыре команды L0, L1, L2 и L3. При вводе команды L0 (с помощью префиксной клавиши F и клавиши ИП) из целого числа, записанного в регистре 0, вычитается единица и проверяется, не равна ли разность нулю. Если не равна, то эта разность записывается в Р0 и осуществляется переход на адрес, записанный в программе вслед за командой L0. Таким образом, команда L0 заменяет пять шагов, подчеркнутых в программе 10. При выполнении команды L0 никаких изменений в операционных регистрах не происходит. Аналогично действуют команды L1, L2 и L3, но они относятся к другим регистрам — Р1, Р2 и Р3. Это позволяет осуществить в пределах одной программы до четырех циклических последовательностей. Иногда команды L0, L1, L2 и L3 применяют и для других

целей. Давай-ка применим команду L0 в нашей последней программе 10. Как ты думаешь, на сколько шагов удастся сократить программу?

— Ясно, на четыре шага. Ведь вместо пяти команд мы применяем одну. Вместо 16 шагов будет 12.

— Нет, Миша, ты поторопился с выводом. Ведь те команды, от которых мы избавляемся, вносят определенные изменения в операционный стек. Поэтому необходимы еще шаги для вызова нужных величин в операционные регистры. Так, например, в программе 10 приходится каждый раз после подчеркнутых команд вызывать промежуточный результат A из РА в РХ: он там не остается. Теперь же все промежуточные произведения A будут сохраняться в операционном стеке, только перемещаясь из РХ в РУ и обратно. Поэтому их можно вообще не записывать в РА, и от программы 10 останутся несколько шагов, которые и программой назвать стыдно.

Программа 11. Возведение в степень

ИПВ П0 1 ИП1 × L0 03 С/П

Инструкция. $m = P1$, $n = PB$ В/О С/П
 $PX = A$.

— Как видишь, от 16 шагов осталась половина. Проверь-ка, сколько времени такая программа будет считать 5^{10} .

Я проверил. Оказалось, всего лишь 16 с.

— Поговорим еще относительно команд L0 ... L3, которые часто облегчают составление программ. Они являются счетчиками, но считают «назад» от введенного в соответствующий регистр целого положительного числа до единицы, после чего счет останавливается и в регистре остается единица, сколько бы раз мы не повторяли операцию LN. С другой стороны, операторы L0 ... L3 являются операторами условного перехода. Условие, при котором переход осуществляется, заключается в том, что целая часть числа, хранящегося в соответствующем регистре (P0 ... P3), не равна единице. При использовании оператора в программе для организации цикла достаточно помнить, что, записав в соответствующий регистр N целое положительное число n, можно с помощью команды LN обеспечить повторение цикла n раз.

Обрати внимание еще на одну деталь, которая, собственно говоря, к программированию не относится, но которую следует знать, чтобы не удивляться. Предположим, ты ввел в Р0 число 8. Если ты его выведешь в РХ, то на инди-

каторе увидишь в самом обычном виде цифру 8. Пусть теперь запущена программа, в которой была команда L0. При этом, конечно, число 8 в Р0 уменьшилось на единицу. Если после этого нажмешь клавиши ИП0, то ты ожидаешь увидеть на индикаторе число 7 также в самом обычном виде. А что ты увидишь на самом деле?

— Не знаю.

— Это, безусловно, верно. Ты этого сейчас не знаешь. А как сделать, чтобы ты узнал?

— Нужно попробовать.

Я набрал коротенькую программу, по которой любое число вводится в Р0, затем осуществляется операция L0 с переходом на следующий же шаг программы. Другими словами,

П0 L0 03 С/П

Введя число 8 и нажав клавиши В/О С/П, я через мгновение увидел на индикаторе опять число 8.

— Погоди, — сказал папа. — Это число сохранилось в РХ. А что находится в регистре памяти? Нажми-ка ИП0.

Я нажал и увидел вместо обычного числа 7 такую запись: 00000007.

— Почему это так? — удивился я.

— Просто так было удобнее конструкторам, — ответил папа. — Никакого влияния на вычисления это не оказывает. В Р0 содержится число 7, которое можно переслать в любой другой регистр, заставить участвовать в одноместных или двухместных операциях, и везде оно будет себя вести как обычное целое число 7, несмотря на то, что записано оно в Р0 с дополнительными семью нулями в старших разрядах. С такого рода записью целых чисел мы еще встречимся.

— А что будет, если в Р0 записано не целое число?

— Если оно положительное и больше единицы, то команда L0 будет действовать так, как если бы в регистре присутствовала только целая часть этого числа. Так, если в Р0 записано число 5,68, то по команде L0 переход совершился, а в Р0 окажется число 4. Если в Р0 записано 5,9999999, т. е. почти 6, то все равно команда L0 учитывает только целую часть этого числа, т. е. 5, уменьшает его на единицу, так что остается 4. Если же в Р0 записано 1,8, то для команды L0 это все равно, что 1, переход не происходит, а в Р0 остается то, что там было записано, т. е. 1,8.

Более странные вещи происходят, если в Р0 записано положительное число, меньшее 1, например 0,6. В этом случае переход происходит. Ведь переход не происходит

только в том случае, когда целая часть числа, записанного в регистре, равна 1. А вот содержимое Р0 изменяется по-другому. В частности, 0,6 превратится в $5,9999999 \cdot 10^{-1}$, т. е. уменьшится на $1 \cdot 10^{-8}$. Если же было записано число 0,06, то оно уменьшится на $1 \cdot 10^{-9}$, т. е. станет $5,9999999 \times 10^{-2}$. При исполнении команды L0 любое число в интервале $(1 \cdot 10^{-2}, 1 \cdot 10^{-1})$ уменьшается на $1 \cdot 10^{-9}$, а все положительные числа в интервале от $2 \cdot 10^{-10}$ до $1 \cdot 10^{-2}$ на $1 \cdot 10^{-10}$. Эти свойства иногда используются при программировании.

Я не буду останавливаться на том, каким изменениям подвергаются отрицательные числа, хранящиеся в Р0 Р3 при выполнении команды LN для этого регистра. Там соотношения очень сложные и, насколько мне известно, никогда при программировании не используются.

— А для чего же тогда ввели такие сложные зависимости для команды LN?

— Их никто специально не вводил. Когда разрабатывали ПМК и создавали команду LN, конструкторы ставили перед собой одну задачу — реализовать счетчик для организации цикла, выполняющий функции шести команд, аналогичных тем, которые мы подчеркнули в программе 10. Они заботились только о тех ситуациях, когда в управляемый регистр введено целое положительное число. А во всех остальных ситуациях получилось то, что получилось.

Реши-ка простую задачу, используя команду LN. Что такое факториал, ты знаешь?

— Конечно. Это произведение последовательных натуральных чисел $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n - 1) n$. Я даже знаю, что $n!$ — это число различных перестановок каких-либо n предметов. Ты ведь сам мне в прошлом году об этом рассказывал.

— Действительно, был такой разговор. Так вот, составь программу для вычисления факториала, включающую как можно меньше шагов. Я через полчаса подойду.

С этими словами папа оставил меня одного. Я очень хочу быть самостоятельным, только вот вопрос — с чего начать?

Конечно, заданное число n нужно ввести в РХ, затем запустить ПМК и на индикаторе будет $n!$. Я уже понимаю, что начинать с 1, умножить ее на 2, затем произведение умножить на 3 и т. д. — путь не лучший. Здесь пришлось бы каждый новый множитель сравнивать с n . Конечно, лучше начать с n , умножить его на $n - 1$ затем на $n - 2$

и т. д. до единицы. А уменьшение множителя каждый раз на единицу лучше всего сделать командой LN , которая позволит и организовать цикл умножений. Но запишем прежде всего алгоритм

```
алг ФАКТОРИАЛ (цел  $n$ ,  $A$ )
арг  $n$ 
рез  $A$ 
нач
     $A := n$ 
    пока  $n > 1$ 
        нц
             $n := n - 1$ ;  $A := nA$ 
        кц
    кон
```

Итак, я должен прежде всего в Р0 записать n . Кажется, я смогу не записывать число A в регистр памяти, а хранить его в операционном стеке. Дальше я просто выполняю команды алгоритма и замыкаю цикл командой LN . В результате получаю такую программу:

П0 L0 03 ИП0 × L0 03 С/П

Инструкция. $n := RX$ В/О С/П $RX = n!$ В этой программе 8 шагов. Команду L0 пришлось применить 2 раза. Считает она правильно. Но нельзя ли сделать ее еще короче? Я ничего не смог придумать, а тут папа вернулся. Посмотрев на мою программу, он улыбнулся.

— Я уже говорил, что, по мнению программистов, всякую программу можно сократить на один шаг. Правда, они имеют в виду сложные длинные программы для больших ЭВМ. Здесь же программа простенькая и короткая. Однако смотри, вот программа на шаг короче:

П0 1 ИП0 × L0 02 С/П.

Здесь в RX введена единица, которая умножается на n , благодаря чему первый сомножитель оказался в таком же положении, как и остальные. Впрочем, твоя программа считает, вероятно, быстрее: в ней производится n умножений, а в моей $n + 1$.

Мы вооружились секундомером и измерили время вычислений. Действительно, моя программа затрачивает на 1 с меньше. Так, $5! = 120$ моя программа вычислила за 7 с, а папина — за 8; $10! = 3628800$ моя программа вычислила за 14 с, а папина — за 15 с.

— Почему же, — спросил я, — программа, записанная по алгоритму, оказалась длиннее твоей, которую ты составил, не глядя ни на какой алгоритм?

— Вовсе нет, — ответил пapa, — моя программа тоже выполнена по алгоритму, но несколько лучше приспособленному для ПМК. Давай, я его запишу:

алг ПЕРЕСТАНОВКА (цел n , A)

арг n

рез A

нач

$A := 1$

пока $n > 1$

нц

$A := nA$; $n := n - 1$

кц

кон

Видишь, небольшая перестановка операций не меняет результата вычислений, но делает алгоритм более приспособленным к ПМК.

— Обе ваши программы никуда не годятся, — сказала мама, прислушивавшаяся к нашему разговору. Если в них ввести $n = 0$, то они зациклятся и никогда не дадут верного ответа.

— Но можно указать в инструкции, что n — целое положительное число ($n > 0$), и тогда никто не будет вводить $n = 0$, — сказал я.

— Если это отдельная программа, то можно, — сказала мама. — Но я не думаю, чтобы кто-нибудь, кроме тебя, вычислял факториалы из любви к искусству программирования. Обычно приходится их вычислять в ходе ре-

шения больших и сложных задач, где эти факториалы фигурируют, и никогда заранее нельзя сказать, какое n окажется введенным в эту часть программы. Поэтому нужно предусмотреть, чтобы и при $n = 0$ программа давала правильное решение $0! = 1$.

— Как $1!$ — закричал я. — Ведь очевидно $0!$ равен нулю. Если мы будем умножать, начиная с нуля и заканчивая единицей, то нуль и получится.

Тут папа и мама рассмеялись.

— Ты не очень внимательно следишь за определениями, — сказал папа. — То определение факториала, которым ты пользуешься, т. е. $n! = 1 \cdot 2 \cdot 3 \dots n$, для $n = 0$ явно не годится. Начав с единицы и беря каждый следующий множитель на 1 больше предыдущего, ты до нуля никогда не дойдешь. Поэтому можно сказать, что в рамках твоего определения $0!$ вообще не существует. Но, как это часто бывает в математике, возникают задачи, которые требуют обобщения ранее определенных понятий на более широкий класс объектов. Вот и потребовалось определить $0!$. И тогда обратили внимание на существование такого соотношения: $n! = n(n - 1)!$, из которого следует также $(n - 1)! = n!/n$. Такие формулы, которые позволяют найти функцию $f(n)$ по значениям этой функции от других аргументов, называют рекуррентными. Если добавить условие $1! = 1$, то любая из этих двух формул может быть принята за определение факториала. Ты можешь легко убедиться, что при любом $n > 1$ они вполне совпадают со старым определением. Но, кроме того, вторая формула позволяет определить и $0!$

$$0! = 1!/1 = 1.$$

Вот почему мама совершенно права. Наша программа при вводе $n = 0$ должна бы давать такой же ответ, как и при $n = 1$. А она этого не делает и даже зацикливается.

Между прочим, это беда не алгоритма, а самой программы. Если буквально выполнить алгоритм ПЕРЕСТАНОВКА, то при $n = 0$ условие цикла с самого начала не выполняется, и поэтому вместо перехода к циклу алгоритм заканчивается и выдает верный результат $A = 1$. Но, к сожалению, программа, построенная по этому алгоритму, не работает, если $n = 0$, так как оператор LN к этому не приспособлен. Если же реализовать алгоритм, не прибегая к оператору LN , а заменяя его вычитанием единицы и сравнением разности с нулем, то программа будет работать и при $n = 0$, но будет очень длинной.

Многие авторы предлагали программы для вычисления факториала, в которых использовался условный переход, дававший при $n = 0$ сразу ответ 1, например

П0 x \neq 0 09 1 ИП0 × L0 04 С/П 1 С/П

Но в этой программе 11 шагов, так что ради одного значения аргумента пришлось удлинить программу больше чем на 50 %.

Очень интересное и оригинальное решение нашли А. Н. Цветков и В. А. Епанечников [3]. Они воспользовались одним свойством Б3-34. Если в РХ находится 0 и вводится команда ВП (Ввод порядка), то 0 превращается в 1. Таким образом, на индикаторе отсчитывается $1 \cdot 10^0$, т. е. 1. Всякое же другое число в РХ при нажатии клавиши ВП не изменяется. Это позволило им сделать программу для $n!$ длиной всего в 8 шагов, пригодной и для $n = 0$.

Программа 12. Вычисление факториала $n!$ при $n \geq 0$

ВП П0 1 ИП0 × L0 03 С/П

Инструкция. $n = \text{РХ}$ В/О С/П РХ = $n!$

Примеры. $10! = 3\ 628\ 800$; $40! = 8,1591516 \cdot 10^{47}$. Время вычисления — около 1,5 n с.

— А у дробных чисел факториалы бывают? — спросил я.

— Конечно, — ответил папа. — Понятие факториала распространяли и на дроби, как положительные, так и отрицательные. Но об этом в другой раз.

— Во многих формулах, — заметила мама, — встречается также «двойной факториал» — число с двумя восклицательными знаками.

— Это, вероятно, факториал от факториала? — догадался я.

— Нет, ничего подобного. Это условное обозначение произведения натуральных чисел, взятых через одно:

$$n!! = \begin{cases} 1 \cdot 3 \cdot 5 \dots (n-2) n & \text{для нечетных } n, \\ 2 \cdot 4 \cdot 6 \dots (n-2) n & \text{для четных } n. \end{cases}$$

Попробуй-ка составить программу для вычисления $n!!$

— Правильно, — сказал папа. — Это тебе хорошее домашнее задание. А вот и другое. Ты можешь легко убедиться, что при $n > 69$ $n! > 10^{100}$ — разрядная сетка ПМК переполнена, на индикаторе высвечивается ЕГГОГ. Но предположим, что я все-таки хочу получить представ-

ление хотя бы о порядке величины, скажем $100!$ или $1000!$.
Придумай, как это сделать, и составь программу.

С этими словами родители бросили меня одного с моими заданиями. Я решил, что должен в конце концов справиться с ними самостоятельно.

Первое задание особых трудностей не вызвало. Я быстро составил программу, почти не отличающуюся от программы 12. При этом я учел, что $n!! = (n+2)!!/(n+2)$, откуда вытекает, что $0!! = 1$. Поэтому я применил ту же хитрость для превращения нуля в единицу.

Программа 13. Вычисление $n!!$ для целых $n \geq 0$

ВП П0 1 ИП0 \times L0 07 L0 03 С/П

Инструкция. $n = РХ$ В/О С/П РХ = $n!!$

Примеры. $7!! = 105$, $8!! = 384$. Время вычисления — около 8 с.

Со вторым заданием я просидел дольше. Собственно говоря, составить программу для него оказалось совсем нетрудно, но я не сразу додумался, как к нему подойти, ведь числа, большие 10^{100} , ПМК не признает. Вдруг меня осенило — логарифмы таких больших чисел довольно умеренны и их вполне можно считать на ПМК. По десятичному логарифму числа можно точно судить о порядке его величины. Итак, я составил следующую программу, основанную на очевидной формуле $\lg(n!) = \lg 1 + \lg 2 + \lg 3 + \dots + \lg n$.

Программа 14. Вычисление $\lg(n!)$ для любых целых $n \geq 0$

ВП П0 Сх ИП0 Ig + L0 03 С/П

Инструкция. $n = РХ$ В/О С/П РХ = $\lg(n!)$

Примеры. $\lg(100!) = 157,97$; $\lg(1000!) = 2567,6$.

Отсюда следует, что $100!$ немного меньше 10^{158} , а $1000!$ лежит между 10^{2567} и 10^{2568} . Это, конечно, невообразимо огромные числа.

Считал я эти примеры долго. Из самой программы с циклом, повторяющимся n раз, видно, что время вычисления приблизительно пропорционально n . Я нашел коэффициент пропорциональности и с помощью секундомера определил время вычисления (в секундах) как $3n$. Действительно, $\lg(100!)$ я вычислял 300 с, т. е. 5 мин, а на вычисление $\lg(1000!)$ ушло в 10 раз больше — 50 мин. За это время я успел принять душ и приготовиться ко сну. Итак, спокойной ночи!

10

ВАРИАЦИИ НА ТЕМУ О ФАКТОРИАЛЕ. ГАММА, НЕ ИМЕЮЩАЯ ОТНОШЕНИЯ К МУЗЫКЕ

Сегодня впервые папа, кажется, остался доволен моими программами. На счет программы 13 он сказал:

— Прекрасно. Ты правильно заметил, что $0!! = 1$. На это иногда не обращают внимания даже авторы хороших книг.

По поводу программы 14 он выразил полное удовлетворение.

— Действительно, для таких огромных величин, как $1000!$, ничего другого не придумаешь, как вычислять логарифм и по нему судить о порядке величины. Что же касается $100!$, то его можно вычислить и по старой нашей программе 12.

— Как же так? — удивился я. — Ведь $100!$, как мы убедились, это почти 10^{158} , а наш ПМК умеет считать только до $9 \cdot 10^{99}$.

— Не совсем так, хотя это и написано в руководстве. Наш ПМК ведет себя как скромный человек, который не афиширует всех своих умений. Когда с ним познакомишься поближе, то оказывается, что он умеет считать до 10^{300} , а это значительно больше, чем $100!$ Он только не умеет выставлять такие большие числа на индикаторе и поэтому вместо вычисленного результата выдает ЕГГОГ, показывая этим, что мы переполнили разрядную сетку. Но он только шутит. Если мы разделим это число, например, на 10^{99} , то он выдаст результат деления. Попробуй!

Я быстро набрал программу 12, благо в ней всего 8 шагов, и ввел в нее 100. Микрокалькулятор стал считать и примерно через 4 мин выдал «результат» ЕГГОГ, но я не испугался, набрал вручную: $1 \text{ ВП } 99 \div$ и увидел результат $9,332619 \cdot 10^{58}$.

— А теперь, — сказал папа, — нужно обратно умножить на 10^{99} , но уже в уме. Сколько же получится?
 $9,332619 \cdot 10^{58+99} = 9,332619 \cdot 10^{157}$.

— Действительно, около 10^{158} , как и с помощью логарифмов.

Мне это очень понравилось. Этот прием можно применить не только при вычислении факториалов, но и при

любом расчете, когда возникает переполнение разрядной сетки. Я решил вычислить $150!$ по той же программе 12. На этот раз получился «результат» даже не ЕГГОГ, а ЭГГОГ.

— Это потому, — сказал папа, — что ты превзошел даже 10^{200} .

Я опять вручную ввел 1 ВП 99 \div , после чего получил ЕГГОГ. Дальше я набрал F Вх \div и получил ответ $5,7133784 \cdot 10^{64}$, а значит, с учетом двух делений на 10^{99} , такой результат:

$$150! = 5,7133784 \cdot 10^{64+99+99} = 5,7133784 \cdot 10^{262},$$

— Вот видишь, — сказал папа, — как можно расширить область применения ПМК. Но будь осторожен, не доходи до реального предела 10^{300} , иначе ПМК может рассердиться и перестать считать или даже перейти самопроизвольно в режим программирования и спутать тебе все карты. Поэтому такие величины, как $1000!$ и даже $170!$, считай только с помощью логарифмов, по твоей программе 14. Вот уж с этой программой тебе никогда переполнение разрядной сетки не грозит.

— А почему? Вот я возьму и задам такое большое n , что даже $\lg(n!)$ будет больше 10^{300} .

— Потому что на это просто не хватит времени. Пусть, например, $n = 10^{10}$. Для него $\lg(10^{10}!)$ приблизительно равен 10^{11} , что еще очень далеко до переполнения. Однако продолжительность вычисления $\lg(n!)$ по программе 14, как ты сам определил, равна 3л секунд. Подсчитай-ка на ПМК, сколько это составит лет при $n = 10^{10}$.

— Сейчас будет сделано.

Я перевел $3 \cdot 10^{10}$ в минуты, часы, сутки и годы и увидел, что на мое вычисление уйдет немного больше 950 лет.

— Я от всей души желаю тебе самых долгих лет жизни, но все же не надеюсь, что тебе удастся по своей программе вычислить величину $\lg(10^{10}!)$.

— А откуда же ты, папа, определил, что $\lg(10^{10}!) \approx 10^{11}$? Считал это на большой ЭВМ?

— Нет, стариk, если у ЭВМ быстродействие будет даже в 100 тысяч раз больше, чем у твоего ПМК, то и тогда, как ты можешь проверить, $\lg(10^{10}!)$ она будет считать несколько суток. Но существуют приближенные формулы, которые позволяют оценить факториалы больших чисел с помощью довольно простых вычислений. Так, для очень больших n можно определить порядок величины $n!$ по

такой простой формуле, справедливой при любом основании логарифмов:

$$\log(n!) \approx n \log n. \quad (10.1)$$

Ею я и воспользовался. Насколько это приближенное значение близко к истинному, мы с тобой узнаем, когда подсчитаем достаточно точное значение $\lg(10^{10}!)$. И для этого нам не придется считать ни 950 лет, ни даже несколько суток. Мы воспользуемся приближенной формулой Стирлинга, которую удобнее всего представить в следующем виде:

$$\begin{aligned} \ln(n!) &\approx S(n) = (n + 1/2) \ln n + \\ &+ \ln \sqrt{2\pi} - n + 1/(12n). \end{aligned} \quad (10.2)$$

Эта формула асимптотическая. Это значит, что отношение $\ln(n!)/S(n)$ с увеличением n становится все ближе к единице. Если вместо $n!$ взять величину $e^{S(n)}$, то относительная погрешность будет тем меньше, чем больше n . Так, при $n = 1$ $n! = 1$, а $e^{S(n)} = 1,00227 \dots$, так что относительная погрешность равна $2,27 \cdot 10^{-3}$, при $n = 4$ она равна $4,275 \cdot 10^{-5}$, а при $n = 10$ уменьшается до $2,31 \cdot 10^{-6}$. При $n > 20$ относительная погрешность меньше $1 \cdot 10^{-6}$ и уменьшается с дальнейшим увеличением n , хотя и довольно медленно. Разумеется, абсолютная погрешность с увеличением n возрастает, так как очень быстро растет сама величина $n!$

— Подожди, папа, я что-то не совсем понимаю. Абсолютная погрешность — это разность между полученным результатом и истинным значением $\Delta = e^{S(n)} - n!$ и она с увеличением n возрастает. А относительная погрешность — это отношение абсолютной погрешности к истинному значению. Она равна $\Delta/n!$. Ну, понятно, так как Δ растет очень быстро, то, несмотря на возрастание Δ , относительная погрешность уменьшается. А что важнее, абсолютная или относительная погрешность?

— Это зависит от обстоятельств. Но в большинстве случаев интересуются относительной погрешностью. Поэтому формулой Стирлинга широко пользуются. Попробуй составь программу для вычисления $\lg(n!)$ по формуле Стирлинга. Обрати только внимание на то, что по формуле Стирлинга ты получаешь натуральный логарифм $n!$, а нам нужен десятичный. Поэтому не забудь разделить $S(n)$ на $\ln 10$.

Я довольно быстро записал алгоритм вычисления $S(n)$ и программу для $S(n)/\ln 10$:

```
алг СТИРЛИНГ (цел  $n$ , веществ  $S$ )
арг  $n$ 
рез  $S$ 
нач веществ  $z_1, z_2$ 
 $z_1 := (n + 0,5) \ln n; z_2 := n - 1/12n;$ 
 $S := z_1 - z_2 + \ln \sqrt{2\pi}$ 
кон
```

Программа 15. Вычисление $\lg(n!)$ по формуле Стирлинга при $n \gg 1$

П1 \ln ИП1 2 $1/x + \times 2 \pi \times$
 $\sqrt{-\ln +}$ ИП1 — ИП1 1 $2 \times 1/x$
+ 1 0 $\ln \div$ С/П

Инструкция. $n = РХ$ В/О С/П РХ = $\lg(n!)$.

Пример. $\lg 10! = 6,559764$ (истинное значение 6,5597627), $\lg 50! = 64,483076$ (истинное значение 64,4830745). Время вычисления при любом n около 15 с.

— Теперь, — сказал папа, — ты можешь проверить, насколько точно я оценил $\lg(10^{10}!)$ по простейшей формуле $n \lg n$.

Я ввел в ПМК значение $n = 10^{10}$ и через 15 с получил $\lg(10^{10}!) = 9,565705 \cdot 10^{10}$.

— Как видишь, Миша, моя грубая оценка отличается от этого значения всего лишь на 4,3 %.

— А можно ли считать, что этот результат $9,565705 \times 10^{10}$ совершенно точный?

— Ну, конечно, нет. Во-первых, как видишь, в нем всего семь знаков, тогда как даже одна целая часть этого результата должна содержать 10 знаков. Микрокалькулятор округляет результат до восьми значащих цифр. Поэтому в нем содержится относительная ошибка округления, составляющая по крайней мере $10^{-9} \dots 10^{-8}$. На самом деле в процессе вычисления также возникают ошибки, которые накапливаются, и фактически относительная погрешность в полученном результате по меньшей мере

должна достигать $10^{-7} \dots 10^{-5}$. Поэтому, выдавая этот результат, лучше ограничиться пятью значащими цифрами и написать $\lg(10^{10}!) = 9,5657 \cdot 10^{10}$.

Конечно, — продолжил папа, — относительная методическая погрешность, определяемая методом, в данном случае формулой Стирлинга, с увеличением n уменьшается. Однако относительная погрешность вычисления и округления с ростом n возрастает. Поэтому общая относительная погрешность результата $\lg(n!)$ по программе 15 остается примерно постоянной: $10^{-6} \dots 10^{-5}$ при изменении n от 40 до очень больших значений. Хуже обстоит дело с относительной погрешностью самой величины $n!$, если она вычисляется с помощью формулы Стирлинга. Это легко понять, если немного подумать. Обозначим для удобства неизвестное нам истинное значение $n!$ через x , а $\lg(n!) = \lg x$ через y . Относительную погрешность при вычислении этих величин будем обозначать δ_x и δ_y , а абсолютную — соответственно Δ_x и Δ_y . Тогда, очевидно, $\Delta_y = y\delta_y$, откуда видно, что при приближительно постоянном значении $\delta_y = 10^{-6} \dots 10^{-5}$ абсолютная погрешность Δ_y растет пропорционально y и, значит, быстрее, чем n .

— Почему?

— Да потому, что, как ты уже знаешь, при больших n $y \approx n \lg n$. Теперь смотри, что представляет собой относительная погрешность $n!$, которую мы обозначили δ_x . По определению $x = 10^y$, откуда

$$x + \Delta_x = 10^{(y + \Delta_y)} = 10^y 10^{\Delta_y},$$

$$\Delta_x = 10^y 10^{\Delta_y} - 10^y = (10^{\Delta_y} - 1) 10^y = (10^{\Delta_y} - 1)x,$$

и, наконец,

$$\delta_x = \Delta_x/x = 10^{\Delta_y} - 1. \quad (10.3)$$

Поскольку с увеличением n погрешность Δ_y довольно быстро возрастает, то погрешность δ_x возрастает еще быстрее. Поэтому при очень больших n (например, для рассмотренного нами примера $n = 10^{10}$), хотя $\lg(n!)$ вычисляется с небольшой относительной погрешностью — порядка 10^{-5} , о самой величине $n!$ можно судить лишь в очень расплывчатых рамках. Действительно, если бы записанное нами значение $y = 9,5657 \cdot 10^{10}$ для $\lg(10^{10}!)$ было абсолютно точным, то мы могли бы утверждать, что $x = (10^{10}!) = 10^{95\,657\,000\,000}$. Но если при вычислении y получилась

ничтожная ошибка и в действительности $y = 95\,657\,010\,000$ (т. е. относительная погрешность y чуть больше, чем $1 \cdot 10^{-7}$), то $x = (10^{10})! = 10^{95\,657\,010\,000}$. Это новое значение x в 10^{10000} раз больше предыдущего!!!!

Здесь я поставил четыре восклицательных знака от изумления, а не как символы факториала. Ведь даже 10^{100} — величина неимоверно большая. Другими словами, мы даже порядок величины $10^{10!}$ определяем неточно. Если можно так выразиться, мы можем судить только о «порядке порядка величины». В общем, при вычислении $n!$ по формуле Стирлинга относительная погрешность меньше 10^{-6} , если $8 < n < 50$, и 10^{-5} , если $5 < n < 1000$. При $n < 30$ основную роль играет методическая погрешность вычисления по формуле Стирлинга. Если же $n > 40$, то превалирует погрешность округления. При увеличении n до 10^5 относительная погрешность $n!$ возрастает до 10^{-2} , а при $n > 10^6$ можно определить только порядок величины $n!$

Тут я вспомнил, что папа обещал рассказать, как определяются факториалы дробных чисел.

— Хорошо, — сказал папа. — Это довольно интересная история, и произошла она в нашем городе.

— Как, в Ленинграде?

— Тогда он еще назывался Петербургом. Было это примерно 260 лет назад, в 1729 г. Такую задачу — распространение понятия факториала на любые числа — поставил перед собой Леонард Эйлер. Ему тогда было 22 года, и он начал работать в Петербургской Академии наук. Ему удалось построить функцию, которую он обозначил греческой буквой Г (гамма) и которую с тех пор и называют «гамма-функцией». Эйлер предложил несколько определений гамма-функции с помощью предельных переходов, а также в виде интегралов. Из них наиболее распространенным стало определение функции Г (x) для положительных x с помощью так называемого интеграла Эйлера второго рода:

$$\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt. \quad (10.4)$$

Если взять в качестве x целое число n , то этот интеграл вычисляется элементарно и равен $(n - 1)!$. Следовательно, для целых положительных чисел гамма-функция совпадает с факториалом, если только аргумент уменьшить на единицу.

Ты помнишь, что для факториала мы пользовались определением $n! = n(n-1)!$ Аналогично можно определить $\Gamma(n)$ функциональным уравнением $\Gamma(n) = (n-1)\Gamma(n-1)$, откуда также следует

$$\Gamma(n) = \Gamma(n+1)/n. \quad (10.5)$$

Эйлер доказал, что из его определений гамма-функции формула

$$\Gamma(x) = (x-1)\Gamma(x+1) = \Gamma(x+1)/x \quad (10.6)$$

справедлива и для дробных чисел. После этого уже можно было утверждать, что понятие факториала распространено и на дробные числа. Обрати внимание, что, хотя интеграл Эйлера определяет гамма-функцию только для положительных x , с помощью соотношений вида (10.6) можно найти значения гамма-функции и для отрицательных x . Ведь из (10.8) следует

$$\Gamma(x) = \frac{\Gamma(x+a)}{x(x+1)(x+2)\dots(x+a-1)}, \quad (10.7)$$

где a — любое целое число. Если, например, x — отрицательное число, то можно взять $a > |x|$, и тогда $x+a$ будет положительным числом, для которого мы в принципе можем найти $\Gamma(x+a)$ по интегралу Эйлера. А тогда из (10.7) можно найти и $\Gamma(x)$, хотя $x \leq 0$.

— А что если $x = 0$? — спросил я. — Ведь для него можно найти гамма-функцию из (10.6). Но тогда получается $\Gamma(0) = \Gamma(1)/0$, а значение $\Gamma(1)$, мы знаем, равно единице. Но делить на нуль в математике категорически запрещено.

— Да, — сказал папа. — Значение $\Gamma(0)$ или, что тоже самое, $(-1)!$ осталось неопределенным. Это же относится и ко всем целым отрицательным значениям x . Если ты присмотришься к формуле (10.7), то заметишь: когда x целые, причем $x < 0$, а $x+a > 0$, то в знаменателе этой формулы один из сомножителей будет равен нулю. Поэтому гамма-функция оказывается неопределенной для нуля и целых отрицательных чисел. Это вовсе не недостаток определения Эйлера. Таково свойство функции, развивающей понятие факториала. Она получила очень широкое применение в различных областях математики и породила другие функции. Но мы так далеко углубляться не будем. Хочу обратить твое внимание на очень простой

ход функции $1/\Gamma(x)$ — обратной величины гамма-функции. Она оказалась определенной для всех отрицательных и положительных значений x , а также для нуля. При $x = 0$, так же, как и при целочисленных отрицательных x , $1/\Gamma(x)$ принимает нулевые значения. Функция эта непрерывная и очень гладкая, как говорят математики, аналитическая. Я советую тебе построить ее график. Для этого нужно вычислить ряд значений $\Gamma(x)$ и взять их обратные величины.

— А как мне считать гамма-функцию? По интегралу? Но я же не умею.

— Зачем по интегралу? Вспомни, вчера мы пользовались асимптотической формулой Стирлинга. Она пригодна не только для факториалов целых чисел, но и для гамма-функции любых положительных чисел x , и точность ее тем выше, чем больше x . Нужно только внести поправку в аргументе. Тогда она получит следующий вид:

$$\Gamma(y+1) \approx \sqrt{2\pi y} \exp[y(\ln y - 1) + 1/12y]. \quad (10.8)$$

Это равенство приближенное, и хорошую точность — порядка 10^{-6} — обеспечивает при $y > 9$. Если же мы хотим вычислить $\Gamma(x)$, когда $x < 9$ или тем более когда $x < 0$, то будем вычислять его косвенным образом. Для этого выберем некоторое значение y , такое, чтобы оно было достаточно большим и отличалось от x на целое число. Тогда, вычислив по (10.8) $\Gamma(y+1)$ и обозначив через a величину $y+1-x$, получим $\Gamma(x+a)$, а отсюда по формуле (10.7) найдем искомое значение $\Gamma(x)$. Ясно?

— Ясно, только не очень. Как мы будем выбирать значение y , которое должно быть одновременно и достаточно большим, и отличаться от x на целое число?

— А мы не будем сами выбирать. Предоставим это ПМК, а он уже не ошибется, если, конечно, мы его научим.

— А как его учить?

— Нужно ввести соответствующую программу. Давай прикинем подходящий алгоритм. Прежде всего, нужно рассмотреть случай, когда само значение x достаточно велико, так что можно принять $a = 0$. Тогда $y+1-x = 0$, т. е. $y+1 = x$, и значение $\Gamma(x)$ вычисляется прямо по формуле (10.8). Пусть это имеет место, когда $x > A$, где A — число, которое мы потом выберем из таких соображений, чтобы при $x \geq A$ можно было пользоваться непосредственно формулой Стирлинга, а при $x < A$

дополнять ее преобразованием (10.7). Поэтому в нашем алгоритме нужно прежде всего сравнить x с числом A . Учитывая возможности ПМК, мы сначала вычислим разность $A - x$, а затем будем ее сравнивать с нулем. Если эта разность отрицательна, то можно сразу вычислить $\ln \Gamma(x)$ по формуле (10.8), положив в ней $y = x - 1$. Если же эта разность положительна, то нужно к величине x прибавить такое целое число a , чтобы обеспечивалось выполнение условия $x + a = y + 1 \geq A$. Это можно выполнить, положив a равным ближайшему целому числу, превышающему разность $A - x$, при этом $x + a > x + (A - x) = A$.

Затем, как мы уже говорили, вычисляется $\Gamma(y + 1)$, где $y = x + a - 1$ и по формуле (10.7) переходим к $\Gamma(x)$. Теперь можно записать алгоритм:

```

алг ГАММА (вещ  $x$ ,  $\Gamma$ )
арг  $x$ 
рез  $\Gamma$ 

нач цел  $A, a$ , вещ  $d, y, S, k, m, n$ 
 $d := A - x$ 
если  $d < 0$ 
    то  $y := x - 1; k := 1; a := 0$ 
    иначе  $a := E[d + 1]; y := a + x - 1; k := y$ 
        пока  $a \geq 1$ 
            нц
             $a := a - 1; k := (a + x - 1) k$ 
        кц
    все
 $m := y (\ln y - 1) + 1/12y; n := \sqrt{2\pi y}$ 
 $S := n e^m; 1/\Gamma := k/S$ 
кон

```

Начало алгоритма, вероятно, вопросов не вызовет. Об этом мы уже говорили. Если разность $A - x = d$ оказалась отрицательной, то мы можем непосредственно воспользоваться формулой Стирлинга (10.8), положив $x =$

$= y + 1$. Если же $d \geq 0$, т. е. $x \leq A$, то определяется число a , равное $E[d + 1]$, т. е. целой части от $d + 1$. Это значит, что a будет, во-первых, целым числом, во-вторых, больше $A - x$, и, наконец, не больше $A - x + 1$.

— Понятно. Я, конечно, могу легко определить в уме ближайшее целое число, большее d . Но как это сделает ПМК? У него ведь нет таких операций.

— Ты в этом уверен?

— Во всяком случае, я с такой операцией не встречался.

— С точно такой не встречался, но очень похожую ты знаешь. Я тебе подскажу. Предположим, что ты ввел число $d + 1$ в Р0. Теперь подумай, что делать дальше.

— Кажется, догадался. Применяю команду L0, которая реагирует на целую часть содержимого регистра. Теперь из $d + 1$ будет удалена дробная часть, а оставшаяся целая часть уменьшится на единицу. Так мы получим целую часть от d . Но нам нужна целая часть от $d + 1$.

— Нет ничего проще. Ввести в Р0 не $d + 1$, а $d + 2$. Остальная часть алгоритма, по-видимому, понятна. Через k я обозначил знаменатель в (10.7). При $d < 0$, когда мы к формуле (10.7) не прибегаем, и $a = 0$ мы просто положим $k = 1$. Кстати, при вычислении k рекомендую также воспользоваться счетчиком и хранить k в стеке. Да не забудь еще при составлении программы, что мы хотим построить график не для $\Gamma(x)$, а для обратной величины $1/\Gamma(x)$.

При активной помощи папы я составил такую программу, положив $A = 9$.

Программа 16. Вычисление величины, обратной гамма-функции $1/\Gamma(x)$

П1	1	—	П3	П2	9	XY	—	$x \geq 0$	48
2	+	П0	L0	15	ИП0	ИП3	+	П2	1
ИП0	ИП3	+	\times	L0	20	ИП2	\uparrow	ln	1
—	\times	ИП2	1	2	\times	$1/x$	$+$	e^x	2
π	\times	ИП2	\times	γ	\times	\div	С/П	1	БП
26									

Инструкция. $x = RX$ В/О С/П RX = $1/\Gamma(x)$.

Примеры. $1/\Gamma(-2,5) = -1,0578521$ (истинное значение — 1,0578555), время вычисления — около 50 с;

$1/\Gamma(11) = 2,7557251 \cdot 10^{-7}$ (истинное значение $2,7557319 \times 10^{-7}$), время вычисления — около 15 с.

Здесь пришлось дать два тестовых примера, так как программа имеет ветвление. Если сделать ошибку при ее вводе, то она может исказить результаты в одной ветви (например, при $x > A$) и не окажет влияния в другой (при $x < A$). Поэтому для проверки правильности ввода программы необходимо испытать оба случая.

Истинные значения $\Gamma(x)$ папа взял из таблиц. Относительная погрешность наших вычислений во всех проверенных случаях меньше 10^{-5} .

Затем я просчитал по программе 16 значения $1/\Gamma(x)$ на интервале $(-4, +4)$ (табл. 10.1) и построил график этой функции (рис. 10.1).

Таблица 10.1

x	$1/\Gamma(x)$	x	$1/\Gamma(x)$	x	$1/\Gamma(x)$
-4,0	0,00000	-1,2	0,206145	1,6	1,11917
-3,8	3,33742	-1,0	0,000000	1,8	1,07367
-3,6	4,05092	-0,8	-0,174260	2,0	1,00000
-3,4	3,06850	-0,6	-0,270494	2,2	0,90760
-3,2	1,45126	-0,4	-0,268604	2,4	0,80504
-3,0	0,00000	-1,2	-0,171787	2,6	0,69948
-2,8	-0,878268	0	0,000000	2,8	0,59648
-2,6	-1,12526	0,2	0,217824	3,0	0,500000
-2,4	-0,902429	0,4	0,450825	3,2	0,412546
-2,2	-0,453518	0,6	0,671503	3,4	0,335434
-2,0	0,000000	0,8	0,858935	3,6	0,269031
-1,8	0,313667	1,0	1,000000	3,8	0,213030
-1,6	0,432790	1,2	1,08912	4,0	0,166666
-1,4	0,376041	1,4	1,12706		

Пока ПМК вычислял каждую точку, я записывал и наносил на график предыдущую. Все вместе заняло примерно 40 мин. Я с гордостью показал свои результаты маме.

— Подумаешь! — сказала она. — ЭВМ за несколько секунд даст распечатку таблицы, а графопостроитель сам нанесет эту кривую.

Тут вмешался папа.

— Давай, — сказал он маме, — устроим соревнование. Выбери любую специальную функцию, выраженную с

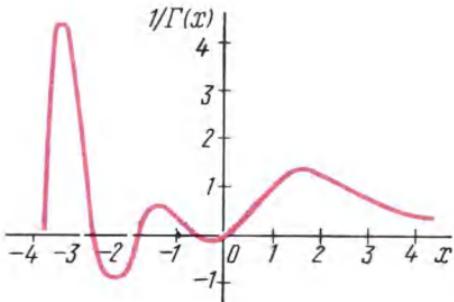


Рис. 10.1

помощью ряда, и построй график на твоей ЭВМ, а я построю с помощью ПМК. Посмотрим, кто раньше.

— Ты хитрец, — сказала мама. — У меня только завтра после обеда будет машинное время, а ты за сегодняшний вечер справишься.

— То-то и оно. ПМК хорош тем, что всегда при тебе.

11

ПРОГРАММА В ПРОГРАММЕ

— Сегодня я, пожалуй, не буду заниматься ПМК, — сказал я папе. — По московской программе телевидения будут транслировать матч на кубок и играет моя любимая команда.

— Хорошо, — ответил папа, — ты ведь занимаешься на добровольных началах. Я охотно вместе с тобой посмотрю матч. Только я что-то не вижу его в программе ленинградского телеканала.

— Сегодня утром, — уверенно сказал я, — по радиопрограмме «Маяк» объявили, что в 19.15 будет прямая трансляция второго тайма этого матча по московской программе телевидения.

— Ах, вот оно что, — улыбнулся пapa. — Тогда, чтобы посмотреть эту передачу, тебе придется на реактивном самолете быстро слетать в Москву. У нас ведь московская программа не принимается.

— Как же так? — воскликнул я. — Ведь мы же принимаем 1-ю и 2-ю общесоюзные программы, которые тоже идут из Москвы!

— Да, — сказал пapa, — но к нам они приходят не в виде радиоволн, распространяющихся от антennы с Останкинской башни, а транслируются по коаксиальному кабелю на Ленинградский телецентр и уж оттуда излучаются для ленинградских телезрителей. В другие города эти программы также передаются либо по кабелю, либо по радиорелейной линии, либо через ретранслятор, находящийся на спутнике связи.

— Почему же так?

— Да очень просто. Современное телевизионное вещание ведется в диапазоне метровых радиоволн, которые земную поверхность почти не огибают и от ионосферы не отражаются. Более длинные, декаметровые волны, которые хорошо отражаются от ионосферы и поэтому обеспечивают связь между весьма отдаленными точками земного шара, не могут передать всю информацию, которая требуется для телевизионной программы. Поэтому хороший прием телевидения возможен только при прямой видимости между передающей и приемной антennами, т. е. в пределах горизонта. За этими пределами еще можно вести прием на небольшом расстоянии благодаря огибанию радиоволнами земной поверхности (дифракции), а иногда рассеянию волн в тропосфере, но с увеличением расстояния прием быстро ухудшается и, наконец, совсем пропадает.

Вспомни, недавно ты составил программу для определения дальности прямой видимости и даже подсчитывал ее для Останкинской башни. Сколько у тебя получилось?

Я перерыл свои записи и нашел под номером 3 результаты, полученные ровно неделю тому назад. Оказывается, что расстояние прямой видимости с высоты 500 м (приблизительно высота Останкинской антennы) составляет около 80 км. Неужели только в этом радиусе москвичи могут смотреть свои передачи? Но если отъехать немного подальше, то уже телепередачу не посмотришь?

— Это было бы верно, если бы приемная антenna находилась непосредственно у земли. Но ты ведь заметил, в городе телевизионные антennы ставят обычно на крышах, а в пригородах стараются поднять их на высокие шесты.

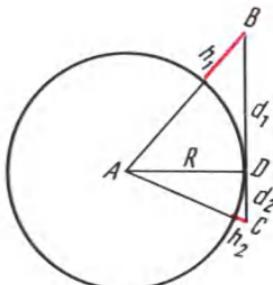


Рис. 11.1

телевидения как $d_1 + d_2$ в соответствии с нашим рисунком. Составь-ка программу для ее вычисления.

— Но ведь такая программа у нас уже есть. Это наша старая программа 3, всего на 8 шагов. Я ввожу в нее сначала высоту передающей антенны h_1 и нахожу d_1 , затем ввожу высоту приемной антенны h_2 и нахожу d_2 . Остается только сложить оба результата.

— Все это верно, — сказал пapa. — Но складывать ты будешь как, в уме? Дополни программу так, чтобы она сама все сделала после того, как ты введешь h_1 и h_2 . Например, можно предложить такую инструкцию: $R = P1$, $h_1 = PY$, $h_2 = PX$ В/О С/П $PX = d_1 + d_2$. Сначала ты вводишь радиус Земли R в $P1$, набираешь значение h_1 , нажимаешь клавишу \uparrow для перевода h_1 в PY , затем набираешь h_2 и запускаешь ПМК клавишами В/О С/П, после чего читаешь на индикаторе ответ. Хорошо бы еще предусмотреть возможность не вводить каждый раз h_1 : иногда нужно при определенной высоте передающей антенны рассмотреть влияние подъема приемной антенны на дальность телевизионного приема. Кроме того, я рекомендую использовать в качестве основы не программу 3, а немного более точную программу 4.

Я нашел эту программу и соответствующий алгоритм ГОРИЗОНТ. Очевидно, значения h_1 и h_2 нужно записать в какие-то регистры памяти, затем проделать операции алгоритма ГОРИЗОНТ с h_1 и записать результат в какой-то регистр. Потом проделать то же самое с h_2 и результат сложить с первым. В общем получается такой алгоритм:

Ведь к дальности d_1 , обеспечивающей высотой передающей антенны, нужно еще прибавить дальность горизонта приемной антенны d_2 . Вот, посмотри на схему (рис. 11.1). И тут подъем приемной антенны на какие-нибудь 10 м дает существенное увеличение дальности. Кроме того, радиоволны частично огибают земную поверхность, что еще увеличивает зону телевизионного вещания. Но не будем сейчас учитывать этого огибания и вычислим радиус зоны уверенного приема

```

алг ТЕЛЕВИДЕНИЕ (вещ R, h1, h2, D)
    арг h1, h2
    рез D
    нач вещ x, d1, d2
        x := h1
        ГОРИЗОНТ (R, x, d)
        d1 := d; x := h2
        ГОРИЗОНТ (R, x, d)
        d2 := d; D := d1 + d2
    кон

```

В этом алгоритме вспомогательный алгоритм ГОРИЗОНТ входит два раза. Это значит, что в моей программе два раза придется выполнять команды программы 4. Но не выписывать же ее дважды? И тем более стоит ли вводить в общую программу два совершенно одинаковых куска? Вероятно, есть способ более сокращенного построения такой программы? Я стал припоминать, что читал в руководстве. Ну, конечно же! Для этого существуют подпрограммы. Я поделился этой мыслью с папой.

— Молодец, — сказал он. — Именно это я и хотел от тебя услышать. Возьми руководство, освежи в памяти то, что там написано о подпрограммах, и попробуй применить это на деле.

Я уселся вновь за руководство и прочитал, что *подпрограммой* называется часть программы, к которой можно обращаться несколько раз. Обращение к подпрограмме осуществляется командой ПП, за которой следует двузначный адрес шага, с которого начинается подпрограмма. В конце подпрограммы ставится команда В/О, которая в данном случае означает «Возврат из подпрограммы». Тогда программа возвращается на тот шаг, который следует за адресом команды ПП. Таким образом, если в разных местах основной программы мы применим переход к подпрограмме, то возврат из нее будет осуществляться каждый раз в разные места, в зависимости от того, с какого места осуществлено обращение к подпрограмме. Это, конечно, очень удобно во всех случаях, когда какую-то последовательность операций нужно повторить в разных местах программы. В одной программе может быть несколько подпрограмм. Более того, внутри подпрограммы может быть

другая подпрограмма, внутри второй — третья, и так далее до пяти.

Изучив все это, я решил составить программу для нашей задачи с привлечением подпрограммы, которая вычисляет величину d . Вот что получилось.

Программа 17. Вычисление дальности телевизионного вещания

```
P2 XY ПП 11 П4 ИП2 ПП 11 ИП4 +
С/П ↑ ИП1 × 2 × XY x2 + √
B/O
```

Инструкция. ($R = 6\ 366\ 127,8 = P1$) $h_1 = PY$, $h_2 = PX$ B/O С/П $PX = D$.

Примеры. При $h_1 = 500$ м, $h_2 = 10$ м $D = 91073,318$ м; при $h_1 = 500$ м, $h_2 = 15$ м $D = 93609,282$ м. Время каждого вычисления — около 12 с.

Смысл этой программы прост. Я ввожу h_1 в PY и h_2 в PX , сразу же 1-й шаг программы переводит h_2 в $P2$, а 2-й шаг h_1 в PX . Затем подпрограмма вычисляет d_1 , которое записывается в $P4$, после чего из $P2$ вызывается h_2 и подается снова на подпрограмму. Подпрограмма вычисляет d_2 , которое затем складывается с d_1 , и сумма D выдается на индикатор. Что же касается подпрограммы, то она в точности совпадает с программой 4.

Я был доволен и хотел уже эту программу показать родителям, но вдруг вспомнил. Папа учил меня никогда не успокаиваться на достигнутом, а пытаться улучшить программу.

Прежде всего, нужно посмотреть, нет ли в программе чего-нибудь лишнего. Я легко обнаружил, что 1-й шаг (P2) лишний. Дело в том, что величина h_1 сохраняется в операционном стеке и к моменту, когда h_2 нужно ввести в PX , она находится в PY и легко может быть выведена. Действительно, к моменту перехода к подпрограмме (шаг 03) h_2 находится в PY . Затем в течение подпрограммы h_2 перемещается по такому маршруту:

Шаг	11	12	13	14	15	16	17	18	19	20
Положение h_2	Z	T	Z	T	Z	Z	Z	Y	Y	Y

Поэтому 1-й шаг можно исключить, заменив на шаге 05 команду ИП2 на ХY. Тогда программа начнется с ХY. Но это ведь нелепо. Не проще ли убрать и 2-й шаг, а в инструкции указать, что h_2 вводится в РY, а h_1 — в РХ? Таким образом, можно сократить программу на два шага.

А теперь нужно подумать, нельзя ли ее сделать более удобной? И тут я вспомнил, что папа советовал предусмотреть возможность вводить только h_2 в тех случаях, когда h_1 остается прежним. Что ж, это можно сделать. Значение d_1 , вычисленное по h_1 , будет оставаться в Р4, и будем вычислять d_2 по разным h_2 . Теперь программа будет такой:

```
ПП 11 П4 ХY ПП 11 ИП4 + С/П БП  
04 ↑ ИП1 × 2 × ХY x2 + √  
В/О
```

Инструкция. ($R = 6366127,8 = P1$) $h_2 = PY$,
 $h_1 = PX$ В/О С/П РХ = D, $h_2 = PX$ С/П РХ = D, ...
Для вычисления при новых значениях h_1 и h_2 они вводятся так: $h_2 \uparrow h_1$ В/О С/П. Если же h_1 сохраняет старое значение, то вводится h_2 С/П (без В/О).

Примеры. Те же, что и для программы 17. Время вычисления при новом значении h_1 — около 12 с, при старом значении h_1 — около 7 с.

Длина программы осталась той же — 21 шаг, но она стала удобнее. Я пошел к папе и показал ему программу. А папа вдруг спросил:

- Ты эту программу опробовал?
- Ну, конечно!
- И она работает?
- Безусловно! Вот примеры, которые я вычислил.

Они соответствуют тем числам, которые ты называл.

— Ну, тогда тебе повезло, с чем тебя и поздравляю. Дело в том, что многие экземпляры ПМК «Электроника Б3-34» имеют капризный характер при возврате из подпрограммы. Если команда В/О в конце подпрограммы следует за командами Ig, In, X^Y, e^x, √, то она срабатывает неправильно, и переход осуществляется не на тот адрес, который нужен. Но твой ПМК, к счастью, не проявляет такого каприза, по крайней мере к √.

- А что же делать, если такой каприз есть?

— Приходится прибавлять лишний шаг: перед командой В/О включить «пустышку» — команду НОП (Нет

операции). Она набирается последовательностью префиксной клавиши К и клавиши 0, под которой черным написано НОП. Если это сделать, то в твоей программе 17 будет 22 шага, а не 21. Такая программа с «пустышкой» будет работать на любом исправном экземпляре ПМК БЭ-34 или его аналогов, т. е. на прошедшем тест, приложенный к заводскому руководству.

Между прочим, капризов у ПМК довольно много, в руководствах о них не пишут, но знать их надо. Если не забывать о них, то никакой опасности они не представляют. Я сейчас расскажу тебе еще об одном.

Команда X^Y иногда выполняется неправильно, причем это зависит от того, какое число находилось в РХ тогда, когда вводилось значение Y . Здесь речь идет не о том, что обычная погрешность становится очень большой. Об этом мы уже говорили. А дело в том, что полученный результат чрезвычайно далек от истинного. Так, например, если в РХ находится число, в восьмом разряде которого цифра 5 или 7, а после этого ты захочешь вычислить 5^3 и введешь такие команды $3 \uparrow 5 X^Y$, то вместо истинного значения 125 ты получишь число, близкое к 8, т. е. 2^3 . Такие «фокусы», связанные с командой X^Y , к сожалению, наблюдались у многих экземпляров БЭ-34, с которыми мне приходилось встречаться.

Для борьбы с этим капризом лучше всего вообще не применять команду X^Y , тем более, что даже в нормальных случаях она дает большие погрешности и занимает много времени. Лучше вместо нее использовать несколько шагов. Например, для вычисления x^3 (здесь я буду считать всюду, что x находится в РХ) лучше воспользоваться командами $\uparrow x^2 \times$, для вычисления x^4 — командами $x^2 x^2$, а для вычисления x^n — командами $\uparrow x^2 x^2 \times$. Для вычисления x^n , где n — целое число, можно предложить n -кратное циклическое перемножение, как мы делали в программе 11. Для вычисления x^y , где y — не целое число, лучше всего воспользоваться формулой

$$x^y = e^{y \ln x} \text{ или } x^y = 10^{y \lg x}. \quad (11.1)$$

О «капризе», связанном с использованием команды $/—/$ (Перемена знака) мы уже говорили (см. § 8).

Я хотел бы дать тебе еще одну поучительную задачу. В чем ее поучительность, ты увидишь, когда составишь

программу. Кстати, ты что-нибудь слышал о комбинаторике?

— Кое-что. Это что-то очень плохое, вроде мошенничества.

— Ну и познания у тебя! Откуда ты это взял?

— А как же? Ведь великим комбинатором называли Остапа Бендера.

— Это совсем другое дело. Комбинаторика — это раздел математики, изучающий свойства размещений предметов. В частности, в комбинаторике родился и факториал, которому мы посвятили много времени. Ты помнишь, что это количество способов, которыми можно расставить n предметов по порядку. Между прочим, когда-то основы комбинаторики входили в школьную программу.

Я сейчас познакомлю тебя с одним из важнейших понятий комбинаторики. Пусть ты имеешь n каких-то предметов, например n учеников в классе. Тебе нужно отобрать из них m ($m < n$), например направить их на какое-то мероприятие. Сколькими способами это можно сделать? Число таких способов называется числом *сочетаний из n по m* и обозначается C_n^m или $\binom{n}{m}$. В комбинаторике очень просто доказывается равенство

$$C_n^m = \frac{n!}{m!(n-m)!}. \quad (11.2)$$

Учитывая выражение факториала как произведение последовательных натуральных чисел и произведя сокращения, можно выражение (11.2) переписать так:

$$C_n^m = \frac{n(n-1)(n-2)\dots(n-m+1)}{1 \cdot 2 \cdot 3 \dots m}. \quad (11.3)$$

Эти величины C_n^m часто называют также биномиальными коэффициентами, они играют довольно значительную роль в математике.

Так вот тебе простое задание. Составь хорошую программу для вычисления C_n^m по заданным n и m .

— Ясно, — ответил я. — Нет ничего проще. Вот только найду программу для факториалов, она будет здесь подпрограммой. Я просмотрел свои записи, нашел программу 12 и записал

Программа 18. Вычисление биномиальных коэффициентов (мой вариант)

```
П1 XY П2 ПП 21 П3 ИП1 ПП 21 П4  
ИП2 ИП1 — ПП 21 ИП4 × ИП3 XY ÷  
С/П П0 1 ИП0 × Л0 23 В/О
```

Инструкция. $n = PY$, $m = RX$ В/О С/П
 $PX = C_n^m$.

Пример. $C_{20}^5 = 15\ 504,001$. Время вычисления 70 с.

— Вот, папа, я использовал три раза подпрограмму для вычисления факториала, записал результаты в регистры Р3 и Р4, третий результат записывать не стал, а сразу подставил в формулу (11.2).

— Я и думал, что ты так построишь программу. А как ты оцениваешь точность своего результата?

— Конечно, — сказал я, — число сочетаний — это по идее целое число. Так что, по-видимому, правильный ответ $C_{20}^5 = 15\ 504$, а дополнительная одна тысячная — это погрешность в восьмом знаке. Здесь относительная погрешность меньше $1 \cdot 10^{-7}$ — это ведь совсем неплохо.

— Да, — подтвердил папа, — неплохо. А попробуй-ка вычислить C_{71}^2 .

— Пожалуйста!

Я ввел 71 ↑ 2 и запустил ПМК. Примерно спустя 2 мин на индикаторе высветилось ЕГГОГ.

— Значит, это число очень большое, больше 10^{99} .

— Ты уверен? — спросил папа. — Давай-ка я прикину на бумаге. По формуле (11.3) $C_{71}^2 = 71 \cdot 70 / (1 \cdot 2) = 2485$. Как видишь, даже и не приближается к 10^{99} .

— Почему же у меня получилась чепуха?

— Потому что ты выбрал неудачный алгоритм. Впрочем, аналогичные программы рекомендуют некоторые солидные авторы. Смотри, ты начинаешь с вычисления $n!$ Но если у тебя $n \geq 70$, то, поскольку $70! > 10^{99}$, произойдет переполнение разрядной сетки и отказ микрокалькулятора, хотя сама величина C_n^m может быть совсем небольшой. Но и кроме этого у твоего алгоритма есть недостаток. Тебе приходится при вычислении $n!$ умножать такие же числа, на которые потом придется делить. В общем тебе приходится выполнить $2n + 1$ умножение и одно деление. Лучше исходить из формулы (11.3). Здесь в чис-

лителе и знаменателе по m сомножителей, так что приходится делать $2m$ умножений и одно деление.

Еще удобнее организовать вычисление по формуле (11.3) следующим образом:

$$C_n^m = n : m \times (n-1) : (m-1) \dots \times (n-m+1) : 1, \quad (11.4)$$

используя $m - 1$ умножение и m делений. Этот алгоритм реализуется следующей программой.

Программа 19. Вычисление биномиальных коэффициентов C_n^m (папин вариант)

П1 x≠0 15 XY П0 1 ИП0 × ИП1 ÷

L0 12 L1 06 С/П 1 С/П

Инструкция. $n = PY$, $m = RX$ В/О С/П
 $PX = C_n^m$.

Примеры. $C_{20}^5 = 15\ 504$, время вычисления 16 с; $C_{20}^{10} = 184\ 755,96$ (истинное значение 184 756), время вычисления 30 с; $C_{71}^2 = 2485$, время вычисления 7 с (эти примеры я тут же вычислил).

— Видишь, — продолжал папа, — насколько эта программа лучше твоей. Она считает быстрее, точнее и в ней не происходит переполнения разрядной сетки, если вычисляемое значение C_n^m не приближается к 10^{99} .

Так вот на этом примере ты видишь, что не всегда следует применять подпрограмму, даже если это решение кажется весьма естественным. Никогда не мешает подумать о других вариантах.

Конечно, я не говорю, что программа 19 идеальная. Она не очень красива — ее портят шаги 01, 02, 15 и 16, без которых на первый взгляд можно обойтись. Их пришлось ввести, чтобы программа допускала случай $m = 0$, когда по определению $C_n^0 = 1$. Если бы этих команд не было, ПМК при $m = 0$ зацикливался бы. Кроме того, программа работает правильно только при условии $n \geq m$. От этого недостатка можно избавиться, но для этого придется применить клавишу К, о которой мы еще не говорили. Но сейчас уже слишком поздно, чтобы этот разговор начинать, к тому же сейчас по телевидению начнется футбольный матч, правда, не тот, который ты хотел посмотреть, нарушив законы распространения радиоволн, а наш, ленинградский, в котором играет «Зенит».

12

ЕЩЕ О ТЕЛЕВИДЕНИИ

Сегодня папа решил продвинуть мое калькуляторное образование.

— Слушай, старик (папа считает, что это самое современное обращение к людям моложе его), вчера ты решил, какую дальность передачи в пределах прямой видимости можно обеспечить, если высота передающей антенны h_1 , а приемной h_2 . Но в жизни мы обычно встречаемся с другой задачей, можно сказать, обратной. Например, гражданин Иванов живет за городом на расстоянии D от телеконцерна, высота башни которого h_1 известна. На какую высоту h_2 он должен поднять свою приемную антенну, чтобы обеспечить прямую видимость? Другими словами, заданы h_1 , R и D и нужно определить h_2 . Как ты будешь это решать?

— Очень просто. У меня есть программа 17 с подпрограммой, которая по h_1 и h_2 вычисляет D . Ввожу известные R и h_1 и какое-либо разумное h_2 . Считаю и нахожу D . Если оно меньше заданного, я увеличиваю R_2 и считаю снова. И так буду увеличивать или уменьшать h_2 , пока не получу заданное D .

— Правильно, — сказал папа. — Такой способ последовательных приближений для решения уравнений известен и нередко используется. Но согласись, что это далеко не быстрый способ. Придумай что-нибудь лучше и короче.

Я задумался и стал рассматривать вчерашнюю схему (см. рис. 11.1).

— Вот как это можно сделать. Имеется треугольник ABC , в котором известны две стороны $|AB| = R + h_1$ и $|BC| = D$. Нужно найти третью сторону $|AC| = R + h_2$. Для этого нужно знать еще угол B между двумя заданными сторонами. Его можно определить, так как известна высота из вершины A , равная R . Отсюда

$$\sin B = R / (R + h_1) \quad (12.1)$$

и по теореме косинусов

$$|AC| = R + h_2 = \sqrt{(R + h_1)^2 + D^2 - 2(R + h_1)D \cos B}. \quad (12.2)$$

— Прекрасно, — сказал папа. — Эти две формулы по существу уже алгоритм. Теперь составь программу. Только учи все, чему ты за последние дни научился. В частности, сделай программу поудобней.

Я немного посидел и представил следующее.

Программа 20. Вычисление высоты h_2 приемной антенны для обеспечения приема телевизионной передачи на расстоянии D от телебашни высотой h_1

```
ПД 2      ВП 7      л  ÷  П1 ↑  ИП2 +
÷ arcsin cos ИПД × 2      × ИП1 ИП2 +
× Вх  x² XY — ИПД x² + √ ИП1
— С/П ПД ИП1 БП 07
```

Инструкция. ($h_1 = Р2$) $D = РХ$ (В/О) С/П
 $РХ = h_2$. Все величины в метрах. Переключатель Р — Г
в любом положении.

Примеры. Для $h_1 = 300$ при $D = 70\ 000$ $h_2 = 5$; при
 $D = 80\ 000$ $h_2 = 25,7$. Время первого вычисления — около
18 с, последующих (при том же значении h_1 , но других
 D) 14 с.

Папа остался очень доволен:

— Эта программа уже напоминает работу вдумчивого
программиста. Очень хорошо, что ты отказался от ручного
ввода радиуса Земли, а ввел его в программу и к тому же
не поразрядным набором цифр, что заняло бы 8 шагов, а
путем деления половины длины окружности меридиана на
я, всего за 5 шагов (от 01 до 05). К тому же такой ввод более
надежен. Хорошо также, что для повторных вычислений
при том же h_1 и новом значении D ты предусмотрел
сокращенное начало вычислений. Ты правильно исполь-
зовал команду Вх для того, чтобы вызвать из РХ1 сумму
 $R + h_1$, вместо того чтобы вычислять ее заново.

Сократить твою программу на один шаг можно без
особого труда. Если после вычисления $R + h_1$ (шаг 09) за-
писать сумму, например, в РЗ, то потом вместо шагов 17, 18
и 19 можно будет ограничиться одной командой ИП3.
Впрочем, это сокращение на один шаг ничего существен-
ного не даст. Опытные программисты обратили бы внимание
в первую очередь на возможность ускорить вычисле-
ние.

— А как это сделать?

— Очень просто. Нужно устраниить «тихоходные» команды \arcsin и \cos и вместо них вычислить $\cos B$ непосредственно из $\sin B$ по простой формуле

$$\cos B = \sqrt{1 - \sin^2 B} = \sqrt{1 - R^2/(R+h_1)^2}. \quad (12.3)$$

В результате после простых преобразований получим

$$h_2 = \sqrt{(R+h_1)^2 + D^2 - 2D\sqrt{2Rh_1 + h_1^2}} - R. \quad (12.4)$$

Давай составим программу по этой формуле и сравним число шагов и скорость вычисления с программой 20. Вот что у нас получилось.

Программа 21. То же, что и программа 20 (с увеличенной скоростью)

```
ПД 2 ВП 7 π ÷ П1 ИП2 + x2
ИПД x2 + ИП1 ИП2 × 2 × ИП2 x2
+ √ ИПД × 2 × - √ ИП1 -
С/П ПД ИП1 БП 07
```

Инструкция. Та же, что и к программе 20.

Примеры. Для $h_1 = 300$ при $D = 70000$ $h_2 = 5,2$; при $D = 80000$ $h_2 = 25,9$. Время первого вычисления — около 13 с, последующих (при том же h_1) — около 11 с.

Эта программа оказалась на два шага короче предыдущей и примерно на 30 % более «быстроходной».

— А скажи-ка, папа, — не без ехидства спросил я, — почему результаты по последним двум программам не совпадают? Какая из них врет?

— Ну, если это называть враньем, то подвирают они обе. И это вполне естественно — в обеих ~~программах~~ существует вычитание близких чисел, и, по-видимому, избежать этого нельзя или во всяком случае трудно. Мы ведь вычисляем в конечном счете $R + h_2$, а затем из него вычитаем R . Поскольку R больше h_2 примерно в сотни тысяч раз, то во столько же раз возрастает относительная погрешность: если для $R + h_2$ она составляла $10^{-8} \dots 10^{-6}$, то для h_2 будет $10^{-3} \dots 10^{-1}$. А на вопрос, какая из этих программ точнее, я отвечу, что точнее программа 21. В ней нет операций \arcsin и \cos , которые не только «тихоходны»,

но и дают большие погрешности, чем возведение в квадрат и извлечение корня.

Но все это полбеды. В данном случае погрешность в несколько процентов несущественна. Ведь никто не будет рассчитывать высоту приемной антенны с точностью до сантиметров. Значительно хуже другой недостаток обеих программ. Предположим, что ты живешь в Москве рядом с Останкинской башней, так что расстояние D у тебя всего лишь 100 м. Как ты думаешь, на какую высоту тебе нужно поднять приемную антенну?

— Ни на какую. Она может находиться хоть у самого телевизора. При таком расстоянии можно принимать без всякой антенны.

— Очень хорошо, что ты это знаешь. Но знает ли это твой ПМК?

— Сейчас проверим, — сказал я и ввел в ПМК $h_1 = 500$ м и $D = 100$ м. Каково было мое удивление, когда я получил ответ 498,7 м.

— Что же, приемную антенну нужно поднять до высоты самой Останкинской башни?

— Я этого не говорю, но это сказал твой ПМК.

— Постой, я еще кое-что попробую, — сказал я и стал вводить различные значения D .

При $D = 80\,000$ м я получил $h_2 = 0$. Это совершенно естественно: мы еще раньше подсчитали радиус прямой видимости с высоты 500 м, который равен 79,8 км. Но далее, когда я уменьшаю D , величина h_2 не остается нулем, а почему-то возрастает. Так, при $D = 70\,000$ м я получил $h_2 = 7,5$ м, при $D = 50\,000$ м — $h_2 = 69,7$ м, а при $D = 30\,000$ м — $h_2 = 194,7$ м.

— Что за чушь! — воскликнул я. — Почему программа 21, годная для больших расстояний, становится непригодной при малых?

— А ты вдумайся в суть формулировки задачи и поймешь, что иначе и быть не могло. И это в той же степени относится к программе 20. Все дело, как это часто бывает, в неполном совпадении физической задачи и ее математической модели. Наша физическая задача — найти высоту h_2 приемной антенны, находящейся на расстоянии D от передающей антенны, поднятой на высоту h_1 , при которой прямая, соединяющая обе антенны, не пересекает поверхности земли. Мы заменили эту задачу другой математической задачей, представляющей в некоторой степени модель задачи физической. В треугольнике ABC (см. рис. 11.1) сторона AB равна $R + h_1$, сторона BC равна D ,

а высота из вершины A равна R . Требуется найти разность между длиной стороны AC и высотой R .

Легко видеть, что эти задачи совершенно эквивалентны, если в треугольнике ABC угол C острый. Этот случай имеет место только тогда, когда $D > d_1$, где d_1 — длина касательной, проведенной из точки передающей антенны B сферы Земли. В этих условиях обе программы дают верные результаты (с точностью допустимых погрешностей). Но если $D < d_1$, то физическая и математическая задачи перестают быть эквивалентными. Если сохранить математическую формулировку и рассмотреть рис. 12.1, то легко понять, что с уменьшением расстояния D длина стороны AC увеличивается и в пределе, когда D стремится к нулю, стороны AB и AC должны совпасть. Но это уже ничего общего не имеет с нашей физической задачей. Для нее при $D < d_1$ прямая видимость между двумя антеннами обеспечивается без всякого подъема приемной, если, конечно, поверхность земли ровная и на ней нет загораживающих предметов.

А вообще запомни: микрокалькулятор, как и всякая ЭВМ, предназначен помочь голове, а не заменять ее.

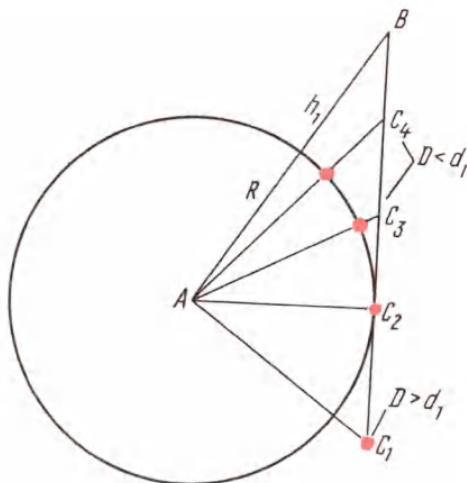


Рис. 12.1

13

ТРЕУГОЛЬНАЯ ФАНТАЗИЯ

— Ты заметил, — спросил папа за завтраком, — что задача, которой ты занимался, это решение треугольника. Тебе известны две стороны треугольника и высота, опущенная на одну из них. Требуется найти третью сторону треугольника. Решать треугольники приходится в разных областях науки и техники, но больше всего в геодезии и картографии. Кстати, когда определяли эталон метра и измеряли для этого длину парижского меридиана, то пользовались методом триангуляции. Это значит, что на земле установили ряд точек (тригонометрических пунктов), которые являются вершинами разных треугольников (рис. 13.1). Две из них A и B находятся на меридиане, и нужно очень точно измерить длину AB . Непосредственно это сделать невозможно. Во-первых, расстояние AB через сур вело — невозможно даже наметить прямую, соединяющую эти точки. Во-вторых, между A и B могут быть реки, горы и другие препятствия. Но зато можно с большой точностью измерить углы всех треугольников, и, кроме того, выбрать одну маленькую сторону одного из треугольников, например сторону CD , которую со всей возможной тщательностью измеряют. Так вот, по одной стороне и всем этим углам можно точно вычислить все стороны всех наших треугольников и подсчитать длину AB .

— Как же это делали, когда ни ЭВМ, ни ПМК не было?

— Очень просто. Считали на бумаге, пользуясь таблицами логарифмов. На эти вычисления уходило подчас больше времени, чем на сами измерения.

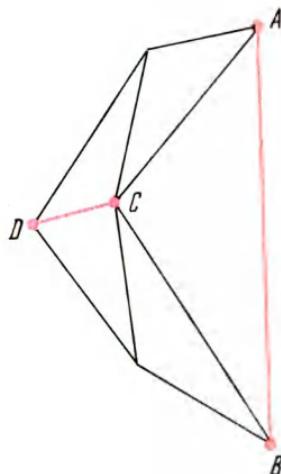


Рис. 13.1

— Но сейчас, вероятно, существуют программы, с помощью которых все это можно сделать быстро и точно?

— Конечно, такие программы есть. Вот в этом прекрасном сборнике прикладных программ (А. Н. Цветков и В. А. Епанечников [3]) приведены пять программ решения треугольников. К сожалению, однако, они содержат опечатки и не учитывают «капризы» ПМК Б3-34, о которых мы говорили вчера. Так, например, нахождение неизвестного угла (например, C) по двум известным (A и B) производится по элементарной формуле

$$C = \pi - (A + B) \text{ рад или } C^\circ = 180^\circ - (A^\circ + B^\circ). \quad (13.1)$$

Однако авторы вполне разумно решили составить такие программы, которые бы допускали измерения углов как в радианах, так и в градусах без всякой переделки программы одним переключателем Р — Г. Для этого они преобразовали формулы (13.1) так, чтобы они были одинаковыми и при радианах, и при градусах:

$$C = \arccos [-\cos (A + B)]. \quad (13.2)$$

Все это очень хорошо. Но в программах это получило такой вид:

ИПА ИПВ + cos /—/ arccos ПС

Операция смены знака /—/, как я тебе уже говорил, прекрасно работает при пошаговом прохождении программы, но в автоматическом режиме (в первых выпусках Б3-34) приводит к пропуску одного шага, и в результате истинного решения получить не удается. В данном случае я вижу только один выход — удлинить этот фрагмент на один шаг, добавив между командами cos и /—/ команду \uparrow . Кроме того, в программе II.4.2 [3] ошибочно переставлены команды в шагах 15 и 16, это, скорее всего, опечатка.

Я очень рекомендую тебе разобраться в этих программах. Они могут тебе пригодиться. В них применены общепринятые обозначения: a , b , c — стороны треугольника; A , B , C — соответственно противолежащие им углы; S — площадь треугольника. Как я уже говорил, расчет производится в градусах или радианах в зависимости от положения переключателя Р — Г.

Я стал рассматривать эти программы. Каждая из них посвящена своему варианту задачи. Одна из них решает треугольник по трем известным сторонам, следующая — по двум сторонам и углу между ними, затем по двум сторонам и углу, лежащему против одной из сторон (эта за-

дача не всегда имеет однозначное решение), затем по двум углам и стороне между ними и, наконец, по двум углам и стороне против одного из них. Помимо (13.2) использованы следующие расчетные формулы:

$$A = \arccos [(b^2 + c^2 - a^2) / (2bc)], \quad (13.3)$$

$$B = \arccos [(a^2 + c^2 - b^2) / (2ac)], \quad (13.4)$$

$$a = c \sin A / \sin C, \quad (13.5)$$

$$b = a \sin B / \sin A, \quad (13.6)$$

$$a = \sqrt{b^2 + c^2 - 2bc \cos A}, \quad (13.7)$$

$$S = 0,5ab \sin C. \quad (13.8)$$

Меня не удовлетворило, что для каждой задачи на решение треугольников требуется своя программа.

— Папа, — сказал я, — у меня идея. Нужно сделать единую универсальную программу для решения треугольников при задании любых трех его элементов.

— В чем же твоя идея?

— Сделать универсальную программу для решения треугольников.

— Прости, но это не идея, а пока только голубая мечта. Вот когда у тебя будут предложения, как такую программу составить, тогда можно будет говорить о какой-то идее.

— Но у меня кое-какие мысли есть. Нужно просмотреть существующие программы по разным типам задач на решение треугольников и отметить в них общие места. Из них можно сделать подпрограммы и использовать их в разных задачах.

— Это все? — спросил пapa. — Не думаю, что так тебе удалось бы все типы задач вместить в 98 шагов программной памяти наших трех типов ПМК. Нужно очень серьезно подумать о разных путях сокращения, а может быть, и пожертвовать некоторыми удобствами. Я бы посоветовал пойти так. Вводить данные об известных и неизвестных элементах треугольника вручную. Результаты не выводить на индикатор, а записывать в регистры, откуда их можно затем считывать после окончания вычисления. Кроме того, не пытайся охватить задачи для всех возможных вариантов решения треугольников. Ограничиться основными, которые имеют однозначные решения: по трем сторонам, по двум сторонам и углу между ними и по двум углам и стороне между ними.

— А знаешь, папа, относительно последнего варианта у меня есть такая мысль. Если известны два угла, то ничего не стоит узнать третий. Поэтому какая бы сторона не была известна, всегда можно эту задачу свести к случаю, когда известны оба прилегающие к этой стороне угла.

— Да, это тоже мысль. Теперь подумай, как ты будешь, вернее, как будет калькулятор различать, какие элементы треугольника известны, а какие нет.

— Я думаю так: мы вводим в соответствующие регистры те элементы, которые мы знаем, а места для неизвестных оставляем пустыми, чтобы ПМК их затем заполнил.

— Что значит пустые места? Это ведь регистры, в которые введены нули.

— А хоть бы и так! Ведь среди заданных элементов нулей не бывает. Иначе треугольник перестает быть треугольником.

— Это, пожалуй, верно. И еще я тебе посоветую — стандартизируй обозначения известных и неизвестных элементов. Например, так. Если известен хотя бы один угол, будем этот угол называть A и записывать в РА. Тогда, если ПМК обнаружит, что РА = 0, то это будет значить, что ни один угол не известен, и, следовательно, задача заключается в решении треугольника по трем сторонам. Таким же образом установим, какая сторона обязательно известна. Пусть это будет сторона c . Причем будем записывать элементы треугольника $a = P1$, $b = P2$, $c = P3$, $A = PA$, $B = PB$, $C = PC$.

— Хорошо, — сказал я. — Запишем в таблицу (табл. 13.1), какие данные нам известны при разных вариантах решения треугольников.

Таблица 13.1

Задача	Элементы	
	известные	неизвестные
1. По двум углам и стороне между ними	A, B, c	a, b, C, S
2. По двум углам и стороне против одного из них	A, C, c	a, b, B, S
3. По одному углу и двум смежным с ними сторонам	A, b, c	a, B, C, S
4. По трем сторонам	a, b, c	A, B, C, S

Теперь можно сформулировать хотя бы в словесной форме алгоритм различения типов задач:

1. Если $b = 0$ и $B \neq 0$, задача 1-го типа.
2. Если $b = 0$ и $B = 0$, задача 2-го типа.
3. Если $b \neq 0$ и $A \neq 0$, задача 3-го типа.
4. Если $A = 0$, задача 4-го типа.

Я стал думать о том, как построить общий алгоритм.

Прежде всего я заметил, что когда уже все углы и стороны найдены, можно найти площадь его по формуле (13.8), общей для всех типов задач. Для 4-го типа нужно вычислить три угла A , B , C . Угол A вычисляем по формуле (13.3), угол B — по формуле (13.4), а угол C — по формуле (13.2). Формула (13.2) может использоваться во всех типах задач, поэтому вычисление по ней оформим в виде подпрограммы. Формулы (13.3) и (13.4) практически одинаковы, но в них входят разные величины, что мешает их «превращению» в подпрограмму.

И тут меня озарила хорошая мысль:

— Папа, давай введем в операционный стек значения a , c и b , на что потребуется только три шага, а затем в подпрограмме из этих чисел вычислим значение A . Потом введем в операционный стек те же значения, но в другом порядке: b , c и a . Тогда та же подпрограмма вычислит величину B . Это, вероятно, решит проблему сокращения части программы для 4-го типа задач.

Папа с этим согласился и тут же заметил, что для задач 3-го типа нам нужно только вычислить значение a , что можно сделать по простой формуле (13.7), после чего все три стороны треугольника известны и можно вычислять углы так же, как в задаче 4-го типа, конечно, за исключением угла A , который уже известен.

Затем мы стали думать о задачах 1-го и 2-го типов. Это, по существу, одно и то же: в одном случае нужно найти угол C , а в другом — угол B , что делается в подпрограмме, работающей по формуле (13.2), а затем обе задачи сводятся к вычислению a и b по формулам синуса (13.5) и (13.6). Здесь можно применить ту же хитрость, что и с формулами (13.3) и (13.4), т. е. ввести используемые величины в стек, а операции в стеке оформить в виде подпрограммы.

— Я в этом не убежден, — возразил папа. — Это формулы довольно простые и могут быть вычислены за шесть шагов каждая. Если же ты захочешь их вычислять в подпрограмме, то на подпрограмму уйдет также шесть шагов, да еще два раза по три шага на ввод данных в операци-

онный стек. Кроме того, придется затратить еще два раза по два шага для обращения к подпрограмме. Экономнее будет вычислять по отдельности.

— Хорошо, так и сделаем. После того как все стороны и углы определены, нужно вычислить площадь треугольника. Это можно сделать для задач всех типов одинаково.

— Погоди, — сказал папа. — Ты сильно многое хочешь от одной программы. Пока попытаемся составить универсальную программу для вычисления сторон и углов треугольника и убедимся, что она вмещается в программную память нашего ПМК. А если при этом еще останутся свободные шаги, то посмотрим, не удастся ли нам вычислить и площадь, а может быть, и еще что-нибудь.

Я занялся составлением алгоритма, который получился таким:

```
алг ТРЕУГОЛЬНИК (вещ A, B, C, a, b, c, S)
    арг A, B, C, a, b, c
    рез A, B, C, a, b, c, S
    нач если b = 0
        то если B = 0
            то B := arccos [−cos (A + C)]
            иначе C := arccos [−cos (A + B)]
        все
        a := c sin A / sin C; b := a sin B / sin A
    иначе если A ≠ 0
        то a := √(b² + c² − 2bc cos A)
        иначе A := arccos [(b² + c² − a²) / 2bc]
    все
    B := arccos [(a² + c² − b²) / 2ac]
    C := arccos [−cos (A + B)]
    все
    S := ab sin C / 2
кон
```

В соответствии с этим алгоритмом мы составили универсальную программу решения треугольников. К сожалению, последняя операция — вычисление площади S треугольника — в эту программу не уместилась. Вот эта программа.

Программа 22. Универсальная для решения треугольников

```

ИП2 x=0 31 ИПВ x=0 26 ИПС ПП 68 ПВ
ИП3 ИПА sin × ИПС sin ÷ П1 ИПВ sin
× ИПА sin ÷ П2 С/П ПП 68 ПС БП
10 ИПА x=0 50 ИП1 ИП3 ИП2 ПП 76 ПА
ИП2 ИП3 ИП1 ПП 76 ПВ ПП 68 ПС С/П
ИП2 x2 ИП3 x2 + ИП2 ИП3 × 2 ×
ИПА cos × — √ П1 БП 40 ИПА +
cos ↑ /—/ arccos НОП В/О П4 x2 XY П5
x2 + XY x2 — ИП4 ИП5 × 2 ×
÷ БП 73

```

Инструкция. $a = P1$, $b = P2$, $c = P3$, $A = PA$, $B = PB$, $C = PC$ (все неизвестные величины заменяются нулями) В/О С/П $P1 = a$, $P2 = b$, $P3 = c$, $PA = A$, $PB = B$, $PC = C$.

При обозначении элементов треугольника соблюдать условия — из известных углов один обозначается A , из известных сторон одна обозначается c , угол A противоположен стороне a и т. д. Вычисления производятся в радианах или градусах в зависимости от положения переключателя Р — Г. При останове высвечивается C или b зависимости от типа задачи.

Примеры.

$$1. \ a = 14,87, \ b = 10,46, \ c = 5,43.$$

Ответ: $A = 2,3801$ рад = $136,37^\circ$, $B = 0,5068$ рад = $= 29,037^\circ$, $C = 0,2547$ рад = $14,594^\circ$.

2. $A = 1,4$ рад = $80,214^\circ$, $b = 1,1$, $c = 1$. Ответ:
 $a = 1,3550$, $B = 0,9273$ рад = $53,1288^\circ$, $C = 0,8143$ рад =
= $46,657^\circ$.

3. $A = 1$ рад = $57,296^\circ$, $B = 1,4$ рад = $80,214^\circ$,
 $c = 100$. Ответ: $C = 0,74159$ рад = $42,490^\circ$, $a = 124,58$,
 $b = 145,89$.

4. $A = 0,8$ рад = $45,837^\circ$, $C = 1,2$ рад = $68,755^\circ$,
 $c = 1$. Ответ: $B = 1,14159$ рад = $65,408^\circ$, $a = 0,76966$,
 $b = 0,97560$.

Результаты здесь округлены. Независимо от типа задачи время вычисления 30 ... 35 с.

— Итак, — сказал папа, — удалось уложить программу в 93 шага без вычисления площади S . В программной памяти остались свободными пять шагов. Но, к сожалению, этого недостаточно для вычисления площади треугольника. По самой простой формуле (13.8) для этого потребуется девять шагов:

ИП1 ИП2 × ИПС sin × 2 ÷ С/П

— Но ведь каждую программу можно сократить на шаг, — съязвил я.

— Конечно, — спокойно ответил папа. — Нашу программу 22 в принципе можно тоже сократить на один шаг. Но это проблемы не решает: для вычисления площади недостает еще трех шагов. Если бы твой ПМК не имел квиззов, то можно было бы изъять еще два шага (71 и 74), но все равно для вычисления площади этого недостаточно. Вот на этом примере ты видишь, как бывает, когда программа не вмещается в память ПМК. Конечно, если хочешь, попробуй поискать какие-либо хитрые методы сокращения. Но я боюсь, что на данном этапе ничего не получится.

— Но без вычисления площади треугольника решение какое-то неполноценное. Что же делать?

— Во-первых, не нужно стремиться в одной программе решить все на свете. В конце концов, площадь по формуле (13.8) можно вычислить и вручную. Впрочем, мне кажется, что я смогу применить еще одну хитрость и, сэкономив пару шагов, дополнить программу вычислением площади треугольника. Для этого тебе придется познакомиться поближе еще с некоторыми секретами. Но отложим это до завтра. Все-таки у меня сегодня выходной, да и много накопилось.

СЕКРЕТЫ КЛАВИШИ К

Сегодня мы ездили за город. И вдруг во время прогулки мама вспомнила, что прошло ровно две недели с тех пор, как я занялся ПМК.

— Как твои успехи? — спросила она. — Что ты уже умеешь?

— Да я уже почти все умею. Вчера я отгрохал такую программу, которая еле уместилась в 98 шагах памяти, и она умеет делать все с треугольниками. Правда, папа мне немного помогал, но я бы и сам справился, если бы подольше посидел.

— Ну и хвастунишка, — сказал услышавший это пapa. — Тебе еще многому надо учиться. Ты пока не умеешь решать уравнения, разлагать числа на простые множители, вычислять интегралы и многое другое, что тебе может понадобиться в школе. Я уж не говорю о более серьезных вещах. Ты даже еще не все клавиши своего ПМК знаешь.

— Клавиши я знаю все.

— В том числе и клавишу К?

— Да, и клавишу К. Я ее вчера использовал, когда вводил команду-пустышку НОП.

— Ну, это лишь ничтожная доля обязанностей клавиши К.

— И это я знаю. Ты ведь меня однажды упрекал, что я не заглядывал в руководство. Так вот сейчас я его внимательно изучил и могу прочесть доклад о всех возможностях клавиши К.

— Отлично, — сказал пapa. — Когда поедем домой, ты нам и расскажешь все, о чем узнал.

— Идет, — ответил я.

И вот, когда мы отправились в обратный путь, я начал свой рассказ.

Клавиша К является префиксной и обеспечивает возможность косвенной адресации обращения к памяти, к подпрограммам, а также условных и безусловных переходов. С помощью этой клавиши образуются следующие команды: КПN, КИПN, КППN, КБПN, Kx < 0N, Kx = 0N, Kx ≥ 0N и Kx ≠ 0N. Здесь везде N — регистр памяти от 0 до 9 и от А до Д, в котором хранится адрес. Так, например, если в программе имеется команда КП7, то это значит, что число, находящееся в РХ, должно

быть записано в тот регистр, номер которого хранится в Р7. Команда КИП7 означает, что в РХ должно поступить число из того регистра, номер которого записан в регистре 7. Команда КПП7 означает «Перейти к подпрограмме», которая начинается с шага, адрес которого записан в регистре 7. Команда Кх ≥ 0 7 значит «если условие $x \geq 0$ не выполнено, то перейти к шагу, адрес которого записан в регистре 7», и т. д. Другими словами, команды с клавишей К действуют так же, как и соответствующие команды без этой клавиши, например П, ИП, ПП, БП, х < 0 и т. д., но только адрес (номер регистра или шага) указывается не прямо после команды, а косвенно — через какой-то регистр N, где этот адрес записан. Это вроде как письма, направленные по чужому адресу, для передачи тому, кому они предназначены.

Но так просто это происходит только в тех случаях, когда регистр, в который записан адрес, — это один из семи «старших» регистров, т. е. Р7, Р8, Р9, РА, РВ, РС или РД (их называют немодифицируемыми), да к тому же в этом регистре N записано целое число от 0 до 13. если речь идет о командах КПN или КИПN, либо от 0 до 98, если речь идет об остальных косвенных командах. При этом записанные адреса литерных регистров кодируются так: А — 10, В — 11, С — 12, и Д — 13. Если же N — один из старших регистров, но в него записано не целое число, большее единицы, то это число автоматически округляется, вернее, от него отбрасывается дробная часть, а адресом служит оставшаяся целая часть. Если же записано дробное число между нулем и единицей, то оно не изменяется, но адресом служит его целая часть, т. е. нуль.

— Погоди, погоди, — прервал меня папа. — Ты все это вычитал в руководстве?

— Нет, там об отбрасывании дробной части ничего не говорится. Но я ведь не только читал, я и экспериментировал на моем ПМК. Так вот, я, кажется, все рассказал о косвенных командах при старших регистрах Р7, Р8, Р9, РА, РВ, РС и РД. Немного иначе ведут себя младшие регистры Р0, Р1, Р2, и Р3. Это те самые, для которых существуют команды L0, L1, L2 и L3. В руководстве их называют «модифицируемые вниз». Я бы лучше назвал их регистрами с убывающим содержимым. Если в таком регистре РN записано целое число, скажем n, то по команде КПN, или КИПN, или КБПN и т. д. это число прежде всего уменьшается на единицу и становится адресом для команд П, ИП, БП и т. д., т. е. число из РХ записывается в

$P(n - 1)$ или наоборот из $P(n - 1)$ в RX или безусловный переход осуществляется на шаг $n - 1$ и т. д.

При следующем косвенном обращении к этому регистру его содержимое уменьшится еще на единицу и станет равным $n - 2$ и т. д., пока не дойдет до нуля. После нуля оно становится равным — 99999999, а потом начинает возрастать. В общем содержимое этих регистров изменяется так же, как и по команде LN , и тогда, когда в регистре N записано не целое или не положительное число.

Если в регистрах $P4$, $P5$ и $P6$ записано целое число, то они ведут себя совершенно аналогично, с той лишь разницей, что это регистры не с убывающим, а с нарастающим содержимым или, как пишут в руководстве, модифицируемые вверх. Так, если в $P4$ записано число 8 и подана команда $KP4$, то содержимое $P4$ прежде всего увеличится на единицу и в соответствии с этим число, находящееся в RX , запишется в $P9$. Если команда $KP4$ повторится еще раз, то 9 в $P4$ превратится в 10 и число из RX запишется в PA , поскольку регистру А соответствует номер 10. Если в $P4$ записан 0, то по команде $KP4$, или $KIP4$, или другим косвенным командам, относящимся к этому регистру, 0 превратится в 1, которая и станет косвенно заданным адресом. Вот собственно и все, что я знаю о клавише К.

— В общем, — сказал папа, — знаешь ты довольно много. А знаешь, в какой форме записано целочисленное содержимое регистра PN после косвенного обращения к нему?

— Конечно. Я просто забыл об этом сказать. Это восьмизначная запись с заполнением нулями всех старших разрядов, такая же, как и после команды LN , например 00000002 или 00000015.

— Правильно. Но это не существенно. Я должен заметить, что косвенные команды, в частности $KIPN$, часто используются не для косвенной адресации, а для увеличения или уменьшения содержимого регистра PN , а также для выделения его целой части. С этим мы еще не раз встретимся. Но я хочу обратить твоё внимание на одну особенность регистра $P0$, которая помогла многим авторам составлять хорошие программы, и надеюсь, поможет еще и нам. Дело в том, что помимо команд $KP0$ и $KIP0$, с помощью которых мы записываем или считываем число, находящееся в регистре, номер которого хранится в $P0$, существует возможность сделать то же самое с помощью команд $KP \uparrow$ и $KIP \uparrow$. Да, да, не удивляйся, вместо номера регистра мы нажимаем клавишу \uparrow . Но при таком

косвенном обращении к регистру Р0 единица из его содержимого не вычитается. Один автор назвал поэтому клавишу ↑ «черным ходом» в Р0. Такая возможность обращаться к памяти через регистр Р0 по желанию с уменьшением или без уменьшения его содержимого позволяет значительно упростить и сократить многие программы.

Следует отметить еще некоторые особенности команд КПN и КИПN в регистрах с убывающим содержимым. Действие косвенных команд на содержимое регистров Р0 ... Р3 не совсем такое, как действие команд L0 ... L3. Различие проявляется тогда, когда в соответствующем регистре записано число 1. В этом случае команда LN не изменяет эту единицу, а косвенные команды уменьшают ее до нуля. Когда мы приедем домой, я покажу тебе, как это свойство можно использовать для улучшения одной из наших программ.

Далее следует сказать о свойстве «переноса». Я поясню тебе его на примере. Пусть в Р4 записано число 10. Я применяю команду КП4. Тогда число 10 увеличивается на единицу. Адресу 11 соответствует регистр РВ. Значит, моя команда приведет к записи числа из РХ в РВ. Если еще раз применить команду КП4, то 11 превратится в 12 и запись будет произведена в РС. В следующий раз 12 превратится в 13 и запись будет произведена в РД. Ну, а что, если я еще раз применю команду КП4?

— Не знаю. Больше ведь регистров нет.

— Так вот, после РД мы перейдем к самому началу. В Р4 теперь будет число 14, и оно, как ни странно, соответствует регистру Р0, куда и будет произведена запись. При следующем обращении к команде КП4 число 14 заменится числом 15, а оно, оказывается, соответствует Р1.

— Ну, дальше понятно, — сказал я. — Следующая команда превратит 15 в 16, а этому соответствует Р2, а затем ...

— А вот и нет, — прервал меня папа. — Все идет по такому порядку только до Р1, а затем начинается путаница. Числу 16 соответствует не регистр Р2, а опять Р0, числу 17 — Р1 и только числу 18 — Р2 и числу 19 — Р3. Далее числам 20, 21, 22 и 23 соответствуют опять РА, РВ, РС, РД и т. д. То же самое, конечно, происходит, если не записывать, а извлекать числа из памяти командой КИП4, или КИП5, или КИП6.

Аналогично ведут себя регистры с убывающим содержимым, например Р3. Если в этом регистре записано, скажем, число 16 и мы подадим команду КП3, то число 16

уменьшится на единицу и превратится в 15, а этому числу соответствует Р1, куда и запишется число из РХ. При следующем обращении к этой команде запись произойдет в Р0, затем в РД, РС и т. д. Эти свойства переноса, насколько мне известно, ни в каких книгах не описаны, хотя иногда используются. Мы с тобой тоже будем о них помнить.

За этим разговором мы доехали до дома, где у каждого из нас были свои дела, а я, в частности, сел записывать эти страницы.

15

ЕЩЕ О КЛАВИШЕ К

Сегодня я стал примерять свойства клавиши К к нашим старым программам. Я обратил внимание на программу 19 (вычисление биноминальных коэффициентов), в которой команды L0 12 служат только для уменьшения содержимого Р0 на единицу. Их можно заменить командой КИП0, но она вдобавок вызовет в РХ содержимое какого-то регистра, которое отодвинет хранящийся там результат деления в РY и тем нарушит последующие операции. Для того чтобы этого не произошло, нужно после КИП0 включить команду XY и вывести обратно в РХ хранившийся там результат деления. Вместо XY можно применить команду →, которая также восстановит содержимое РХ.

В любом случае здесь вместо двух шагов L0 12 придется применить также два шага КИП0 XY или КИП0 →, так что никакой экономии в длине программы мы не получим. То же самое относится к программе 13, в которой команды L0 07 можно заменить на КИП0 XY.

Я подумал также о возможности заменить команды L0 12 одной командой КП0. При этом РХ остается нетронутым. Но может произойти еще худшая неприятность — содержимое РХ переписывается в регистр, номер которого хранится в Р0. В какой-то момент содержимое Р0 может стать равным единице, и тогда число из РХ запишется в Р1, и весь ход вычисления нарушенится. Нет, уж лучше с командой КП0 в таких случаях не связываться. Больше я не обнаружил никаких программ, в которых стоило бы применить клавишу К.

Когда папа вернулся с работы, я ему доложил об этих результатах.

— Не густо, — сказал он. — Я ожидал от тебя большего. Мне все-таки не ясно, заменишь ли ты в программе 19 команды L0 12 на КИП0 ХУ?

— Зачем же менять? Это ведь никакой экономии не дает.

— Какой ты стал экономный! Разве только в длине программы дело? А ты забыл, о каком недостатке программы вычисления биномиальных коэффициентов мы говорили? Она дает правильный результат лишь при $m \leq n$, а при $m > n$, когда результат должен быть нулевым, ошибается. А это очень важно, когда эта программа используется как часть другой программы, и забота о том, чтобы вводимые в нее значения m не были больше n , вызовет дополнительные трудности. А отчего этот недостаток возможен?

— Не знаю.

— А ты подумай.

Я думал довольно долго и понял, в чем дело, только после того, как прошел программу 19 по шагам. По формуле (11.3), если $m > n$, один из сомножителей в числителе должен быть равным нулю. В программе эти множители образуются командой L0. Но когда содержимое регистра Р0 уменьшится до единицы, дальнейшего уменьшения эта команда не производит, и поэтому нуля не получается. Что же касается команды КИП0, то она позволяет уменьшить содержимое Р0 до нуля. Таким образом, заменив L0 12 на КИП0 ХУ или КИП0 →, мы получим программу, пригодную для любых целых n и m .

— Убедился? — спросил папа. — Но это еще далеко не все. Ты нашел только те программы, в которых косвенные команды могут служить для увеличения или уменьшения целого числа на единицу. Но эти, хотя и очень важные и широко используемые функции клавиши К все же являются, так сказать, побочными. А основной обязанности этой клавиши — косвенной адресации — ты применения не нашел. По-видимому, ты недостаточно подумал, а ПМК за тебя думать не будет. Посмотри, например, на программу 9 (вычисление многочлена 4-й степени). Вспомни, сначала ты составил программу на 26 шагов, затем рассмотрел некоторые возможные упрощения и сократил ее до 23 шагов. Потом я тебя познакомил со схемой Горнера для вычисления многочленов и, пользуясь ею, мы построили программу 9 на 18 шагов. Теперь, когда ты умеешь пользоваться

ваться косвенными командами, не попробуешь ли еще сократить программу? Это очень важно: во многих задачах приходится попутно вычислять многочлены и необходимо уметь это делать с минимальной затратой программной памяти.

Итак, вычисляем многочлен $P(x) = ax^4 + bx^3 + cx^2 + dx + e$ при различных значениях x . Разместим коэффициенты, как и в программе 2: $a = PA$, $b = PB$, $c = PC$, $d = PD$, $e = P0$, аргумент x вводим в PX .

Вызывать коэффициенты будем косвенными командами через $P4$. С этой целью в $P4$ запишем 9. Тогда при первом обращении к команде КИП4 9 превратится в 10, и будет вызван коэффициент a из PA , второй раз эта же команда вызовет коэффициент b из PB и т. д. Чтобы сосчитать количество вызванных коэффициентов, используем счетчик — регистр 1, куда следует записать число 5 (количество коэффициентов). Величину x будем хранить в $P2$. Таким образом, получим следующее.

Программа 23. Вычисление многочлена 4-й степени (взамен программы 9) $P(x) = ax^4 + bx^3 + cx^2 + dx + e$

П2 9 П4 5 П1 0 ИП2 × КИП4 +

L1 06 С/П

Инструкция. ($a = PA$, $b = PB$, $c = PC$,
 $d = PD$, $e = P0$) $x = PX$ В/О С/П $PX = P(x)$.

Пример. Для $a = 0,05$, $b = -0,1$, $c = 0,12$, $d = 1,5$, $e = 1,7$ при $x = 3$ $P(x) = 8,63$. Продолжительность вычисления — около 15 с.

— Как видишь, — продолжал пapa, — с помощью команды косвенного обращения удалось сократить программу 9 с 18 до 13 шагов. Теперь она вдвое короче, чем твой первый вариант (26 шагов).

Но, конечно, не в краткости основное достоинство программ. Этую программу стоит немного удлинить (не увеличивая продолжительности вычисления), чтобы облегчить ввод коэффициентов. Для этого используем команду КП4, по которой коэффициенты разместим по регистрам, а вручную будем набирать только сами числа. С этой целью после команды С/П наберем:

↑ 9 П4 ХУ КП4 С/П БП 17

а инструкцию предложим такую: ($a = RX$, БП 13 С/П, $b = RX$ С/П, $c = RX$ С/П, $d = RX$ С/П, $e = RX$ С/П) $x = RX$ В/О С/П $RX = P(x)$.

Я все это проделал и убедился, что папино предложение полезно. Вводить коэффициенты стало значительно проще, а время самой вычисления не изменилось.

— Теперь обрати внимание на самую последнюю и наиболее «мощную» нашу универсальную программу для решения треугольников (программу 22). Не далее, как завчера, ты сокрушился, что площади треугольников эта программа не вычисляет. Попробуем сейчас избавиться от этого недостатка, используя команды косвенной адресации. Думаю, что это удастся, так как нам не хватало только пяти шагов.

Я воодушевился, достал программу 22 и стал искать, куда бы вставить косвенную адресацию, да так, чтобы сэкономить пять шагов. Но ничего не приходило в голову. Помогла папина подсказка:

— Смотри, в первой строке есть команда перехода к подпрограмме ПП 68. Она занимает два шага. Заменим ее одной командой КПП7 (косвенного перехода к подпрограмме), а число 68 запишем в Р7, который свободен. Результат будет тот же. Переходим к подпрограмме, начинавшейся с шага 68, но вместо двух шагов мы заняли один. Сколько мы сэкономим?

— Один шаг.

— Нет, Миша, три. Ведь та же команда у нас повторится в 3-й и 5-й строках. Теперь дальше — в 4-й и 5-й строках одна и та же команда ПП 76. Заменим их командой АПП 8, записав в Р8 число 76, и мы сэкономим ... сколько?

— Еще два шага, — ответил я, — т. е. получим все, что нам нужно.

— Еще не все. Нам нужно в двух местах вместо команды С/П осуществить переход к вычислению площади. Это тоже нужно будет сделать командой косвенного перехода КИР9, записав в Р9 соответствующий адрес перехода. Теперь ты видишь еще одно достоинство косвенных переходов — они позволяют сэкономить один шаг на каждом обращении.

— Тогда, папа, можно в этой программе заменить еще много условных и безусловных переходов на косвенные и сэкономить на каждой по одному шагу. Смотри, их у нас еще шесть, так что можно выгадать шесть шагов и вычислить еще что-нибудь.

— К сожалению, у тебя не хватит для этого регистров. Шесть регистров у нас заняты элементами треугольника и два, Р4 и Р5, используются как оперативные в ходе вычис-

лений. Остается всего шесть регистров, из которых Р0 и Р6 использовать нельзя: у них содержимое при косвенном обращении изменяется. Можно, конечно, перераспределить регистры, но много на этом не выиграешь. Впрочем, тут остается простор для твоей фантазии. А пока используем то, что можно сделать легко.

И мы стали перерабатывать программу 22, в результате получили следующую программу.

Программа 24. Универсальная для решения треугольников (включая вычисление площади S).

ИП2	x = 0 29	ИПВ	x = 0 25	ИПС	КПП7	ПВ	ИП3
ИПА	sin ×	ИПС	sin ÷	П1	ИПВ	sin ×	
ИПА	sin ÷	П2	КБП9	КПП7	ПС	БП 09	ИПА
x = 0	45 ИП1	ИП3	ИП2	КПП8	ПА	ИП2	ИП3 ИП1
КПП8	ПВ	КПП7	ПС	КБП9	ИП2	x ²	ИП3 x ² +
ИП2	ИП3 ×	2	×	ИПА	cos ×	—	✓
П1	БП 37	ИПА	+ cos	↑	/ - /	arccos	НОП
В/О	П4 x ²	ХУ	П5 x ²	+	ХУ	x ²	—
ИП4	ИП5 ×	2	×	÷	БП 68	ИП1	ИП2
×	ИПС sin	×	2	÷	ПД	С/П	

Инструкция. (63 = Р7, 71 = Р8, 88 = Р9) $a = P_1$, $b = P_2$, $c = P_3$, $A = PA$, $B = PB$, $C = PC$ (все неизвестные величины заменяются нулями) В/О С/П РХ = РД = S , Р1 = a , Р2 = b , Р3 = c , РА = A , РВ = B , РС = C . Правила обозначения такие же, как в программе 22.

Примеры. Те же, что и к программе 22. Дополнительно вычисляются значения $S = 19,60; 0,542; 6138,2; 0,3499$. Продолжительность вычислений — около 40 с.

— Обрати внимание, — сказал папа, — время вычислений по этой программе немного больше, чем по программе 22. Несколько секунд тратится на вычисление площади треугольника. Но основная причина замедления — это то, что косвенные переходы занимают больше времени, чем прямые. Поэтому не следует где надо и где не надо заменять все переходы на косвенные, чтобы сократить программу на несколько шагов. Это нужно делать, только когда это настоятельно необходимо.

Теперь ты убедилсяся, какие преимущества сулит применение клавиши К. А теперь я предложу тебе для закрепления навыков программирования несложную задачу.

Ты видел на примере программы 23, как вычисляется многочлен 4-й степени. Конечно, совсем нетрудно аналогичную программу построить для многочленов 5-й, 6-й, 7-й, 8-й и даже 9-й степени. Для этого придется только изменить числа, вводимые в регистры Р4 и Р1. Дальше увеличивать степень многочлена становится труднее — недостаточно регистров для записи коэффициентов и для управления их вызовом. Попытайся составить программу для вычисления многочленов более высокой степени, хотя бы 12-й:

$$P_{12}(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{11} x^{11} + a_{12} x^{12}. \quad (15.1)$$

При этом число шагов должно быть не более 20.

Я понимаю, что это непросто, возможно, придется применить какие-то хитрости. Алгоритм тебе известен — это схема Горнера, которую ты уже использовал. Так что все дело в том, чтобы уже известный алгоритм уложить в ограниченное число шагов и регистров памяти.

Пока будем вводить коэффициенты вручную, однако только один раз, до начала вычислений. Эта задача представляет не чисто спортивный интерес. Много различных функций, встречающихся в физических и прикладных задачах, можно аппроксимировать многочленами тем точнее, чем выше степень многочлена. Поэтому в вычислительной математике умение вычислять такие многочлены играет большую роль. Думаю, что тебе пары часов для создания такого шедевра хватит.

С этими словами папа ушел и оставил меня наедине со своим заданием.

Я начал с таких прикодок. Если строить программу по образцу программы 9, т. е. не используя косвенную адресацию, а вызывая каждый коэффициент из того регистра, в котором он хранится, то можно будет из 14 регистров использовать один для хранения значения x , а остальные 13 — для коэффициентов с a_0 по a_{12} . Но на вычисление многочлена 12-й степени уйдет около 40 шагов. Каждый коэффициент нужно вызвать из памяти, сложить с текущим результатом и умножить на x , т. е. проделать не менее 4 шагов на каждый коэффициент, кроме первого и последнего, что ясно видно из программы 9. В заданные 20 шагов никак не уложиться.

Скедовательно, нужно, использовать косвенный вызов коэффициентов, как в программе 23. Но там использован один регистр для хранения x , другой — для управления косвенным вызовом и третий — в качестве счетчи-

ка. Значит, для хранения коэффициентов остается 11 регистров, а это позволяет записать только многочлен 10-й степени. Опять нехватка.

Размыслия о резервах, я вспомнил про операционный стек. Какие-то величины можно хранить в нем. Что именно? Прежде всего x , а кроме того, один из коэффициентов, например старший a_{12} . Он ведь нужен только в начале вычисления — хранить его придется недолго. Так, конечно, сделать можно, но тогда всякий раз, начиная новое вычисление, придется вводить не только значение x , но и a_{12} . Может быть, это не так трудно, но явно противоречит условию моего задания.

Еще немного поразмыслив, я понял, что хранить в операционном стеке кроме значения x еще и один из коэффициентов не удастся. Ведь там все время хранится текущий результат сложений и умножений. Итак, приходится считаться с тем, что из 14 регистров памяти 13 будут заняты коэффициентами многочлена, в операционном стеке будет храниться значение x , и один-единственный регистр останется для управления косвенным вызовом и для счета вызванных коэффициентов. Конечно, команду LN здесь применить не удастся, поскольку в этом же регистре содержимое будет изменяться на единицу при каждом косвенном обращении. Но я могу всякий раз проверять, не уменьшилось ли его содержимое до нуля, и этим ограничить число циклических повторений.

Теперь можно составить программу. В качестве управляющего регистра выберу $P0$. Тогда a_0 запишу в РД, $a_1 = P1$, $a_2 = P2$, $a_3 = P3$, ..., $a_9 = P9$, $a_{10} = PA$, $a_{11} = PB$ и $a_{12} = PC$. В $P0$ запишу число 13, тогда вызов коэффициентов начнется с PC . Число 13 нужно записывать в $P0$ всякий раз, начиная вычисление. Его лучше включить в программу. Значение x вводим в PX и сразу в программе переводим в PY , чтобы отдельить от числа 13, вводимого в $P0$. Итак, получилась программа

$\begin{array}{ccccccccc} \uparrow & 1 & 3 & P0 & XY & \uparrow & \uparrow & 0 & КИП0 \quad x \neq 0 \\ 15 & + & \times & БП & 08 & \rightarrow & ИПД & + & С/П \end{array}$

Она имеет 19 шагов. Первые 8 шагов подготовительные. Здесь я ввожу число 13 в $P0$, а также добиваюсь того, чтобы в PZ и PT находился x , что гарантирует его сохранение в операционном стеке при любом числе умножений на него. Для большей верности я составил таблицу, по кото-

рой видно, как размещены данные в стеке после каждого шага:

Шаг	00	02	04	05	06	07	08	11	12	08	11
Регистр											
T				x	13	13	x	x	x	x	x
Z		x	x	13	x	x	x	x	x	x	x
Y	x	x	13	x	x	x	0	x	x	$a_{12}x$	x
X	x	13	x	x	x	0	a_{12}	a_{12}	$a_{12}x$	a_{11}	$a_{11}x + a_{11}$

Далее на шагах 08, 11 и 12 осуществляются основные операции схемы Горнера — сложение со следующим коэффициентом и умножение на x . В то же время на шаге 09 проверяется, не вывела ли команда КИП0 нуль, что произойдет тогда, когда содержимое Р0 станет равным нулю и окажется, что косвенный адрес совпал с номером того регистра, в котором он хранится. (Правда, для работы этой программы нужно, чтобы все коэффициенты многочлена $P_{12}(x)$ отличались от нуля, но об этом немного позже.) Когда содержимое Р0 дойдет до нуля, управление передается шагу 15, на котором команда → служит для того, чтобы убрать введенный нуль. Затем к полученному результату добавляется a_0 , хранящееся в РД, и вычисление закончено. Всего на него ушло 19 шагов, так что в папи-но условие я уложился.

Неприятным в этом решении является требование, чтобы ни один коэффициент не равнялся нулю. Чтобы его можно было всегда выполнить, не искажая результат вычисления, я предлагаю в тех случаях, когда какой-либо коэффициент на самом деле равен нулю, скрыть это от ПМК, введя в соответствующий регистр вместо нуля число $1 \cdot 10^{-99}$. Если остальные коэффициенты не чрезмерно малы, скажем больше 10^{-90} , то эти фиктивные коэффициенты, заменяющие нули, не внесут никаких ошибок.

Все-таки, меня такое решение не вполне удовлетворило. Поэтому я продолжал думать. Если бы операцию LN можно было производить над регистром, содержащимое

которого не изменяется при косвенном обращении, то все было бы просто. Я бы осуществлял косвенный вызов, а затем командой LN уменьшал содержимое регистра на единицу и в нужный момент, после того как считан a_1 , осуществлял бы переход к заключительной части.

И вдруг меня осенило — ведь такой регистр существует! И это именно $P0$, который я использовал. Нужно только входить в него с «черного хода», о котором папа мне рассказывал вчера. Теперь все очень просто и можно составить программу, которой не грех дать номер.

Программа 25. Вычисление многочлена 12-й степени $P_{12}(x) = a_0 + a_1x + a_2x^2 + \dots + a_{11}x^{11} + a_{12}x^{12}$

```
↑ 1 2 П0 XY ↑ ↑ 0 КИП↑ +
× L0 08 ИПД + С/П
```

Инструкция. ($a_0 = РД$, $a_1 = Р1$, $a_2 = Р2$, $a_3 = Р3$, $a_9 = Р9$, $a_{10} = РА$, $a_{11} = РВ$, $a_{12} = РС$) $x = РХ$ В/О С/П $РХ = P_{12}(x)$. Если степень многочлена меньше 12, то недостающие коэффициенты заменить нулями.

Пример. Для $a_0 = 1000$, $a_1 = 100$, $a_2 = 10$, $a_3 = 1$, $a_4 = 0,1$, $a_5 = 0,01$, $a_6 = 0,001$, $a_7 = 10^{-4}$, $a_8 = 10^{-5}$, $a_9 = 10^{-6}$, $a_{10} = 10^{-7}$, $a_{11} = 10^{-8}$, $a_{12} = 10^{-9}$ $P_{12}(2) = 1250$, $P_{12}(3) = 1\ 428,5712$, $P_{12}(10) = 13\ 000$. Время вычисления — 27 с.

Этой программой я остался доволен и отнес ее папе. Папа согласился с тем, что я сделал, и дал еще пару советов.

— Очень часто, — сказал папа, — косвенная адресация, а иногда команда LN , применяется для выделения целой части положительного числа x . Для этого число x записывается в какой-либо из «старших» регистров ($Р7\dots РД$), обозначим номер его N . Тогда команда КИП N приведет к тому, что дробная часть числа x в $РN$ сотрется и сохранится только целая его часть. Кроме того, в $РХ$ поступит число, хранящееся в том регистре, номер которого соответствует целой части числа x , а все то, что до этого было в операционном стеке, поднимется на одну ступеньку, как мы рисовали на рис. 3.1. Поэтому при составлении программы ты должен убедиться, что это не нарушит хода дальнейших вычислений. В противном случае нужно принять меры, например такие, какие ты предлагал для программы 19 при замене команд $L0 12$ на КИП 0 XY или КИП $0 \rightarrow$ (см. стр. 111).

Еще одно ты должен запомнить. Команда КИПN выделяет целую часть числа x , только если $x > 1$. В противном случае содержимое PN должно было бы стать равным нулю (целой части положительной дроби, меньшей 1), но оно сохраняется прежним. Поэтому если ты, составляя программу, заранее не знаешь, что число x всегда будет больше единицы, то для выделения его целой части нужно действовать по-другому. Нужно прибавить к x единицу и запомнить $x + 1$ в каком-либо из PN ($N = 0, 1, 2, 3$) с убывающим содержимым. Тогда команда КИПN обеспечит появление в PN целой части положительного числа x . Но это потребует двух лишних шагов.

— А зачем вообще выделять целую часть числа, если на самом деле оно не целое? Для округления?

— Нет, не только, — ответил пapa. — Впрочем, скоро ты сам в этом убедишься.

16

ДОШЛИ ДО ПРЕДЕЛА

Сегодня пapa вернулся с работы какой-то озабоченный. За обедом он не произнес ни слова. Но мне очень хотелось выяснить, откуда взялось число e и натуральные логарифмы. Я подошел к папе и сказал, что у меня возникли неотложные вопросы.

— Отстань. Сегодня мне тоже надо решить неотложные вопросы, подумать о них хотя бы с полчаса без посторонних помех.

— Это я-то посторонняя помеха? — возмутился я. — А кто захотел обучить меня программированию? Теперь я от тебя не отстану, пока не научусь.

— Ладно, придется мне прибегнуть к тому же способу, который применил когда-то учитель Гаусса. Когда Гауссу было лет восемь, его учителю нужно было заняться своими делами и он задал классу такую арифметическую задачу — вычислить сумму всех натуральных чисел от единицы до 100: $1+2+3+4+\dots+100$. Он прикинул, что вряд ли его ученики справятся с этим заданием быстрее, чем за полчаса. Вот и я хочу тебе дать эту же задачу. Но во времена Гаусса не было микрокалькуляторов, а у тебя он есть. Поэтому вычисли сумму натуральных чи-

сел от единицы до тысячи. Мы запишем это в таком виде
1000

$\sum_{i=1}^{1000} i$. Это означает сумму всех чисел i , когда i принимает подряд значения от 1 до 1000.

— Пожалуйста, нет ничего проще. И я взял свой ПМК и отошел с ним в сторону. Мне долго думать не пришлось — ведь задача эта почти такая же, как и вычисление факториала. Нужно только сменить умножение на сложение. Поэтому программу я составил сразу и, конечно, в общем виде для вычисления суммы $1+2+3+\dots+N$, для любого натурального N :

ПО О ИПО + L0 02 С/П

Инструкция. $N = RX \text{ B/O C/P RX} = \sum_{i=1}^N i$.

Я быстро проверил ее для малых N и, убедившись, что она считает верно, ввел $N = 1000$ и стал ожидать результата. Через 25 мин он высветился на индикаторе — 500 500. С этим я подошел к папе, который меня уже поджидал.

— Вот видишь, — сказал папа. — Мне повезло больше, чем учителю Гаусса. Я действительно за это время со своими делами справился, а Гаусс не дал такой возможности своему учителю. Он принес ему ответ примерно через 2 мин.

— Но ведь Гаусс считал только до 100, а я — до 1000. До 100 я тоже за 2 мин просчитал.

— Не в том дело, — сказал папа. — У Гаусса калькулятора не было, и, конечно, он не суммировал все числа подряд как это делает твой ПМК. Вместо этого он за 2 мин вывел общую формулу $\sum_{i=1}^N i = 0,5 N(N + 1)$ и, подставив в нее $N = 100$, сразу получил 5050. Если бы ему нужно было сосчитать сумму до $N = 1000$, то он тоже воспользовался бы своей формулой и нашел бы ответ 500 500 за те же 2 мин. Но ты, к сожалению, не Гаусс ...

Я немного сконфузился, но сразу сообразил, что не всем же быть Гауссами. Я только поинтересовался, как Гаусс додумался до этой формулы.

— Он записал два равенства для искомой суммы x :

$$x = 1 + 2 + 3 + \dots + (N-2) + (N-1) + N;$$

$$x = N + (N+1) + (N-2) + \dots + 3 + 2 + 1,$$

а затем почленно сложил эти формулы. Тогда в левой части он получил $2x$, а в правой — N одинаковых слагаемых, каждое из которых равно $N + 1$, так что

$$2x = N(N + 1), \quad (16.1)$$

откуда сразу вытекает его формула.

Из этой формулы легко получить и выражение для суммы членов арифметической прогрессии. Кстати, одним из основных достоинств ПМК является возможность довольно быстро вычислять такого рода суммы, в том числе и бесконечные. И мы с тобой обязательно этим займемся, когда ты научишься решать на ПМК более простые задачи.

— А что такое бесконечная сумма?

— Тебе все не терпится забежать вперед и узнать все математические понятия. Ладно, это не так уж плохо, тем более что понятие предела тебе скоро понадобится. Так вот, предположим, что тебе дана некоторая последовательность чисел $a_1, a_2, a_3, a_4, \dots$. Другими словами, тебе известно, как можно вычислить значение a_k при любом натуральном числе k . Чтобы тебе было понятнее, возьмем конкретный пример. Пусть $a_1 = 1, a_2 = q, a_3 = q^2, a_4 = q^3$ и т. д. Такая последовательность называется геометрической прогрессией. Здесь q — некоторое заданное число, называемое знаменателем прогрессии. Возьмем n членов этой прогрессии и вычислим их сумму:

$$S_n = 1 + \sum_{i=1}^{n-1} q^i. \quad (16.2)$$

Это совсем нетрудно сделать на калькуляторе. Но попробуем вывести общую формулу для такой суммы, запишем ее так:

$$S_n = 1 + q + q^2 + q^3 + \dots + q^{n-1}.$$

А теперь подумай, как бы поступил восьмилетний Гаусс на твоем месте.

— Не знаю. Записывать эту сумму в обратном порядке, по-видимому, нет смысла. А что, если умножить ее на q ? Тогда получится такая же сумма, но только без единицы вначале и с лишним членом q^n .

— Идея хорошая, — сказал пapa и записал

$$qS_n = S_n - 1 + q^n. \quad (16.3)$$

Получилось уравнение относительно S_n , которое очень легко решается:

$$S_n(1-q) = 1 - q^n,$$

$$S_n = (1 - q^n)/(1 - q). \quad (16.4)$$

Это, конечно, не общая формула для суммы членов геометрической прогрессии, а только частный случай при $a_1 = 1$.

Рассмотрим случай, когда $q < 1$, например $q = 1/2$, и определим для каждого n величину S_n . Очевидно, $S_n = [1 - (1/2)^n]/(1 - 1/2) = 2(1 - 1/2^n)$. Прикинь-ка на ПМК значения S_n хотя бы до $n = 8$.

Я сделал это очень быстро по такой программе:

$\uparrow 2 X^Y 1/x 1 XY - 2 \times C/P$

И н с т р у к ц и я. $n = RX$ В/О С/П РХ = S_n .

Получил следующие результаты:

n	1	2	3	4	5	6	7	8
S_n	1	1,5	1,75	1,875	1,9375	1,96875	1,984375	1,9921875

— Очень хорошо, — сказал папа. — Но ведь ты мог получить эти же результаты, не прибегая к формуле (16.4), а просто суммируя значение a_k , по такой программе.

Программа 26. Суммирование геометрической прогрессии при $a_1 = 1$

П9 1 ПА ПС С/П ИП9 × ПА ИПС

+ ПС С/П БП 05

И н с т р у к ц и я. $q = RX$ В/О С/П РХ = S_1 С/П, $RX = S_2$ С/П, ..., $RX = S_n$.

Хотя эта программа и длиннее, но она намного удобнее: не нужно каждый раз вводить n , а достаточно нажимать клавишу С/П. Кроме того, она универсальнее — можно ввести любое q , а не только $q = 1/2$, как в твоей программе. Как видишь, я ввожу в РА значения a_k . Начинаю с $a_1 = 1$, а затем использую, что $a_k = a_{k-1}q$. А значение q у меня записано в Р9. В РС я записываю S и тоже начинаю

с $n=1$, а затем подсчитываю S_n по формуле $S_n = S_{n-1} + a_n$. Такой способ вычисления с переходом от S_{n-1} к S_n называется рекуррентным.

— Я проверил, что папина программа для $q = 1/2$ и n до 8 дает те же результаты, что и моя.

— Хорошо, — сказал пapa. — Введи теперь $q = 1,1$ и просчитай несколько значений S_n .

Я так и сделал. При этом я получил $S_8 = 11,435888$, а $S_{20} = 57,274995$. Кроме того, я рассчитал S_{20} по формуле (16.4) и получил тот же результат, кроме последней цифры.

— А теперь подумай, — сказал пapa. — Как будет изменяться S_n , если n безгранично возрастает, т. е., например, если мы подсчитаем S_{100} , потом S_{1000} , потом $S_{1\,000\,000}$ и т. д.?

— Здесь не хватит разрядов в ПМК.

— Значит, величина S_n будет все время возрастать и сможет превзойти любое наперед заданное число?

— Да, конечно.

— Так действительно будет, если $q > 1$. А если $q = 1$?

— Тут что-то не ясно, — сказал я, взглянув на формулу (16.4). — Здесь получается $0/0$, а это неопределенность.

— Запомним эту неприятность. Мы еще с такими случаями столкнемся. А все-таки, чему равно S_n при $q = 1$? Может быть, ты это определишь, не прибегая к формуле (16.4), которая на этот раз отказалась?

— Да очень просто. При $q = 1$ $S_n = 1 + 1 + 1 + \dots + 1$ — сумма n единиц и, значит, равна n . При достаточно большом n она может стать больше любого заданного числа.

— А если $q < 1$?

— Да, наверное, тоже. Мы ведь все время с ростом n прибавляем к S_n все больше и больше членов и, значит, рано или поздно превзойдем любое число.

— Взгляни на те результаты, которые ты получил при $q = 1/2$.

— И тут значения S_n растут. Правда, очень медленно. И никак не могут стать больше 2. Почему это? Я начинаю сомневаться...

— А ты взгляни еще разок на формулу (16.4). Видишь, знаменатель не зависит от n , а числитель, конечно, возрастает с увеличением n , так как величина q^n , вычитаемая из единицы, уменьшается, если $q < 1$. Но до какой степени?

— Да ..., — удивился я. — Когда n возрастает, q^n уменьшается, но все-таки остается положительным, так что числитель всегда меньше 1. Таким образом, S_n , при $q < 1$ возрастает, но не беспредельно. Всегда $S_n < 1/(1-q)$ при любом n . Ну, конечно, когда $q = 1/2$, то $S_n < 2$. Так оно и получилось.

— А как ты думаешь, можно ли выбрать такое большое n , чтобы S_n отличалось от 2 меньше чем на 10^{-8} ?

— Конечно, можно. Нужно, чтобы $q^n = 0,5^n$ было меньше, чем $0,5 \cdot 10^{-8}$. А для этого достаточно взять n больше ... (подожди, я сейчас на калькуляторе прикину)... больше 28, так как $(0,5)^{28} \approx 3,725 \cdot 10^{-8}$.

— Значит, можно сказать, что при любом положительном $q < 1$ и любом n $S_n < 1/(1-q)$. Но каково бы ни было положительное число ε , можно найти такое N (разное для разных ε), что $|S_n - 1/(1-q)| < \varepsilon$ при $n > N$. Так определяется в математике стремление к пределу. Поэтому можно сказать, что S_n стремится к пределу, равному $1/(1-q)$. Это верно и для отрицательных q , если $|q| < 1$. Ты легко можешь убедиться в этом на своем ПМК.

Я засел за калькулятор и стал считать, как изменяется S_n с ростом n при $q = 0,1$. В этом случае предел равен

$\frac{1}{1-0,1} = 1,111111\dots$ По программе 26 я получил последовательные значения S_n , равные 1; 1,1; 1,11; 1,111; 1,1111; 1,11111 и т. д. После 1,111111 показания индикатора уже не изменялись, но это не значит, что S_n достигла своего предела, просто следующие значения a_n образуют относительно S_n машинный нуль.

Я попробовал ввести еще $q = 0,9$. В этом случае предел равен 10. (Папа показал мне, что это записывается так: $\lim_{n \rightarrow \infty} S_n = 10$.) Я получил такую последовательность

$S_n : 1; 1,9; 2,71; 3,439; 4,0951$ и т.д. и только при $n = 50$ получил $S_n = 9,9536164$. При увеличении n величина S_n продолжала увеличиваться, явно стремясь к 10, но очень медленно.

Затем я ввел $q = -0,1$. В этом случае $\lim_{n \rightarrow \infty} S_n = 1/(1+0,1) = 0,90909\dots$, и приближение к пределу произошло быстро. Интересно, что при $q < 0$ величина S_n приближается к пределу не снизу, как при $q > 0$, и не сверху, как я в первый момент подумал, а с обеих сторон. В данном случае я получил $S_n = 1; 0,9; 0,91; 0,909; 0,9091; 0,90909$ и т. д.

— Значит, — сказал я, — сумма членов геометрической прогрессии при $|q| < 1$ стремится к пределу, а при $|q| \geq 1$ не стремится.

— Можно было бы сказать и так. Но в математике принято говорить, что при $|q| > 1$ величина S_n стремится к бесконечности. Это значит, что каково бы ни было на-перед заданное число M , всегда можно найти такое N (зависящее от M), что при $n > N$ $S_n > M$. Впрочем, есть и такое значение q , при котором S_n не стремится ни к какому пределу. Это $q = -1$. Посмотри, как при этом ведет себя S_n .

Я ввел $q = -1$ в программу 26 и увидел, что S_n принимает значения 1; 0; 1; 0 и так сколько угодно. Так что нет такого предельного значения, к которому S_n приближалось бы. Значит, бывают величины, зависящие от n , которые с увеличением n не стремятся ни к какому пределу, ни к конечному, ни к бесконечному.

Затем папа рассказал мне, что пределы бывают не только у переменных, зависящих от целочисленного индекса n , но и у переменных, зависящих от любого аргумента x . Мы говорим, что функция $y = f(x)$ стремится к пределу a , когда x стремится к b , и пишем $\lim_{x \rightarrow b} y = a$, если, каково бы ни было положительное число ε , существует такое число δ (для разных ε разное), что, когда $|x - b| < \delta$, выполняется неравенство $|y - a| < \varepsilon$.

— Вот тебе пример, — сказал папа. — Пусть $y = \sin x/x$ (где x измеряется в радианах) и нас интересует предел y , когда x стремится к нулю. Заметь, что при $x = 0$ выражение для y смысла не имеет, так как $\sin 0/0 = 0/0$ — неопределенность. Однако существует предел $\sin x/x$, когда x стремится к нулю. Попробуй определить его с помощью микрокалькулятора. Только не забудь поставить переключатель Р — Г в положение Р.

Я взял $x = 0,1$ и, введя (без программирования) $0,1 \uparrow \sin XY \div$, вычислил $\sin x/x = 0,99833417$. Затем я ввел $x = 0,01$, и это дало 0,9999834. Еще уменьшил x в 10 раз и получил $\sin x/x = 0,999999$. Теперь все ясно: $\lim_{x \rightarrow 0} (\sin x/x) = 1$. И действительно, если взять $x \leq 0,0001$,

то мы получим 1, так как в пределах восьми разрядов микрокалькулятора $\sin 0,0001$ не отличается от аргумента.

— Вот другой пример. Рассмотрим отношение $\log_m(1+x)/x$, когда x стремится к нулю. Здесь логарифм взят по основанию m , которое будем считать положитель-

ным и большим единицы. Тогда при $x = 0$ числитель, так же, как и знаменатель, обращается в нуль, и опять получается неопределенное выражение. Можем ли мы ожидать, что здесь, как и в предыдущем примере, предел равен 1? Конечно, нет. Этот предел может быть различным при разных основаниях логарифма. Попробуй-ка, что получится, если $m = 10$, т. е. мы применяем обычный десятичный логарифм.

Я взял достаточно малое $x = 10^{-4}$, подсчитал $\lg(1,001)/10^{-4}$ и получил 0,43427276 — совершенно не похоже на единицу. Я уменьшил x еще в 100 раз и получил $\lg(1,000001)/10^{-6} = 0,43429426$. Очевидно, что эта величина уже достаточно близка к пределу.

— Вот видишь, что получается, — сказал папа. — А между тем многие математические вопросы решаются значительно проще, если основание логарифма m выбрано так, что $\lim_{x \rightarrow 0} [\log_m(1+x)/x] = 1$. Как выбрать значение m ,

чтобы получить здесь предел 1? Давай подумаем.

Если $v = \log_m(1+x)$, то это значит, что $m^v = 1+x$ или $m = (1+x)^{1/v}$. А мы хотим, чтобы при приближении x к 0 величина v становилась близкой к x . Значит, m должно равняться пределу выражения $(1+x)^{1/v}$, когда x стремится к нулю, а v приближается к x , т. е.

$$m = \lim_{x \rightarrow 0} (1+x)^{1/x}. \quad (16.5)$$

Вот и прикинь, хотя бы приблизительно, чему равно m . Я ввел такую программу:

P9 1 + ИП9 1/x XY X^Y С/П

Инструкция. $x = РХ В/О С/П РХ \approx m$.

Надо взять x достаточно близким к нулю (но, конечно, не настолько маленьким, чтобы калькулятор не отличал $1+x$ от 1). Возьмем, скажем, $x = 5 \cdot 10^{-7}$. В качестве оценки m я получил 2,7182812. Затем, уменьшив x до $2 \cdot 10^{-7}$, получил $m \approx 2,7182817$.

— Ты нашел почти точное значение этого особенного основания логарифмов, которое играет очень важную роль в математике. Только его обозначают не m , а буквой e . Это иррациональное число, приблизительно равное 2,718281828459045 ... Логарифмы с основанием e называют натуральными и обозначают \ln . Обычно в учебниках число e определяют как предел выражения $(1+1/n)^n$, когда n

неограниченно возрастает. Но это совершенно эквивалентно (16.5). Записывается это так:

$$e = \lim_{n \rightarrow \infty} (1 + 1/n)^n. \quad (16.6)$$

— А я как раз и собирался у тебя спросить, откуда взялось число e . Теперь мне все стало ясно.

— Все то, о чем мы сегодня говорили, в частности о пределах, тебе в дальнейшем понадобится. Так что запомни это хорошенько.

17

МСЬЕ АНТЬЕ И МАДАМ ФРАКСОН

Сегодня мама сидит дома в ожидании ночного машинного времени. Ей захотелось меня повоспитывать.

— Как дела у твоего микрокалькулятора? Он уже все умеет считать? — спросила она.

— Без пяти минут все, — ответил я. — Я уже научил его алгебре — он вычисляет многочлен, тригонометрии — он решает треугольники.

— Конечно, — сказала мама, — алгебра и тригонометрия — это довольно простые разделы математики и с ними микрокалькулятор справляется легко. Значительно более сложный раздел — это арифметика. Интересно, сможешь ли ты научить микрокалькулятор решать задачи арифметики.

Я знал, мама любит парадоксы, и поэтому нисколько не удивился, что она считает арифметику труднее алгебры и тригонометрии. Я только спросил, какие арифметические задачи она хотела бы поручить ПМК.

— Начнем, пожалуй, с алгоритма Евклида для нахождения общего наибольшего делителя (ОНД) двух чисел. Помнится, ты говорил, что знаешь о его существовании. Напомню его сущность. Большее из двух чисел делят на меньшее. Если оно разделилось нацело, т. е. остаток равен нулю, то делитель (меньшее число) и есть ОНД. В противном случае большее число заменяется на меньшее, а меньшее — на остаток, и все повторяется, пока не получится деление без остатка. Тогда последний делитель

и есть ОНД. Покажи-ка, как ты будешь это считать на микрокалькуляторе. Для этого он ведь должен уметь делить с остатком.

Я, конечно, не представлял себе ясно, как делить на ПМК с остатком, но не сказал об этом, а обязался за час такую программу составить, если мама не будет мне мешать.

На этом мы расстались, и я стал думать. Если разделить большее число n на меньшее m , то получу в общем случае не целое число $a > 1$. Как выделить из него целую часть, я знаю. Обозначим ее ц. ч. $a = b$, а разность $a - b =$ д. ч. a (дробная часть a). Если она равна нулю, то, значит, n разделилось на m нацело. Если же д. ч. $a \neq 0$, то n [д. ч. a] и есть остаток.

Тут мама все-таки вмешалась.

— Какие у тебя доморощенные обозначения ц. ч. и д. ч. Откуда ты их взял?

— Сам придумал.

— Есть ведь общепринятые обозначения, разве вам в школе их не давали? Частное при делении с остатком обозначается k , а остаток — r . Целая часть числа a называется «антье a » и обозначается $E[a]$, а иногда и просто $[a]$, а дробная часть — «фраксьон a » и обозначается $F[a]$ или реже просто $\{a\}$.

— А кто такие были Антье и Фраксьон? Математики?

— Ну, что ты! Просто по-французски антье — значит целое, пишется entier, поэтому и обозначается буквой E , а фраксьон (fraction) — дробь.

— Я это запомню. Только мне больше нравится считать, что это — красивые французские фамилии. Буду теперь называть целую часть мсье Антье, а дробную — мадам Фраксьон. И теперь перепишу эти формулы так:

$$n/m = a = E[a] + F[a]; \quad (17.1)$$

$$k = E[a]; \quad r = mF[a]; \quad n = mk + r. \quad (17.2)$$

На микрокалькуляторе, если $n = P1$ и $m = P2$, можно вычислить k и r таким путем:

ИП1 ИП2 \div П9 КИП9 Х Y ИП2 — ИП2 \times С/П

Частное k записано в Р9, а остаток r — в РХ. Я записал n/m в Р9 — немодифицируемый регистр. Поэтому косвенная команда КИП9 выделила целую часть — мсье Антье, которая и оказалась записанной в Р9. Однако в РY сохранилось исходное значение n/m , которое я извлек

командой XY, а затем, вычтя из него мсъе Антье, я получил мадам Фраксьон, которую и умножил на m , чтобы найти остаток r . И все это за 11 шагов.

Я ввел этот кусочек программы и испробовал его. Разделив 45 на 4, получил $k = 11$ и $r = 1$, как и следовало ожидать. Разделив 256 на 13, получил $k = 19$, но почему-то $r = 8,999991$. С чего бы это? Ведь остаток должен быть целым числом.

— Все понятно, — сказала мама. — Когда ты делил, ты получил результат не совсем точный, а только в пределах восьми знаков. Далее, умножая дробную часть на 13, ты абсолютную погрешность увеличил в 13 раз. Вот и получилось у тебя число, близкое к девяти, но с относительной погрешностью порядка $1 \cdot 10^{-6}$. Мы, когда решаем задачи с натуральными числами, стараемся программировать так, чтобы погрешностей не было. Арифметика любит абсолютную точность.

— Как же это можно сделать? Ведь делю я все равно не совсем точно. Ага, понятно. Я не должен использовать неточную дробную часть. Это легко сделать так:

$$k = E[n/m]; \quad r = n - mk. \quad (17.3)$$

Тогда в программе должно быть:

ИП1 ИП2 \div П9 КИП9 ИП1 ИП9 ИП2 \times — С/П

Здесь погрешностей не должно быть.

— Вероятно, так, — сказала мама, — если результат деления не очень велик.

На своем примере $256 : 13$ я получил совершенно точно $k = 19$ и $r = 9$.

Тут я вспомнил, что команда КИП9 производит выделение целой части содержимого Р9 только при условии, что оно больше единицы. Будет ли это всегда выполняться? В общем случае нет, если n может быть меньше m . Но в алгоритме Евклида всегда большее число делится на меньшее, так как остаток всегда меньше делителя. Значит, в этой программе можно не опасаться того, что содержимое Р9 окажется меньше единицы. Итак, я смело записываю свою арифметическую программу.

Программа 27. Нахождение общего наибольшего делителя чисел n и m ($n > m$)

ПА С/П ПВ \div П9 КИП9 ИП9 ИПВ ПА
 \times — ПВ ИПА XY x=0 03 ИПА С/П

Инструкция. $n = \text{РХ В/О С/П}$ $m = \text{РХ С/П}$
 $\text{РХ} = \text{ОНД}(n, m)$.

Примеры. Для $n = 1532$, $m = 24$ ОНД = 4, время вычисления — 17 с; для $n = 136\ 833$, $m = 45\ 612$ ОНД = 3, время вычисления — 18 с; для $n = 256\ 300$, $m = 14\ 875$ ОНД = 25, время вычисления — 40 с.

В первой строке программы я вычислил k (в Р9), затем (в начале второй строки) умножил его на m в соответствии с формулами (17.3). При этом я переписал m из РВ в РА, а остаток записал в РВ, что позволяет повторить все вычисления с делним и остатком в соответствии с алгоритмом Евклида. Каждый раз, получая новый остаток, я проверяю, не равен ли он нулю. Если это равенство выполняется, то предыдущий остаток, хранящийся в РА, и является искомым ОНД.

Маме пришлось согласиться, что моя программа вычисляет ОНД правильно и довольно быстро.

— А теперь реши задачу посложнее — разложение натурального числа на простые множители. Надеюсь, ты помнишь, как это делается?

— Конечно. Нужно данное число делить на все простые числа, пока не найдется какое-то, на которое оно разделится. То же самое делать с частным, пока в конце концов не останется тоже простое число. Все это можно сделать на ПМК, если только иметь список простых чисел. Вероятно, это даже проще, чем нахождение ОНД. Здесь ведь не придется искать остаток, нужно только отличать случаи, когда он равен нулю. А это позволит сделать сама мадам Фраксьон.

— Думаю, — сказала мама, — что у тебя все же возникнут определенные трудности. Поэтому я уйду и мешать тебе не буду. А когда будет готово, ты мне покажешь.

Итак, составление этой программы стало для меня делом чести. И я решил приложить все силы, чтобы побыстрее с этим справиться. Для начала нужно все-таки записать алгоритм.

Я несколько раз переписывал его: мне никак не удавалось охватить сразу все возможные соотношения между разлагаемым числом и простыми делителями. Вскоре я сообразил, что если задано число N , то вполне достаточно перепробовать только те простые числа, которые не превышают \sqrt{N} . Если N ни на одно из них не делится, то значит оно само простое число. Действительно, если бы N делилось на какое-то число $m > \sqrt{N}$, то оно должно было

делиться и на N/m , которое меньше \sqrt{N} . Более того, если N уже разделилось на одно или несколько простых чисел и получился резульят N' , то продолжать опробование простых чисел нужно только пока они меньше $\sqrt{N'}$. Если очередное простое число p оказалось равным $\sqrt{N'}$, то значит N делится на p^2 , и на этом разложение заканчивается. Если же оно больше $\sqrt{N'}$, то значит N' — простое число, последнее в разложении числа N .

Исходя из этого, я и построил следующий алгоритм. Это, собственно говоря, не сам полный алгоритм разложения на множители, а только алгоритм одной его части (вспомогательный) — опробование некоторого простого числа p в качестве делителя числа N (или N' , если на некоторые простые числа меньше p мы уже N разделили). Алгоритм решает, во-первых, делится ли N на p , и если да, то сколько раз. Это число раз обозначим α . В список делителей включается p^α . Во-вторых, алгоритм решает, нужно ли еще продолжать вводить новые простые числа p или разложение является законченным. Для этого служит латерная переменная C .

```

алг ДЕЛИТЕЛЬ (нат  $N, N', p, p_i, \alpha$ , лит  $C$ )
    арг  $N, p$ 
    рез  $C, \alpha, N', p_i$ 
    нач нат  $k, x, s$ 
         $N' := N; \alpha := 0; C := \text{«испытание делителя } p\text{»}$ 
        пока  $C := \text{«испытание делителя } p\text{»}$ 
            иц
                если  $\sqrt{N'} = p$ 
                    то  $\alpha := \alpha + 2; p_i := p; N' := 1;$ 
                     $C := \text{«разложение окончено»}$ 
                иначе если  $\sqrt{N'} < p$ 
                    то  $p_i := N'; \alpha := \alpha + 1; N' := 1;$ 
                     $C := \text{«разложение окончено»}$ 
                иначе  $x := N'/p; k := E[x];$ 
                     $s := N' - kp$ 
                    если  $s = 0$ 

```

```

    то  $p_i := p$ ;  $N' := k$ ;
 $\alpha := \alpha + 1$ 
иначе если  $\alpha \neq 0$ 
    то  $p_i := p$ 
все
 $C :=$  «перейти к
испытанию сле-
дующего дели-
теля»
все
все
кон

```

С помощью этого алгоритма я проверяю, делится ли число N' на делитель p , и если да, то сколько раз. Здесь могут быть такие случаи. Если $p > \sqrt{N'}$, то проверять нечего: N' — число простое. Алгоритм выдает это значение единственного оставшегося делителя и сообщает, что разложение окончено. При этом я учитываю, что делители p проверяются в порядке возрастания, так что все простые делители, меньшие p , уже опробованы. Точно так же, если $p = \sqrt{N'}$, то p входит в разложение еще 2 раза (α увеличивается на 2), и разложение окончено. Если же $p < \sqrt{N'}$, то проверяется, делится ли N' на p . Если да, то p входит в число делителей (выдается в качестве результата), и α увеличивается на единицу. При этом сохраняется значение C как «испытание делителя p », и цикл продолжается, а N' принимает значение частного k . Если же N' на p не делится, то C становится «перейти к испытанию следующего делителя», и на этом вспомогательный алгоритм заканчивается. Здесь нужно отметить, что если в результате $\alpha = 0$, то это значит, что p не входит в число делителей числа N .

Имея такой алгоритм и таблицу простых чисел от $p_1 = 2$ до некоторого p_{\max} (так как бесконечную таблицу иметь нельзя), можно составить общий алгоритм разложения на простые множители любого натурального числа $N < p_{\max}^2$. Так, чтобы разлагать на множители все

числа до 10^8 , мне нужна таблица простых чисел от $p_1 = 2$ до $p_{\max} = 9973$, т. е. не так уж много.

На основании этого я составил программу.

Программа 28. Разложение числа $N \leq 10^8$ на простые множители $N = \prod_i p_i^{\alpha_i}$

П1	0	П6	ИПА	Y	ИП1	-	x ≠ 0	43	x ≥ 0
26	ИПА	ИП1	+	П9	КИП9	ХУ	ИП9	-	x = 0
36	ИП9	ПА	КИП6	БП	03	ИП6	x ≠ 0	31	ИП1
C/П 1	ИПА	С/П	1	С/П	ИП6	x ≠ 0	41	ИП1	
C/П 9	С/П	КИП6	КИП6	ИП6	ИП1	БП	33		

Инструкция. $N = PA$, $p_i = RX$ В/О С/П $RX = p_i$, $RY = \alpha_i$ С/П ... Если при остановке $RX = 9$, то ввести следующее значение p_i и нажать В/О С/П; если $RX = 1$, разложение закончено.

Итак, пользователю необходимо иметь таблицу простых чисел и вводить их по очереди. Если при остановке в RX будет введенное число p_i , то это значит, что оно входит в состав простых делителей числа N , а степень, в которую оно входит, находится в RY и может быть прочитана после нажатия клавиши ХУ. Их нужно записать на бумаге. Если же появится число 9 (я нарочно выбрал его, так как это число не простое), то это значит, что p_i не входит в состав делителей и $C :=$ «перейти к испытанию следующего делителя», т. е. нужно ввести следующее простое число и нажать клавиши В/О С/П. Если, наконец, появится 1, то $C :=$ «разложение окончено». Таким образом, числа 1 и 9, которые не могут фигурировать в качестве простых, я использовал для передачи значений литерной переменной C .

Я выбрал для α регистр б с возрастающим содержимым, так как при $s = 0$ нужно увеличивать α на единицу, а это можно сделать за один шаг командой КИП6, вместо четырех команд ИП6 1 + П6. Эту команду я использовал дважды, когда $p = \sqrt{N}$. В общем сделал все для сокращения программы и уложился в 49 шагов.

Когда закончил, папа уже вернулся с работы, и я продемонстрировал программу в действии обоим родителям. Мама не отрицала, что ПМК разлагает числа на множители, но заметила, что с помощью карандаша и бумаги она сделает это быстрее. Я предложил ей соревнование и выдвинул папу в качестве арбитра. Папа выбрал наш семи-

значный номер телефона. Как потом оказалось, он разлагается на четыре простых множителя, из которых один пятизначный — 10639. Поэтому нам обоим пришлось помучиться. Все же с помощью ПМК я справился с этим заданием за 9 мин. Мама упорно продолжала считать с помощью карандаша и бумаги, хотя мы ее уговаривали бросить это пустое занятие. Наконец, спустя 25 мин она дошла до простого числа 107, которое оказалось больше $\sqrt{10639}$, и тем самым доказала, что 10639 — простой множитель, которым заканчивается разложение.

— Ты очень нос не задирай, — сказал мне папа. — В твоей программе есть несколько хороших идей, в частности использование девятки для вызова следующего простого числа. Но чтобы ты понял ее недостаток, реши-ка простой пример. Разложи на множители небольшое пятизначное число 44 521

Я принялся за дело. Прошло 10 мин, потом 20 мин, я вводил все новые и новые простые числа, и ни на одно из них папино число не делилось.

— Ты, по-видимому, подсунул мне простое число? — спросил я у папы.

— Нет, оно составное.

В общем я промучился больше получаса. Оказалось, что $44\ 521 = 211^2$ — квадрат не очень большого простого числа.

— Вот видишь, — сказал папа, — как утомительно вводить одно за другим простые числа, даже имея таблицу. И всегда возникает опасение, не пропустил ли ты случайно одно простое число, которое как раз и является делителем. Это, безусловно, недостаток программы. Я бы предпочел, чтобы разложение длилось дольше, но чтобы числа вводил сам ПМК.

— Но нельзя же засунуть в ПМК список простых чисел.

— Конечно, нельзя. Но нужно найти выход из положения. А выходов всегда бывает несколько. Вот и давай искать наилучший. Прежде всего рассмотрим две крайности — самый примитивный и самый изощренный выход. Самый примитивный — ПМК вместо последовательности простых чисел вводит последовательность всех натуральных чисел от 2 до $E[\sqrt{N}]$.

— Как? Разве это можно? Нам ведь нужно разложить N на простые множители, а мы будем подавать все числа, и простые, и составные.

— Напрасно ты этого боишься. Ведь мы подаем числа в порядке возрастания. Если мы подали некоторое составное число t , которое, например, представляет собой произведение двух простых чисел p_1, p_2 , то раньше мы подавали все меньшие натуральные числа, в том числе p_1 и p_2 . Поэтому, если N делится на t , то оно уже разделилось на p_1 и на p_2 , и поэтому текущее частное уже на t не делится. Следовательно, можно вводить в твою программу все натуральные числа по порядку и все равно она выдаст только простые множители. Другое дело, что такой алгоритм очень невыгоден: он будет работать медленно. Если ты разлагаешь число N порядка 10^4 , то ты должен испытать примерно 100 натуральных чисел до \sqrt{N} , тогда как простых чисел в первой сотне всего 25. Другими словами, такой примитивный алгоритм удлинил вычисления примерно в 4 раза.

Самый изощренный вариант ПМК вводит в твою программу только простые числа. Но списка всех простых чисел в памяти ПМК нет, поэтому в программу включается часть, которая проверяет вводимые числа, пытаясь делить их на меньшие, заведомо простые числа. Например, если мы разлагаем на множители число N , как и в предыдущем примере, порядка 10^4 , то нужно вводить простые числа, меньшие 100. Чтобы убедиться, что число, меньшее 100, простое, достаточно проверить, что оно не делится на 2, на 3, на 5 и на 7. А уж эти четыре простых числа можно хранить в памяти ПМК. Однако, думаю, этот изощренный вариант будет не более быстрым, а может быть, и более медленным, чем примитивный. Ведь довольно много времени должно уходить на проверку простоты каждого вводимого числа. Это время значительно увеличится, если мы захотим разлагать на множители числа не порядка 10^4 , а, скажем, до 10^8 .

Конечно, далеко не всегда истина лежит между двумя крайностями, но в данном случае весьма заманчиво найти какой-то компромиссный вариант, например на вход не подаются некоторые заведомо составные числа, например четные. Специальных проверок не требуется, достаточно числа после числа 3 получать, увеличивая предыдущее на 2. Это значит, что из первой сотни натуральных чисел мы будем испытывать 49 нечетных чисел, среди которых 24 простых и 25 составных.

Несколько лучше вариант — исключить не только четные числа, но и числа, кратные трем. Тогда из первой сотни натуральных чисел вводятся 33, т. е. к 24 нечетным

простым числами примешиваются еще 9 составных: 25, 35, 49, 55, 65, 77, 85, 91 и 95. Такая программа работает примерно втрое быстрее, чем примитивная с вводом всех натуральных чисел. Сейчас я тебе ее покажу, а потом покажу другую программу, над которой я довольно долго проровился и которая исключает из вводимых натуральных чисел все делящиеся на 2,3 или 5. Здесь из первой сотни сохраняются только три составных числа — 49, 77 и 91. Я думаю, что по скорости одна из этих программ наилучшая, но какая — не знаю. Разберись в них, а затем прохронометрируй на разложении разных чисел, больших и малых, с большим и малым числом простых делителей.

Обе эти программы отличаются только вводом делителя. Что же касается алгоритма опробования введенного числа в качестве делителя и определения конца разложения, от он оформляется как подпрограмма. В качестве подпрограммы используем например, твою программу 28, только заменим Р1, в который ты записываешь вводимый делитель, на Р5. Это упростит увеличение его содержимого на единицу — с помощью команды КИП5.

Идея первой из программ ввода заключается в том, что появление в ряду натуральных чисел, кратных 2 и 3, — периодический процесс с периодом 6. Точнее, поскольку число 5 некратно 2 и 3, то следующие числа можно получить по такому закону:

$5 + 2 = 7, 7 + 4 = 11, 11 + 2 = 13, 13 + 4 = 17$
и т. д. Таким образом, разности между двумя вводимыми числами равны 2, 4, 2, 4, 2, 4, и т. д. Вот эта программа.

Программа 29. Разложение числа $N < 10^8$ на простые множители $N = \prod_i p_i^{\alpha_i}$ (вариант с вводом делителей, некратных 2 и 3)

ПА 4	ПД	П2	2	П4 П5	ПП	22	КИП5
ПП 22	5	П5	ПП	22 ИП5	КИПД	ПД	+
БП 13	0	П6	ИПА	γ ИП5	—	$x \neq 0$	63
$x \geq 0$	47	ИПА	ИП5	÷	П9 КИП9	ХУ	ИП9
$x = 0$	57	ИП9	ПА	КИП6	БП 24	ИП6	$x \neq 0$
ИП5 С/П	1	ИПА	С/П	0	С/П	ИП6	52
ИП5 С/П	В/О	ИП6	2	+	ИП5	БП	62
							54

Инструкция. $N = RX$ В/О С/П RX = p_1 ,
 $PY = \alpha_1 C/P, \dots, RX = p_i, PY = \alpha_i C/P, \dots, RX = 0$.

После высвечивания результата записать $p_i^{\alpha_i}$, где p_i находится в РХ, а α_i в РУ. Разложение закончено, если РХ = 0.

Я без особого труда разобрался в этой программе. Начиная с шага 22, идет подпрограмма, которая по существу совпадает с программой 28. Разница в том, что Р1 заменен на Р5, а в конце программы вместо 9, которая использовалась в программе 28 как сигнал в том, что нужно вводить новое число в Р1, здесь стоит команда возврата из подпрограммы, которая и приводит к вводу очередного числа в Р5. Кроме того, разумеется, претерпели изменения адреса переходов, так как добавилось начало программы, содержащее 22 шага. Это начало служит для формирования простых (а иногда не простых, но некратных 2 и 3) чисел.

Сначала в первой строке в Р5 вводится первое простое число 2. После того как оно обработано в подпрограмме, команда КИП5 (шаг 09) увеличивает содержимое Р5 до 3 и затем число 3 обрабатывается в подпрограмме. После этого в Р5 вводится число 5. Далее, все новые числа образуются путем увеличения предыдущего числа на 2, а затем на 4, затем снова на 2 и снова на 4 и т. д.

Для попеременного прибавления то 2, то 4 без затраты большого числа шагов применена такая хитрость. С самого начала в Р2 записано число 4, а в Р4 — число 2. В этих регистрах они и остаются на протяжении всего вычисления. В РД записано тоже число 4. Поэтому первая команда КИПД (шаг 17) вызывает содержимое Р4, т. е. число 2. Это число тут же записывается в РД, а затем прибавляется к числу 5, находящемуся в это время в Р5. Сумма 7 (в результате безусловного перехода на шаг 13) записывается в Р5 и подается на подпрограмму. После обработки числа 7 и вызова его из Р5 (шаг 16) вновь подается команда КИПД. Но в РД теперь записано число 2, следовательно, будет вызвано число из Р2, которое равно 4. Это число 4 будет записано в РД и прибавится к числу 7, находящемуся в Р5, образуя следующее простое число 11. Далее, весь цикл повторяется, и к испытуемому числу попеременно прибавляется 2 или 4. В результате получается последовательность: 13, 17, 19, 23, 25, 29, 31, 35, 37, 41, 43, 47, 49, 53 и т. д. Из этих чисел только 25, 35 и 49 не являются простыми и поэтому занимают лишнее время.

Затем папа показал программу, в которую добавлены операции, обеспечивающие исключение чисел, кратных 5,

т. е. 25, 35, 55 и т. д. По идеи это должно ускорить разложение. Однако сами добавленные операции занимают некоторое время, так что пока не ясно, получится ли ускорение. Период последовательности чисел, не кратных 2, 3 и 5, составляет $2 \cdot 3 \cdot 5 = 30$. Разности между числами (начиная с 7) составляют 4, 2, 4, 2, 4, 6, 2, 6. Для получения такой последовательности используется такой же алгоритм, который в предыдущей программе создает разности: 4, 2, 4, 2, 4, 2, 4, 2, 4, но после подчеркнутых здесь разностей полученное число (кратное 5) на подпрограмму не подается. Для этого используются два счетчика L0 и L1. Подпрограмма здесь такая же, как в программе 29, если не считать изменения адресов переходов.

Программа 30. Разложение числа $N < 10^8$ на простые множители $N = \prod_i p_i^{\alpha_i}$ (вариант с вводом делителей, некратных 2, 3 и 5).

ПА	4	ПД	П2	7	П1	2	П0	П4	П5
ПП	38	КИП5	ПП	38	5	П5	ПП	38	ИП5
КИПД	ПД	+	П5	L1	17	L0	34	2	П0
7	П1	БП	19	3	П1	БП	19	0	П6
ИПА	Y	ИП5	—	x ≠ 0	79	x ≥ 0	63	ИПА	ИП5
÷	П9	КИП9	ХУ	ИП9	—	x = 0	73	ИП9	ПА
КИП6	БП	40	ИП6	x ≠ 0	68	ИП5	С/П	1	ИПА
С/П	0	С/П	ИП6	x ≠ 0	78	ИП5	С/П	В/О	ИП6
2	+	ИП5	БП	70					

Инструкция. Такая же, как и для программы 29.

— Разумеется, — заметил папа, — можно было бы построить программу, которая исключала бы также все числа, кратные 7. Но в этом случае период был бы $2 \cdot 3 \cdot 5 \cdot 7 = 210$. Конечно, такая программа будет более сложной и, несомненно, ввод каждого очередного испытуемого делителя займет больше времени. А экономия, которую это даст, не так велика. В первой сотне она позволит дополнительно исключить только три числа: 49, 77 и 91, во второй — тоже три числа: 119, 133 и 161. Я не сомневаюсь, что такая экономия не окупит дополнительное время, затрачиваемое на получение и ввод делителей. Поэтому на звание наиболее быстрой программы могут претендовать только эти две — 29 и 30, если не считать программы 28, с которой замучишься, вводя по одному все простые числа. Воору-

жись секундомером и проверь, сколько времени затрачивается на разложение разных чисел по каждой из этих программ.

Я так и сделал и получил следующие результаты для разных чисел:

N	Продолжительность разложения по программе	
	29	30
$4565 = 5 \cdot 11 \cdot 83$	1 мин 30 с	1 мин 40 с
$2048 = 2^{11}$	1 35	1 45
$100013 = 103 \cdot 971$	7 25	7 20
$129852 = 2^2 \cdot 3^2 \cdot 3607$	5 20	5 10
$3934272 = 2^6 \cdot 3 \cdot 31 \cdot 661$	4 10	4 05
$3934273 = 7 \cdot 19 \cdot 29581$	12 58	12 10
9967 = 9967 (простое число)	7 56	7 05

Что же можно сказать по этим результатам? Прежде всего, что заметной разницы в быстродействии между этими программами нет. В большинстве случаев разложение по программе 30 проходит немного быстрее и только в самых простых случаях, когда длительность разложения меньше 2 мин, небольшое преимущество имеет программа 29. Однако разница в быстродействии программ нигде не превышает 12 %. Стоит ли ради этого усложнять программу? Мне думается, что лучше воспользоваться более простой программой 29. С этим моим выводом папа согласился.

18

В КАКОЙ ДЕНЬ БЫЛА КУЛИКОВСКАЯ БИТВА?

Вчера, когда мы закончили оценивать программы для разложения на множители, папа взглянул на часы и заметил:

— Сейчас ровно 21 час. Интересно, какое время будет ровно через два часа?

— Разумеется, — ответил я, — 23 часа. Но я не понимаю, на что ты намекаешь?

— А я ни на что не намекаю. Просто немного упражняюсь в арифметике. Ты ведь к числу 21 прибавил 2 и получил 23. Это ведь обычное сложение. А теперь, скажи-ка, который час будет через 10 часов?

— Это тоже нетрудно сосчитать. Через 10 часов будет 7 часов утра.

— А вот это уже не так просто понять. В первом случае ты сосчитал $21 + 2 = 23$ в соответствии с правилами арифметики. Но если ты проделаешь то же самое, отвечая на мой второй вопрос, то получишь $21 + 10 = 31$, а у тебя почему-то получилось $21 + 10 = 7$. Что же это за арифметика?

— Ты, папа, хочешь меня запутать. Ведь в сутках 24 часа, значит, 31 час быть не может. Когда наступает 24 часа, это то же самое, что 0 часов следующего дня, и все начинает считаться заново.

— Значит, в каком-то смысле, когда речь идет о часах, 31 и 7 — это одно и то же?

— Почему же одно и то же? Просто 31 часа не бывает, а бывает 7 часов.

— Но я ведь могут сказать, что через 10 часов будет такое же время, какое было 14 часов тому назад. Да и через 34 часа будет тоже такое время: 7 часов.

Я подумал и согласился с этим.

— Вот мы и говорим, — продолжил папа, — что числа -14 , $+10$ и $+34$ в каком-то смысле похожи, когда речь идет об измерении времени в пределах суток, состоящих из 24 часов. Мы не будем говорить, что эти числа равны. Математики предпочитают говорить, что они сравнимы по модулю 24. Точно так же числа 1, 25, 49 тоже сравнимы по модулю 24. Записывается это так:

$$-14 \equiv 10 \equiv 34 \pmod{24}, \quad 1 \equiv 25 \equiv 49 \pmod{24}. \quad (18.1)$$

Я могу также записать такие сравнения:

$$72 \equiv 48 \equiv 24 \equiv 0 \pmod{24}. \quad (18.2)$$

Ты с этим согласен?

Я сообразил, что и через 24 часа и через 48 или 72 часа часы будут показывать точно такое же время, как и сейчас. И я обратил внимание на то, что все числа в (18.2) делятся без остатка на 24.

— Правильно подметил, — сказал папа. — А теперь попробуй найти нечто общее между числами 1, 25 и 49, за-

писанных во втором примере наших сравнений. Попробуй делить их на 24.

Я сразу же увидел, что все они дают при делении на 24 остаток 1. А числа 10 и 34 дают при делении на 24 остаток 10.

— Значит, можно сказать, что сравнение

$$a \equiv b \pmod{m} \quad (18.3)$$

означает, что числа a и b при делении на m дают одинаковый остаток. Этот остаток называют вычетом числа a (или b) по модулю m . Такая «арифметика сравнений» находит довольно широкое применение в разных областях математики и прикладных наук. Например, очень часто приходится среди натуральных чисел различать четные и нечетные. Попробуй записать в форме сравнений, что a — четное, а b — нечетное число.

Я довольно быстро сообразил, что это можно записать так:

$$a \equiv 2 \pmod{2}, \quad b \equiv 1 \pmod{2}. \quad (18.4)$$

— Правильно, — сказал папа. — Только вместо $a \equiv 2 \pmod{2}$ удобнее записать $a \equiv 0 \pmod{2}$. Очень многие свойства равенств можно распространить и на сравнения. Например, сравнения по одному модулю можно складывать или перемножать, можно также возводить в степень. Поскольку в ряде задач приходится вычислять вычеты разных натуральных чисел по некоторому модулю, то неплохо иметь программу, позволяющую это сделать на ПМК. Давай-ка составь ее.

— Пожалуйста. Эт ведь то же самое, что деление с остатком, только частное нам не нужно и можно вывести один остаток. Вот что получится.

Программа 31. Вычисление вычета a натурального числа n по модулю m

П2 ИП1 ÷ П9 КИП9 ИП2 ИП1 ИП9 × —

С/П

Инструкция. ($m = P1$) $n = RX$ В/О С/П RX = a.

— Хорошо, — сказал папа, — только добавь примечание $n > m$. Если это условие не выполнено, то программа не будет работать. Ведь при этом в Р9 запишется

число, меньшее единицы, и команда КИП9 не вызовет выделения целой части. Поэтому твою программу 31 можно использовать только, если заведомо $n > m$. Если такой уверенности нет, приходится применять более сложную программу, добавив к частному единицу и воспользовавшись регистром с убывающим содержимым.

Программа 32. Вычисление вычета a натурального числа n по модулю m

П2 ИП1 \div 1 + П3 КИП3 ИП2 ИП1 ИП3
× — С/П

Инструкция. Та же, что к программе 31.
Программа пригодна и при $n < m$.

Здесь к результату деления мы прибавляем единицу, и так как $n/m > 0$, то $n/m + 1 > 1$. Затем используем регистр Р3 с убывающим содержимым. Поэтому целая часть числа $n/m + 1$ уменьшится на единицу, и мы получим $E[n/m]$. А дальше вычисляется остаток, т. е. вычет.

— А можно сделать и по-другому. Вместо того чтобы прибавлять единицу к результату деления, можно прибавить делитель к делимому:

П2 ИП1 + Вх \div П3 КИП3 ИП2 ИП1 ИП3
× — С/П

— Да, — ответил папа. — Эта программа тоже возможна. Число шагов в обеих этих программах одинаковое. Их можно считать эквивалентными. Хорошо, — продолжил он, — тебе ясно, как находить вычеты по различным модулям. Ответь с помощью ПМК на такие вопросы: В какой день происходила Куликовская битва? В какой день была провозглашена Парижская Коммуна? В какой день вышли декабристы на Сенатскую площадь?

— Разве для этого нужен ПМК? Я лучше воспользуюсь энциклопедией.

— Попробуй, только я не уверен, что она тебе поможет.

Я порылся в энциклопедии, выписал все данные и победно сообщил:

— Куликовская битва произошла 8 сентября 1380 г., восстание декабристов — 14 декабря 1825 г., а Парижская Коммуна провозглашена 18 марта 1871 г.

— Эти даты я помню. Нас в школе хорошо учили истории. Номеня интересовало, в какие дни недели произошли эти события. В энциклопедии ты этого не найдешь.

— Теперь понятно. Ты хочешь сделать из ПМК календарь?

— Да, и к тому же вечный. Пригодный для любого года, по крайней мере нашей эры, как по юлианскому, так и по григорианскому летоисчислению. Представь, ты вводишь в ПМК дату, месяц и год, и он выдает тебе день недели в цифровом коде — воскресенье 0, понедельник 1, вторник 2 и т. д.

— Так это ведь вычеты по модулю 7 количества дней, прошедших от первого воскресенья нашей эры.

— Совершенно верно. Надо подсчитать это количество дней. Попробуй выразить его формулой отдельно для юлианского и для григорианского календаря. Кстати, ты помнишь, в чем между ними различие?

— Конечно, помню. Но лучше будет, если ты мне напомнишь.

— Юлианский календарь (или старый стиль) был введен незадолго до начала нашей эры, при Юлии Цезаре. В нем средняя длительность года равна 365,25 сут. Поэтому во избежание накопления погрешности предусмостили увеличение длительности каждого четвертого года на сутки. Годы, номера которых делятся на 4, называются високосными и содержат по 366 дней, а остальные — по 365 дней.

Григорианский календарь (новый стиль) был введен в католических странах указом (буллой) папы Григория XIII в 1582 г., в большинстве протестантских стран (Германия, Англия, скандинавских странах) — в 1700 г., а у нас — в 1918 г. При его введении учили, что длительность астрономического года немного короче 365,25 сут, так что по юлианскому календарю за 400 лет набегает лишних трое суток, которые нужно выбросить из календаря. К концу 16 в. накопилось 10 таких лишних суток, и на эту величину сразу внесли поправку, а в дальнейшем годы, кончающиеся двумя нулями, т. е. с нулевым вычетом по модулю 100, которые по юлианскому календарю, конечно, високосные, объявлены в григорианском календаре простыми, если только их вычет по модулю 400 не равен нулю.

Таким образом, год 1600 был високосным, а годы 1700, 1800 и 1900 по григорианскому календарю были простыми. С каждым таким годом отставание юлианского календаря

от григорианского увеличивалось на один день, и поэтому оно равнялось 10 суткам в 16 и 17 вв., 11 суткам в 18 в., 12 суткам в 19 в. и 13 суткам в 20 в. Такая же разность в 13 суток сохранится в 21 в., так как 2000 г. будет високосным по обоим календарям.

Теперь давай составим формулу для вычисления количества дней, прошедших, скажем, с начала нашей эры, с 1 января 1 года по юлианскому календарю, включая и самый этот день 1 января 1 года. Будем обозначать дату A , номер месяца B и номер года C . Определим сначала, сколько дней прошло до конца (31 декабря) года C . Очевидно это будет $365C + E[C/4]$. Не так ли?

— Да, это понятно. Если бы не было високосных годов, то прошло бы $365C$ дней. А второй член добавляет по одному дню на каждый високосный год, количество которых — это целая часть от четверти всех прошедших годов.

— Попробуй теперь вычислить, сколько дней прошло не до 31 декабря, а до A -го числа B -го месяца.

— Это очень трудно. Ведь месяцы имеют неодинаковое число дней.

— Давай попытаемся. Обозначим через l_i количество дней в i -м месяце. Например, $l_1 = 31$, $l_2 = 28$ (если год невисокосный), $l_3 = 31$, $l_4 = 30$ и т.д. Тогда, чтобы определить количество дней, прошедших до начала B -го месяца, нужно из найденной выше величины отнять сумму всех l_i начиная от $i = B$ и до $i = 12$. Это, как ты знаешь, записывается так: $\sum_{i=B}^{12} l_i$.

— Ясно. А дальше тоже понятно. К этому нужно прибавить еще A — число дней, прошедших с начала месяца. Таким образом, количество дней, прошедших с начала нашей эры до данной даты, равно

$$N = 365C + E[C/4] - \sum_{i=B}^{12} l_i + A. \quad (18.5)$$

— Прекрасно, — сказал пapa. — Только нужно помнить две вещи. Во-первых, это формула для даты по юлианскому календарю. Во-вторых, из величин l_i одна, а именно l_2 , зависит от C . Если C кратно четырем, то $l_2 = 29$, а если не кратно, то 28. Поэтому, когда будем составлять программу, прежде всего нужно выяснить, является ли год C високосным, а затем уже считать сумму. Но эту формулу можно еще немного упростить. Ведь нам нужно

не само число N , а его вычет по модулю 7. А так как для сравнений справедливы (с небольшими оговорками) те же законы, что и для равенств, то можно вычет суммы заменить суммой вычетов (или вычетом суммы вычетов), вычет произведения — произведением вычетов и так далее. Будем обозначать через $(x)_m$ вычет числа x по модулю m . Тогда формулу (18.5) можно заменить следующей:

$$(N)_7 = \left((365)_7 C + E [C/4] - \sum_{i=B}^{12} (l_i)_7 + A \right)_7. \quad (18.6)$$

Здесь я сделал не все возможные замены вычетами, а только те, которые мне нужны для упрощения программы. Далее, как ты можешь легко убедиться, $(365)_7 = 1$. Величины l_i принимают значения 28, 29, 30 и 31, а их вычеты соответственно 0, 1, 2 и 3.

Составим теперь программу для вечного юлианского календаря. В этой программе нам нужно будет делить с остатком на 4 и на 7. Поэтому деление с остатком удобно выделить в подпрограмму, вводя делитель 4 или 7 каждый раз в какой-либо регистр, скажем в Р4.

Самая интересная задача в этой программе — вычисление суммы $\sum_{i=B}^{12} (l_i)_7$. Здесь все слагаемые (кроме $(l_2)_7$) — постоянные, верхний предел суммы тоже постоянный (12), а вот нижний предел B определяется номером месяца. Такие задачи встречаются очень редко. В нашем случае такую программу нетрудно построить, пользуясь косвенной адресацией безусловного перехода. Я сейчас покажу тебе, как это делается на довольно общем примере.

Пусть нам нужно вычислить сумму $\sum_{i=t}^n a_i$, где все a_i — однозначные постоянные целые числа, n — постоянная, а t может принимать значения от 1 до n . Запишем в программу числа a_i так, чтобы адрес, по которому находится a_i , равнялся $2i$. Таким образом, a_1 будет находиться на шаге 02, a_2 — на шаге 04, ..., a_8 — на шаге 16, ..., a_n — на шаге $2n$. Разумеется, на Б3-34 это возможно, если $n < 49$. В нашем случае $n = 12$, так что трудностей не возникает. Шаги с нечетными адресами между a_i заполним командами сложения. Получится такой фрагмент начала программы:

$$\dots a_1 + a_2 + a_3 + a_4 + a_5 + a_6. \quad (18.7)$$

Число $2m$ введем, скажем, в РД. В том месте программы, где нужно будет вычислять нашу сумму начиная с a_m и кончая a_n , запишем команду КБПД. Тогда программа перейдет на шаг $2m$, где записано число a_m , и пойдет дальше по программе, складывая a_m с a_{m+1} , a_{m+2} и так далее до a_n .

Сейчас уже время позднее, а ложиться спать, так и не узнав, в какой день была Куликовская битва, не хочется. Давай вместе составим программу юлианского вечного календаря, а универсальную программу для юлианского и григорианского календарей ты уж составишь сам.

Для этого на шагах 02, 04, 06, ..., 24 должны стоять значения $(l_i)_7$. Поэтому основную часть вычисления можно начинать только на более поздних шагах, а программу придется начинать с безусловного перехода на какой-то шаг, дальше 24. Кроме того, в отличие от (18.7) будем числа a_i не прибавлять, а вычитать, как это следует из (18.6). Те команды, которые нам пока не ясны, заменим многоточиями. Итак,

```
БП ... 3   — ... — 3   — 2   —  
3   — 2   — 3   — 3   — 2   —  
3   — 2   — 3   — БП ... ПС 4  
П4 ПП ...
```

Здесь запрограммировано вычисление суммы $(l_i)_7$ со знаком минус и запись числа 4 в Р4 для нахождения значения С по модулю 4. Деление с остатком на число, записанное в Р4, мы выносим в подпрограмму.

В инструкции мы запишем: data = РА, месяц = РВ год = РХ В/О С/П РХ = день недели. Можно было, конечно, ввести год не в РХ, а в РС, и этим сэкономить один шаг программы. Но опыт показывает, что нетерпеливые вычислители часто забывают ввести в регистр памяти последнее из входных данных, так что надежнее это поручить программе. Поэтому с пуском программы мы должны передать управление на шаг, где записана команда ПС. Для этого вместо первого многоточия поставим его адрес, второе многоточие на месте $(l_2)_7$, заполним числом, вызываемым из некоторого регистра, например Р6. А в этот регистр запишем 0, если год простой, или 1, если год високосный, что мы сделаем сразу после вычисления $E [C/4]$.

Теперь подумаем о подпрограмме. Она должна находить вычет, а так как нас интересует также частное от деления с остатком, то это сводится к делению с остатком. Для этого можно воспользоваться нашими программами 31 или 32. Желательно, конечно, применить более короткую программу 31, главным образом чтобы сократить длительность вычисления. Но это можно сделать, только если $n > m$, т. е. число, вводимое в подпрограмму, больше модуля. В нашем случае это будет обеспечено, если, во-первых, $C > 4$ и, во-вторых, в (18.6) $N > 7$. Легко убедиться, что оба эти условия будут выполнены, если $C > 40$. Поэтому мы можем сократить программу при условии, что будем интересоваться только датами не ранее 40 г. н. э. Пойдем на такое ограничение?

— Ладно, пожертвуем первыми 40 годами.

— Это тем более оправдано, что в древности не было общепринятого деления времени на недели. Оно возникло только с распространением христианства на западе (примерно при $C > 300$) и ислама на востоке (при $C > 600$), так что определение дня недели при $C < 40$ вряд ли представляет интерес.

Таким образом, можно составить подпрограмму в таком виде:

ПД ИП4 \div П7 КИП7 ИПД ИП7 ИП4 \times —
Здесь делимое записывается в РД, а делитель (модуль), как мы условились, записан в Р4. Частное от деления с остатком оказывается записанным в Р7, а сам остаток (вычет) — в РХ.

Теперь, основываясь на (18.6) и на составленной подпрограмме, можно записать всю программу.

Программа 33. Вечный юлианский календарь

БП	29	3	—	ИП6	—	3	—	2	—
3	—	2	—	3	—	3	—	2	—
3	—	2	—	5	—	ПП	55	С/П	ПС
0	П6	4	П4	ИПС	ПП	55	x--0	41	1
П6	ИП7	ИПС	+	ИПА	+	7	П4	ХУ	ИПВ
2	\times	ПД	ХУ	КБПД	ПД	ИП4	\div	П7	КИП7
ИПД ИП7 ИП4 \times — В/О									

И н с т р у к ц и я. Дата = РА, месяц = РВ, год = =РХ В/ОС/П РХ = день недели (0—воскресенье, 1—понедельник, ..., 6 — суббота).

Здесь я немного забежал вперед. Дело в том, что мы с папой записали сначала программу не совсем в том виде, в каком она приведена здесь. На шаге 24, соответствующем 12-му месяцу — декабрю, имеющему 31 день, было записано, как и полагается по нашему алгоритму, число 3 = (31)₇. Однако, набрав программу, папа заметил, что она пока что будет вычислять вычет по модулю 7 от числа дней, прошедших с 1 января 1 года (включительно), и этот вычет совпал бы с днем недели, если бы первый день нашей эры был понедельником. Но мы вовсе не уверены, что это так.

— Не беда, — сказал папа, — сейчас мы это узнаем и введем соответствующую поправку в нашу программу. Вспомни какое-либо историческое событие, о котором известно, в какой день недели оно произошло.

Мне сразу пришло в голову 9 января 1905 года — «кровавое воскресенье». Мы ввели эти данные в ПМК и получили в ответе 2 вместо 0.

— Вот видишь, нам микрокалькулятор поставил двойку за то, что мы не внесли поправку. Очевидно, нужно уменьшить число дней (или вычет) на 2, т. е. считать не от 1 января, а от 3 января 1 г. (включительно), когда и был понедельник. А 1 января 1 года, значит, была суббота. Чтобы ввести эту поправку, удобнее всего изменить величину l_{12} , которая входит в формулу (18.6) при любом значении B . Поэтому вместо $(l_{12})_7 = 3$ запишем на шаге 24 число 5.

Так мы и сделали и получили приведенную выше программу 33.

Конечно, мы сразу ввели данные для Куликовской битвы — 8.09. 1380 г. и нашли в ответе 6. Значит, Куликовская битва произошла в субботу. Далее мы ввели 25.10.1917 г. — день Великой Октябрьской социалистической революции (по старому стилю) — и убедились, что она произошла в среду. Восстание декабристов (14.12. 1825 г.) — в понедельник. На каждое вычисление у нас уходило 25 ... 30 с.

— Пока на этом закончим. Можно, конечно, составить аналогичную программу для григорианского календаря. Но я надеюсь, что ты сделаешь это сам.

Я не стал ждать завтрашнего дня. Как же сделать универсальную программу вечного юлианского и григориан-

ского календаря? Пожалуй, лучше всего взять за основу уже готовую программу 33 для юлианского календаря и добавить к ней перевод из григорианского в юлианский. Это ведь совсем не трудно. Нужно прежде всего определить, какой нынче век, точнее нужно найти целую часть о делении номера года C на 100. Для этого можно воспользоваться той же подпрограммой, которая уже имеется. В соответствии с полученным результатом и с правилами определения високосного года по григорианскому календарю я построил таблицу перевода G даты григорианского календаря в юлианский. Этую величину G нужно вычесть из григорианской даты, чтобы получить юлианскую. Таблица эта получилась такая:

$E[C/100]$	G	$E[C/100]$	G
15	10	20	13
16	10	21	14
17	11	22	15
18	12	23	16
19	13	24	16

Эту таблицу я составил начиная с 16 в., когда был введен григорианский календарь, до 25 в.

Я старался выразить зависимость G от C в этой таблице простой формулой, учитывая, что G увеличивается на единицу всегда, когда $(C)_{100} = 0$, если при этом $(C)_{400} \neq 0$. В результате я пришел к формуле

$$G = E[C/100] - E[C/400] - 2. \quad (18.8)$$

Можно легко проверить, что данные таблицы целиком совпадают с этой формулой. Так что если вычислить G и вычесть его из N , то я фактически перейду от григорианского календаря к юлианскому и смогу далее считать по уже готовой программе 33.

Я еще раз проверил ход своих рассуждений, чтобы убедиться в отсутствии ошибок. И тут заметил, что не учел еще одного обстоятельства. Конечно, G я вычисляю по формуле (18.8) правильно и весь мой алгоритм годится, если $(C)_{100} \neq 0$. Если же $(C)_{100} = 0$ и при этом $(C)_{400} \neq 0$,

то необходимо учесть, что этот год простой, тогда как по юлианскому календарю он високосный. Значит, по программе 33 в Р6 будет записана 1, а нужно записать 0. Разумеется, это обстоятельство будет вносить ошибку только в тех редких случаях, когда наше событие произошло в январе или феврале года, оканчивающегося двумя нулями, причем число сотен на 4 не делится. Но для универсального вечного календаря нельзя допускать и редких ошибок. Поэтому нужно ввести в программу дополнительную проверку на високосность годов, для которых $(C)_{100} = 0$, и в соответствии с этим посыпать в Р6 или 0, или 1.

Нужно еще придумать, как сообщить ПМК, по какому календарю он должен считать. Это можно сделать, введя условный код в какой-нибудь регистр, например в Р0. Пусть для григорианского календаря в Р0 записывается число 100. Его в дальнейшем можно будет использовать для вычисления $E[C/100]$. А для юлианского календаря в Р0 запишем 0. Это и будет признаком для ветвления.

Таким образом, я составил следующую программу.

Программа 34. Вечный универсальный календарь (юлианский и григорианский)

БП	29	3	—	ИП6	—	3	—	2	—
3	—	2	—	3	—	3	—	2	—
3	—	2	—	5	—	ПП	86	C/P	ПС
0	П6	4	П4	ИПС	ПП	86	x=0	41	1
П6	ИП7	ИПС	+	ИПА	+	П9	ИП0	x≠0	78
П4	ИПС	ПП	86	П2	4	П4	ИП7	П8	ПП
86	П3	ИП9	ИП8	ИП7	—	2	—	—	П9
ИП2	x=0	78	ИП3	x≠0	78	0	П6	7	П4
ИПВ	2	×	ПД	ИП9	КБПД	ПД	ИП4	÷	П7
							КИП7	ИПД	ИП7
									ИП4
									×
									—
									В/О

Инструкция. (0 = Р0 для юлианского календаря или 100 = Р0 для григорианского) дата = РА, месяц = РВ, год = РХ В/О С/П РХ = день недели. Для юлианского календаря программу можно применять начиная с 40 г. н.э., а для григорианского — с 401 г. н. э., хотя в действительности раньше 1583 г. григорианского календаря не существовало.

Примеры. Для юлианского календаря 25.10.1917 (Великая Октябрьская социалистическая революция) — среда; 14.12.1825 (восстание декабристов) — понедельник; 29.01.1837 (смерть Пушкина) — пятница; 8.09.1380 (Куликовская битва) — суббота. Время вычислений — около 25 с.

Для григорианского календаря 9.05.1945 (День Победы) — среда; 14.07.1789 (взятие Бастилии) — вторник; 18.03.1871 (провозглашение Парижской Коммуны) — суббота; 26.07.1953 (штурм казармы Монкадо) — воскресенье; 1.01.2001 (начало 21 в. и 3-го тысячелетия нашей эры) — понедельник. Продолжительность вычислений — около 50 с.

Когда я просчитывал эти примеры, у меня было такое чувство, будто я стал лучше понимать смысл всех этих событий. Конечно, я подсчитал также, в какой день недели я родился. Оказалось — в четверг.

Но программу нужно представить на родительский суд. А для этого необходимо пояснить смысл производимых в ней операций. Попробую это сделать. Это должно быть не очень трудно, так как значительная часть программы 34 совпадает с «юлианской» программой 33

Действительно, до шага 46 полностью повторяется программа 33, если не учитывать изменений адресации подпрограммы. Сама подпрограмма, начинающаяся с шага 86, полностью совпадает с подпрограммой программы 33, начинающейся там с шага 55. Если в программе 34 вводится дата по юлианскому календарю, обе программы почти полностью совпадают. При этом в Р0 записан 0 и после шага 45 управление передается шагу 78, который соответствует шагу 46 программы 33.

Некоторые различия между двумя программами вызваны тем, что в программе 33 сумма $C + E [C/4]$ не записывается в регистр памяти, а хранится в операционном стеке. Из-за этого в программу 33 введены команды XY на шагах 48 и 53. В программе 34 это оказалось неудобным: за время хранения этой суммы необходимо для григорианского календаря вычислять G. Поэтому я предпочел занять еще один регистр памяти Р9 и записать в него эту сумму (шаг 46), прежде чем осуществлять ветвление (шаги 47 ... 49). Для юлианского календаря в Р9 так и остается эта величина, а для григорианского календаря из нее вычитается G (шаги 62 ... 68) и полученная разность снова записывается в Р9, что позволило начиная с шага 78 унифицировать оба случая.

Шаги с 50 по 61 с двумя обращениями к подпрограмме позволяют, во-первых, вычислить $E[C/100]$ (записываемое в Р8) и $E[C/400]$ (в Р7), используемые далее для вычисления G , и, во-вторых, найти вычеты $(C)_{100}$ (записываемое в Р2) и $(C)_{400}$ (записанное в Р3). Эти вычеты нужны для определения високосности в тех случаях, когда $(C)_{100} = 0$, что проверяется на шаге 71. Если при этом $(C)_{400} = 0$ (проверяется на шаге 74), то год является високосным и в Р6 остается 1 записанная туда на шаге 40. Если же $(C)_{100} = 0$, а $(C)_{400} \neq 0$, то условный переход на шаге 74 не сработает и на шаге 77 в Р6 запишется 0.

Вот с таким комментарием я отправился к папе. Он тщательно проверил мою программу и, кажется, остался вполне доволен.

— Простые программы, особенно развлекательного или тренировочного характера, ты уже можешь составлять. Пора заняться серьезными делами. Завтра начнем решать уравнения.

— Какие? — спросил я. — Квадратные?

— Зачем квадратные? — возразил папа. — Для квадратного уравнения ты, конечно, без всякого напряжения к завтрашнему дню подготовишь простенькую программу. А займемся мы более интересными делами. Будем решать алгебраические уравнения высоких степеней, а также трансцендентные, т. е. не алгебраические, уравнения, на первых порах с одним неизвестным. Здесь без ПМК или ЭВМ очень трудно обойтись.

— А это должно быть интересно, — подумал я.

19

СМОТРИ В КОРЕНЬ!

К сегодняшней встрече с папой я подготовил программу для решения квадратного уравнения

$$ax^2 + bx + c = 0. \quad (19.1)$$

Формула для нахождения корней этого уравнения всем известна *

$$x_{1,2}^* = (-b \pm \sqrt{b^2 - 4ac}) / 2a. \quad (19.2)$$

* Для того чтобы отличать корни от других значений x , мы будем помечать их звездочкой (x^*).

Но я уже привык к мысли, что прежде чем составлять по формуле алгоритм и программу, нужно посмотреть, нельзя ли эту формулу упростить. Не лучше ли сначала разделить b на $2a$ и c на a : $\beta = b/2a$; $\gamma = c/a$. Тогда

$$x_1^*, x_2^* = -\beta \pm \sqrt{\beta^2 - \gamma}. \quad (19.3)$$

Я попробовал составить программы по (19.2) и (19.3) и оказалось, что вторая из них чуть-чуть проще. Поэтому я подготовил следующую программу.

Программа 35. Решение квадратного уравнения $ax^2 + bx + c = 0$

ПА С/П ХY	÷ 2	÷ ПВ x ²	С/П ИПА
÷	—	✓ ПД ИПВ	— ↑ ИПД 2 ×
— С/П			

Инструкция. $a = RX$ В/О С/П, $b = RX$ С/П, $c = RX$ С/П $RX = x_1^*$, $RY = x_2^*$. Если высвечивается ЕГГОГ, то уравнение корней не имеет.

Примеры. $x^2 + x - 2 = 0$; $x_1^* = -2$; $x_2^* = 1$. Время вычисления — примерно 20 с (с вводом коэффициентов). $x^2 + x + 2 = 0$ — корней нет.

Такую программу я представил папе.

— С одной стороны, в твоей программе есть много хорошего, — сказал он. — Хорошо, что ты вводишь коэффициенты в RX, а не прямо в регистры памяти. Это более надежно: при вводе в регистры вручную можно ошибиться. Хорошо, что, остановив ПМК, корни выдаешь ты сразу: один в RX и другой в PY. Но, с другой стороны, здесь как нельзя ярче выявилось различие между математикой и школьной математикой. В настоящей математике всякое алгебраическое уравнение имеет корень в области комплексных чисел. В этом даже заключается основная теорема алгебры. И из этого вытекает, что уравнение n -й степени имеет ни больше, ни меньше n корней. Но дело в том, что не все эти корни — действительные числа, среди них могут быть и комплексные. А так как в школе комплексных чисел не проходят, то рассматривают только корни в области действительных чисел и поэтому признают существование уравнений без корней.

— Но это же правильно, папа, — возразил я. — Зачем считать корни, если они все равно не действительны? У меня есть прошлогодний абонемент на стадион, который

в этом году уже недействителен. Так это все равно, как если бы у меня его не было.

— Знаешь ли, — рассердился пapa, — только при твоем дремучем невежестве можно говорить такие вещи и сравнивать прошлогодний абонемент с комплексными числами. Комплексные корни уравнений играют не меньшую роль в приложениях, чем действительные. А иногда и еще большую. Так, в теории колебаний, изучающей огромное количество научных и технических проблем, от механических часов до радиопередатчиков, от изменений в популяции волков и зайцев до работы лазера, именно комплексные корни так называемых характеристических уравнений определяют параметры колебаний — их частоту, нарастание или затухание амплитуды. А случай действительных корней соответствует отсутствию колебательного процесса.

Ну, ладно. Это все ты будешь изучать в вузе, если поступишь. А пока станем на школьную точку зрения и будем считать корнями только действительные корни или, как их еще называют, вещественные. Тем более, что очень часто встречается задача, которую математик сформулирует так: найти корни данного уравнения в области действительных чисел. Мы же будем в соответствии со школьными традициями говорить просто: найти корни уравнения или решить уравнение.

Так вот, даже с этими оговорками твоя программа, как и многие другие, опубликованные в книгах, обладает некоторым недостатком, который можно, если не устраниТЬ, то во всяком случае смягчить. Этот недостаток я тебе продемонстрирую на примере. Реши-ка такое уравнение $x^2 - 20\,000x + 5 = 0$.

— Пожалуйста.

Я ввел $a = 1$, $b = -20000$, $c = 5$ и получил на индикаторе $x_1^* = 0$, а нажав клавишу XY — второй корень $x_2^* = 20000$.

— Значит, — сказал я, — один корень нулевой, а второй равен 20000.

— А теперь подставь свои корни в уравнение и посмотри, получишь ли ты нуль.

Я подставил оба корня и вручную вычислил, что в обоих случаях левая часть дает не нуль, а 5.

— Это число, которое мы получаем вместо нуля, когда подставляем вычисленный корень в уравнение, называют невязкой. Так вот, не кажется ли тебе, что невязка здесь великовата? Значит, программа вычислила кор-

ни недостаточно точно. Собственно говоря, дело даже не в программе, а в алгоритме. Погляди-ка на свои формулы и подумай, в чем дело.

Я посмотрел на формулу (19.3) и сразу же вспомнил материнские наставления по поводу алгоритмов.

— Все понятно, — сказал я. — В твоем уравнении $\gamma \ll \beta^2$, и поэтому $\sqrt{\beta^2 - \gamma}$ очень мало отличается от $|\beta|$. Поэтому, когда мы вычисляем x_1^* , происходит вычитание двух очень близких друг другу чисел. Поэтому относительная ошибка, имевшаяся в этих операндах, может очень сильно возрасти.

— Совершенно верно. Здесь в данном случае ты получил просто машинный нуль, который и принял за корень уравнения.

— Как же быть?

— Нужно применить другой алгоритм для вычисления меньшего по абсолютному значению корня. Когда ты вычисляешь больший по абсолютному значению корень, который при $\gamma \ll \beta^2$ близок к 2β , то близкие по значению числа не вычтутся, а складываются, и поэтому относительная погрешность не возрастает. А меньший по абсолютному значению корень будем вычислять другим способом.

Напомню, что произведение двух корней уравнения $x_1^* x_2^*$ равно γ . В этом легко убедиться. Из (19.3) следует

$$\begin{aligned}x_1^* x_2^* &= [-\beta + \sqrt{\beta^2 - \gamma}] [-\beta - \sqrt{\beta^2 - \gamma}] = \\&= \beta^2 - (\sqrt{\beta^2 - \gamma})^2 = \beta^2 - \beta^2 + \gamma = \gamma,\end{aligned}\tag{19.4}$$

а отсюда следует, что

$$x_2^* = \gamma / x_1^*. \tag{19.5}$$

Поэтому составим программу, в которой больший по абсолютному значению корень вычисляется по твоей программе, а меньший — по (19.5).

Программа 36. Решение квадратного уравнения с повышенной точностью

ПА С/П ХY ÷ 2 ÷ ПВ x² С/П ИПА
÷ ПС — γ ИПВ x<0 23 — ↑ ИПС
ХY ÷ СП + ↑ /-/ БП 18

Инструкция. Та же, что и к программе 35.

Примеры.

$$x^2 + x - 2 = 0; \quad x_1^* = 1, \quad x_2^* = -2;$$

$x^2 + x + 2 = 0$; корней нет;

$$x^2 - x - 2 = 0; \quad x_1^* = -1, \quad x_2^* = 2;$$

$$x^2 - 20\,000x + 5 = 0; \quad x_1^* = 2,5 \cdot 10^{-4}; \quad x_2^* = 20\,000;$$

$$2x^2 + 1000x + 1 = 0; \quad x_1^* = -1,000002 \cdot 10^{-3};$$

$$x_2^* = -499,99899.$$

В этой программе ветвление на шаге 15 служит для того, чтобы найти сначала корень, имеющий наибольшее абсолютное значение. Очевидно, что при $\beta < 0$ это будет $-\beta + \sqrt{\beta^2 - \gamma}$, а при $\beta > 0$ он равен $|\beta + \sqrt{\beta^2 - \gamma}|$. Команда \uparrow (шаг 24) служит для того, чтобы в ПМК первых выпусков сработала правильно команда $/-$.

— Ладно, — сказал папа, — будем считать наши дела с квадратными уравнениями законченными. Займемся алгебраическими уравнениями более высоких степеней.

— Как же мы будем их решать? В школе мы этого не проходили. Правда, я слышал кое-что о формуле Кардано для уравнений 3-й степени, но саму эту формулу не знаю.

— Я ее тоже не знаю. Знаю, что для уравнений 5-й и более высоких степеней таких формул нет.

— Неужели до сих пор математики их не вывели?

— Математики сделали значительно больше: они доказали, что таких формул вообще не существует.

— Как же решать такие уравнения?

— Известны различные численные методы, которые сводятся к поиску корней и последовательным приближениям. Эти методы хорошо разработаны и с помощью ЭВМ или ПМК позволяют находить с любой точностью интересующие нас корни. Они применимы и к трансцендентным, т. е. не алгебраическим, уравнениям. Более того, даже уравнения 3-й и 4-й степеней, для которых существуют формулы, позволяющие найти все корни, предпочтитаю решать также численными методами. С наиболее простыми из этих методов я тебя познакомлю, а ты уж постараешься научить свой ПМК решать разные уравнения, вернее помогать тебе в их решении. Участие головы в этом весьма желательно, а подчас и необходимо. Я тебе расскажу об общих принципах и методах, относящихся к решению любых уравнений с одним неизвестным, как алгебраических, так и трансцендентных.

Следует твердо запомнить: прежде чем искать действительные корни уравнения, необходимо убедиться, что они существуют, а также определить, сколько их. В случае алгебраических уравнений здесь помогают следующие признаки. Уравнение нечетной степени заведомо имеет хотя бы один корень. Всякое уравнение n -й степени имеет не более n корней. Что же касается трансцендентных уравнений, то они могут иметь в некоторых случаях бесконечное число корней, например уравнению $\sin x = 0$ удовлетворяют все значения $x = k\pi$, где k — любое целое положительное или отрицательное число или нуль. Другие трансцендентные уравнения могут иметь один корень, например уравнение $10^x = 100$ имеет единственный корень $x = 2$. А бывают и уравнения, не имеющие вовсе действительных корней, например $10^x = -100$. Вообще, нужно перед решением исследовать уравнение. Запишем его в виде

$$F(x) = 0. \quad (19.6)$$

Изменяя значения x , необходимо найти, хотя бы приблизительно, те области значений x , при которых $F(x)$ имеет положительные и отрицательные значения. Если функция $F(x)$ непрерывная (например, уравнение алгебраическое), то между двумя значениями $x = x'$ и $x = x''$, при которых знаки $F(x')$ и $F(x'')$ различны, лежит по крайней мере один корень. Если таких корней между x' и x'' больше одного, то должно быть нечетное число.

Хорошо, если можно построить график зависимости $F(x)$, хотя бы грубый, чтобы видеть, где $F(x)$ принимает нулевые значения, т. е. пересекает ось абсцисс. Это тоже можно сделать с помощью ПМК.

— Но если мы это можем сделать, папа, то мы уже решили уравнение, нашли его корни.

— Конечно, это можно также назвать решением уравнения. Иногда так и говорят — графическое решение уравнения. Только это решение очень грубое, корни определяются с большими погрешностями. Поэтому его считают только первым этапом решения уравнения и называют *отделением корней*. В результате можно определить области, в пределах которых находится по одному корню. А вслед за этим начинается новый этап — *уточнение корней*: их значения определяются с допустимой или возможной точностью. Для ПМК обычно с 5...7 верными цифрами.

Будем стараться программировать так, чтобы одна и та же программа помогала нам и на первом этапе (отделение

корней) и на втором (уточнение корней). Существует много способов уточнения корней. Я хочу не ограничиваться одним способом, а изложить тебе хотя бы основные.

Начнем с дихотомии. Это «страшное» греческое слово обозначает простую вещь — деление пополам. Метод дихотомии в принципе самый простой и безотказный, применимый почти ко всем уравнениям с одним неизвестным.

— Для чего же тогда другие методы, более сложные?

— Да потому, что они более быстрые, чем метод дихотомии. Но начнем с дихотомии. Предположим, мы каким-то образом определили, что уравнение $F(x) = 0$ имеет на интервале (a, b) корень, например, $F(a)$ и $F(b)$ разного знака и функция $F(x)$ на этом интервале непрерывна. Разделим интервал пополам, т. е. найдем число $c = (a + b)/2$, вычислим $F(c)$ и найдем ее знак. Если это знак такой же, как у $F(a)$, то, значит, корень находится на интервале (c, b) , в противном случае — на интервале (a, c) . Таким образом, мы сократили вдвое исследуемый интервал. Далее поступаем так же — сокращаем интервал еще вдвое и т. д., пока мы не придем к интервалу, длина которого равна или меньше допустимой абсолютной погрешности. Каждый такой шаг называется *итерацией*.

— Так это же очень долго!

— Конечно, я ведь тебе сказал, что есть другие методы, более быстрые. Но все же не следует думать, что при дихотомии тратится непомерно много времени. Пусть, например, длина первоначального интервала равна 1, а требуется найти корень с абсолютной погрешностью не более 0.001. Сколько раз нужно делить интервал пополам?

— Тысячу раз.

— Никогда не отвечай, не подумав. Сейчас ты сказал чепуху. Если мы 2 раза разделим единицу пополам, то получим $1/4$, если 3 раза — то $1/8$, а если n раз?

— $1/2^n$ — ответил я, немного подумав.

— Вот это другое дело. Пусть теперь $n = 10$. Чему равно 2^{10} ?

— 1024.

— Так, значит, чтобы уменьшить интервал больше чем в 1000 раз, достаточно проделать 10 итераций, а время, которое придется на это потратить, лишь не намного больше того времени, за которое мы можем 10 раз вычислить функцию $F(x)$. А если мы проделаем не 10, а 20 итераций, то первоначальный интервал уменьшится более чем в миллион раз. Этой точности чаще всего хватает.

Итак, давай составим алгоритм и программу для решения уравнений методом дихотомии.

— А для какого уравнения?

— Для любого. Обычно это делается так. Вычисление $F(x)$ оформляется в виде подпрограммы, а основная программа выполняет операции, связанные с делением интервала пополам и определением, в какой половине находится корень. Попробуй составить такой алгоритм, предполагая, что у тебя есть вспомогательный алгоритм ФУНКЦИЯ, позволяющий вычислять значение $y = F(x)$ при любом вещественном x , а также тебе известны a и b , такие, что $F(a) > 0$ и $F(b) < 0$. Алгоритм должен произвести уточнение корня, лежащего в интервале (a, b) , с погрешностью, меньшей заданного числа ε . Вот что получается:

```
алг ДИХОТОМИЯ (вещ a, b, x, ε)
    арг a, b, ε
    рез x
    нач вещ c
        пока |a - b| ≥ ε
            нц c := (a + b) / 2
                ФУНКЦИЯ (c, y)
                если y < 0
                    то b := c
                иначе a := c
            все
        кц
    кон
```

На основании этого алгоритма я составил такую программу.

Программа 37. Решение уравнения $F(x) = 0$ с точностью до ε методом дихотомии

ИПА ИПВ + 2	÷ ПС ПП 26	$x \geq 0$ 21
ИПС ПА ИПВ — x^2	ИП9 — $x < 0$ 00	ИПА
С/П ИПС ПВ ИПА БП 13	...	В/О

И н с т р у к ц и я. Вместо многоточия (начиная с шага 26) ввести подпрограмму для вычисления $F(x)$, считая, что x находится в РС и РХ, а $F(x)$ должно получиться в РХ. Найти исходный интервал (a, b) , содержащий корень x_i^* , причем $F(a) > 0$, $F(b) < 0$, ($\varepsilon^2 = \text{Р9}$), $a = \text{РА}$, $b = \text{РВ}$ В/О С/П РХ $= x_i^*$.

Эту программу мне помог составить папа. Он подсказал, что вместо условия $|a - b| < \varepsilon$ можно проверять эквивалентное ему условие, не требующее нахождения модуля: $(a - b)^2 < \varepsilon^2$. Однако, когда программа была составлена, папа сказал, что принимает ее только в качестве первого пробного варианта. Но все же предложил мне ее опробовать, хотя бы на примере трансцендентного уравнения

$$0,2x - 15 \lg x = 0. \quad (19.7)$$

Я быстро составил подпрограмму для $F(x)$:

5 ÷ ИПС $\lg 1 5 \times - \text{В/О}$

и ввел ее в готовую программу, начиная с шага 26. Папа похвалил меня за то, что я догадался не множить x на 0,2, а разделить на 5, сэкономив на этом два шага. Затем нужно было найти такие a и b , чтобы $F(a) > 0$ и $F(b) < 0$. Я сразу заметил, что при $x = 1$ $F(x) = 0,2 > 0$, а при $x = 10$ $F(x) = 2 - 15 = -13 < 0$. Поэтому, чтобы не ломать голову, я принял $a = 1$, $b = 10$. В качестве допуска ε я взял 0,001, так что $\varepsilon^2 = 10^{-6}$. Введя все эти данные по инструкции, я запустил ПМК и засек время. Через 1 мин 50 с я прочел ответ: $x_1^* = 1,0318602$. Вычислил невязку $F(x_1^*)$, она оказалась равной $-1,40782 \cdot 10^{-3}$. После этого, сохранив полученные значения a и b , я уменьшил ε в 1000 раз (положив $\varepsilon^2 = 10^{-12}$) и снова запустил программу. Примерно еще через 2 мин я получил уточненный результат: $x_1^* = 1,0321970$ с невязкой $5,2 \cdot 10^{-7}$.

— Конечно, — сказал папа, — корень ты нашел с хорошей точностью. Но твоя программа не лишена недостатков, и ты, вероятно, почувствовал это. Во-первых, она тебе никак не помогла найти первоначальные границы интервала (a, b) . Здесь тебе пришлось использовать только интуицию. Во-вторых, ты до сих пор не уверен, что найденный корень единственный. В-третьих, невязку тебе пришлось вычислять вручную. Правда, она не всегда нужна, но все же полезно ее знать для оценки качества решения. Далее, для оценки погрешности решения ты вычисля-

ешь разность $a - b$, т. е. ширину текущего интервала. Казалось бы, для перехода к следующему приближению тебе нужно просто разделить этот интервал на 2 и найти его границу. А ты вместо этого вычисляешь сумму $a + b$ и делишь ее на 2, т. е. делаешь лишние вычисления. И, наконец, если тебя не удовлетворяет точность до 10^{-3} или 10^{-4} , то лучше вовсе не вводить в программу никакого ϵ , а вести вычисления с предельной точностью, на какую способен ПМК при данном методе вычисления. Эта точность будет достигнута тогда, когда разность между a и b не будет различаться ПМК, т. е. окажется машинным нулем. Другими словами, погрешность при этом будет меньше 10^{-7} . Давай составим вместе программу, свободную от этих недостатков. В ней подпрограмма будет использована не только на этапе уточнения корней, но и на первом этапе (разделения корней), и с ее помощью можно будет пытаться обнаружить другие корни уравнения.

Программа 38. Отделение корней уравнения $F(x) = 0$ и уточнение их методом дихотомии с предельной точностью

```

ПС ПП 35 ИП1 С/П ИПВ ИПА — 2 ÷
ПД ИПА + ПС Вх — x==0 20 ИПС С/П
ИПС ПП 35 ИПД БП 08 П1 x≥0 32 ИПС
ПА В/О ИПС ПВ В/О (подпрограмма вычисления
БП 26 F(x))

```

Инструкция. Начиная с шага 35 ввести подпрограмму вычисления $F(x)$, заканчивающуюся командой БП 26. Для отделения корней вводить разные значения x и проверять знак $F(x)$ по схеме $x = RX$ В/О С/П $RX = F(x)$. Когда найдены два достаточно близких значения x , дающие разные знаки $F(x)$, они оказываются записанными соответственно в РА и РВ. Для уточнения корня x_i^* набрать БП 05 С/П $RX = x_i$, Р1 = $F(x_i)$ (невязка). РД = $= \Delta$ — интервал, в котором содержится корень. Далее можно продолжить поиск других корней или убедиться в их отсутствии.

Пример. Для уравнения $0,2x - 15 \lg x = 0$ вводится, как и в предыдущем случае, подпрограмма

5 ÷ ИПС $\lg 1 5 \times -$

но без В/О. Заканчивается программа командой БП 26. Фактически этим подпрограмма не заканчивается. Она продолжается с шага 26 определением знака $F(x)$ и записью x в РА или РВ. Вначале на этапе разделения корней я выделил интервал $(1,00, 1,05)$, в котором должен быть корень. Затем, набрав БП 05 С/П и подождав около 3 мин, я увидел ответ $x^* = 1,032197$. Невязка, хранящаяся в Р1, равна $-9 \cdot 10^{-8}$. В РД хранится ширина Δ интервала (a, b) , до которого мы дошли в процессе делений пополам. Она равна $2,38 \cdot 10^{-8}$ и может служить оценкой максимальной погрешности найденного корня.

Затем я попробовал поискать другие корни, подавая на вход различные значения x в области допустимых для данного уравнения $(0, \infty)$. При уменьшении x от 1 до 0 $F(x)$ возрастает, оставаясь положительным. Значит, в этом интервале уравнение корней не имеет. А значения $x \leq 0$ не входят в область определения $\lg x$. Когда же я стал увеличивать x , то заметил, что первый член растет быстрее, чем второй, и $F(x)$ вначале убывает, а затем возрастает и при $x = 200$ $F(x) > 0$. Варьируя x , я нашел, что $F(x)$ изменяет знак где-то между 160 и 170. Запустив программу на этапе уточнения корня, приблизительно через 4 мин получил ответ: $x_2^* = 166,63184$ с невязкой Р1 = $= -2 \cdot 10^{-6}$ и $|\Delta| = \text{РД} = 4,77 \cdot 10^{-6}$. Других корней уравнение не имеет, так как при дальнейшем увеличении x первый член растет быстрее, чем абсолютное значение второго. Следовательно, $F(x)$ будет возрастать и не сможет превратиться в нуль.

После этого я попробовал решить алгебраическое уравнение по этой программе. Взял я первое попавшееся уравнение 5-й степени

$$F(x) = 2x^5 - 3x^4 + 11x^3 - 10x^2 + x - 7 = 0. \quad (19.8)$$

Так как это уравнение нечетной степени, то оно имеет по крайней мере один корень. Для вычисления $F(x)$ составил подпрограмму по методу Горнера, начиная с шага 35:

1	0	П3	6	П0				
0	ИПС	×	КИП3	+	L0	41	БП	26

и ввел коэффициенты в регистры памяти: 2 = Р9, -3 = Р8, 11 = Р7, -10 = Р6, 1 = Р5, -7 = Р4. Регистр 3 является управляющим для косвенных вызовов, а Р0 — счетчиком числа коэффициентов.

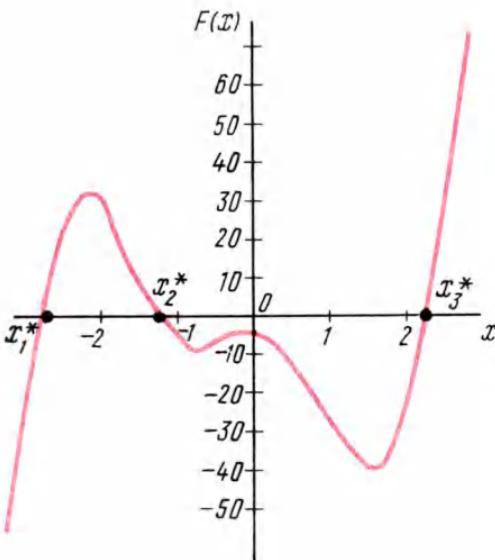


Рис. 19.1

Команда ИПС вызывает значение x , записанное в РС. Сначала оно умножается на нуль, что на первый взгляд может показаться нелепостью. Но это подготавливает операционный стек к приему первого коэффициента, а затем, после возвращения по циклу к команде ИПС, значение x умножается не на нуль, а на находящуюся в РУ промежуточную сумму. Попробовав для разделения корней подавать на вход различные x , я быстро обнаружил, что имеется корень на интервале между 1,2 и 1,3. Запустив ПМК на уточнение корня и подождав около 5 мин, получил значение этого корня $x_1^* = 1,2868814$ с невязкой $P1 = 1,5 \cdot 10^{-6}$ и интервалом $\Delta = РД \approx 4,77 \cdot 10^{-8}$. Других корней это уравнение, по-видимому, не имеет.

Я захотел еще решить какое-нибудь уравнение с несколькими корнями. Папа посоветовал мне изменить знаки некоторых коэффициентов при старших степенях, например

$$F(x) = 2x^5 + 3x^4 - 11x^3 - 10x^2 + x - 7 = 0. \quad (19.9)$$

В поисках корней я подавал различные значения x от -10 до $+10$. Прежде всего я убедился даже без ПМК, что при $x < -10$ определяющим является первый член, так что $F(x) < 0$. Аналогично при $x > 5$ $F(x) > 0$. Таким образом, все корни должны лежать в интервале от -10 до $+5$. Проверив в этих пределах различные x и построив очень грубый график (рис. 19.1), я нашел три области, где находятся корни: между -3 и $-2,5$, между $-1,5$ и -1 и между 2 и $2,5$. Затем на этапе уточнения корней я нашел: $x_1^* = -2,7531841$ при невязке $-6 \cdot 10^{-6}$ и $|\Delta| \approx 4,77 \cdot 10^{-8}$; $x_2^* = -1,2958692$ при невязке $-4 \cdot 10^{-7}$ и $|\Delta| \approx 2,38 \cdot 10^{-8}$; $x_3^* = 2,1805217$ при невязке $2,7 \times 10^{-6}$ и $|\Delta| \approx 4,77 \cdot 10^{-8}$.

Все это заняло много времени — около 40 мин. Но ничего, завтра папа меня научит, как решать уравнения быстрее.

20

ПО СТУПЕНЬКАМ И СПИРАЛЯМ

Сегодня суббота, и я с самого утра с нетерпением ждал, когда папа расскажет об ускоренных методах решения уравнений. Но у него оказались неотложные дела, и он сказал, что в этом лучше разбирается мама. Мама же сказала, что прежде она должна сходить в магазин, приготовить обед и убрать квартиру. Я предложил ей свою помощь. Сходил в магазин и убрал квартиру. К полудню все хозяйственные дела были выполнены. После этого мама сказала, что может на алгоритмическом уровне объяснить мне, что такое простая итерация и метод Ньютона, но программы для ПМК я должен буду составить сам, а если не смогу, то обратиться за помощью к папе.

— Что значит не смогу? — возмутился я. — Я уже умею по любому алгоритму составлять программы.

— Ладно, посмотрим, — сказала мама. — Так вот, метод простой итерации действительно очень прост, если его правильно применять. Пусть нужно найти корни уравнения $F(x) = 0$. Предположим, что ты уже отдалел интервал (a, b) , в котором находится корень. Преобразуем уравнение, приведя его к такому виду:

$$x = \varphi(x), \quad (20.1)$$

где $\varphi(x)$ — некоторая функция. Это можно сделать различными способами, и от выбора способа во многом зависит успех. Забегая вперед, скажу: для применимости метода простой итерации необходимо, чтобы производная правой части (20.1) $\varphi'(x)$ при x , близком к корню, была по модулю меньше единицы.

— А что такое производная? Я что-то краем уха слышал о ней, но толком не помню.

— А что такое график функции, помнишь?

— Конечно.

— Поимаешь, в каждой точке графика функция $\varphi(x)$ с увеличением x может возрастать, а может и убывать.

— Может также оставаться постоянной.

— Так вот, если функция $\varphi(x)$ остается постоянной, то ее производная $\varphi'(x) = 0$ в этой точке. Если $\varphi(x)$ возрастает, то $\varphi'(x) > 0$, а если убывает, то $\varphi'(x) < 0$. А величина производной — это крутизна, с которой функция $\varphi(x)$ возрастает или убывает. Точнее производная определяется так. Рассмотрим значение функции $\varphi(x_0)$ в точке $x = x_0$. Теперь увеличим x на какую-то малюсенькую величину Δ . Тогда и $\varphi(x)$ изменится и станет равной $\varphi(x_0 + \Delta)$ (рис. 20.1). Производной называется предел отношения приращения функции $\varphi(x_0 + \Delta) - \varphi(x_0)$ к приращению аргумента Δ , когда Δ стремится к нулю:

$$\varphi'(x) = \lim_{\Delta \rightarrow 0} \frac{\varphi(x + \Delta) - \varphi(x)}{\Delta}. \quad (20.2)$$

Тебе ведь папа объяснил, что такое предел? Вот еще одно пояснение. Если в точке x_0 провести прямую, касательную к кривой $\varphi(x)$, и продлить ее до пересечения с осью x , то она образует с этой осью некоторый угол, обозначим его α (рис. 20.2). Тогда производная равна тангенсу этого угла:

$$\varphi'(x_0) = \operatorname{tg} \alpha. \quad (20.3)$$

Но мы отвлеклись от метода простой итерации. Пусть ты преобразовал уравнение к виду (20.1) и пусть в процессе отделения корня ты нашел некоторое значение x_0 , которое, хотя и грубо, дает приближенное значение корня. Будем x_0 называть нулевым приближением.

Далее алгоритм такой. Вычисляем $\varphi(x_0)$, которое принимаем за первое приближение к корню $x_1 = \varphi(x_0)$. За-

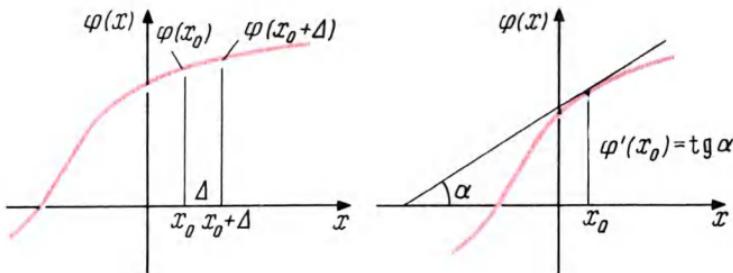


Рис. 20.1

Рис. 20.2

тем так же находим второе приближение $x_2 = \varphi(x_1)$ и продолжаем действовать так и дальше по формуле

$$x_n = \varphi(x_{n-1}). \quad (20.4)$$

Когда значение x на двух последовательных итерациях оказалось одинаковым, т. е. $x_n = x_{n-1}$, то это значит, что x_n удовлетворяет нашему уравнению (20.1), т. е. является корнем.

Почему при такой итеративной процедуре мы приближаемся к корню? Это легко понять из рис. 20.3. На нем в координатах x , y нанесены два графика: $y = x$ и $y = \varphi(x)$. Абсцисса точки, в которой кривая $\varphi(x)$ пересекается с прямой $y = x$, и есть корень уравнения (20.1). Кривая $y = \varphi(x)$ идет менее круто, чем прямая $y = x$, т. е. $\alpha < 45^\circ$, поэтому производная $\varphi'(x)$ в точке пересечения положительна и меньше единицы.

Выберем некоторое начальное значение $x = x_0$, например, правее точки пересечения. Ей соответствует точка Φ_0 , которую принимаем за x_1 . Затем получаем x_2 , x_3 и т. д. Из рисунка видно, как, спускаясь по ступенькам, мы подходим к значению корня. Если выбрано x_0 меньше корня, то мы подходим к значению корня, но не спускаясь, а поднимаясь по ступенькам.

На рис. 20.4 показан случай, когда производная $\varphi'(x)$ отрицательна, модуль ее меньше 1. Здесь тоже, какое бы ни выбрали x_0 , в процессе итераций мы будем приближаться к корню, но не по ступенькам лестницы, а по «ломаной спирали». Другими словами, последовательные прибли-

женные значения x будут подходить к корню не с одной стороны, а колебляясь вокруг корня.

Но так бывает только тогда, когда $|\varphi'(x)| < 1$. В противном случае получаются графики вроде показанных на рис. 20.5 и 20.6. В этом случае наш алгоритм будет не приближать x к корню, а удалять от него, по ступенькам, если $\varphi'(x) > 1$, или по ломаной спирали, если $\varphi'(x) < -1$.

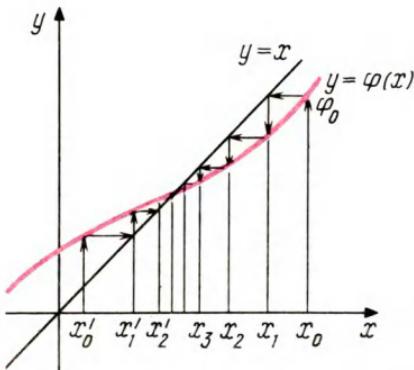


Рис. 20.3

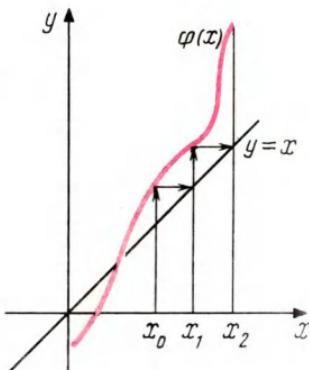


Рис. 20.5

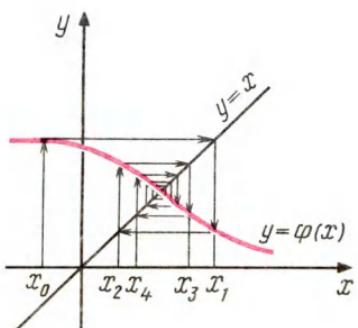


Рис. 20.4

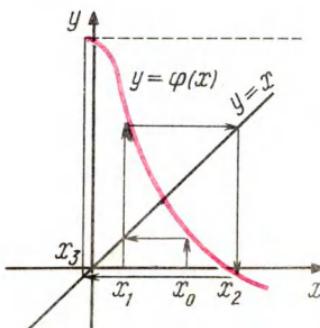


Рис. 20.6

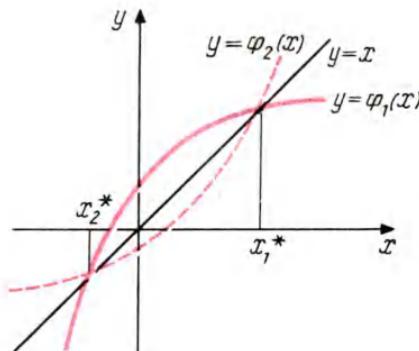


Рис. 20.7

Вот еще один типичный случай, когда уравнение (20.1) имеет два корня x_1^* и x_2^* , т. е. кривые $y = \varphi(x)$ и $y = x$ пересекаются в двух точках (рис. 20.7). В одной точке пересечения $|\varphi'(x)| < 1$, а в другой точке $|\varphi'(x)| > 1$. Следовательно, алгоритм простой итерации приведет нас к одному корню, соответствующему одной точке пересечения. Для того чтобы найти второй корень, придется преобразовать уравнение (20.1) так, чтобы получить другую функцию $\varphi(x)$, которая показана штриховой линией.

— Мне не совсем ясно, как можно преобразовать уравнение так, чтобы вместо большой производной $\varphi'(x)$ получилась маленькая, — спросил я.

— Сейчас я покажу тебе это на простом примере. Возьмем какое-нибудь квадратное уравнение из тех, которые ты решал вчера.

— Вот, пожалуйста, уравнение

$$x^2 - 20\ 000x + 5 = 0. \quad (20.5)$$

У него корни $x_1^* \approx 2,5 \cdot 10^{-4}$ и $x_2^* \approx 20000$.

— Представим его в виде (20.1). Это можно сделать так. Перенесем второй член в правую часть и разделим обе части на 20 000. Получим

$$x = (x^2 + 5)/20\ 000. \quad (20.6)$$

Производная правой части равна $\varphi'(x) = x/10\ 000$. Если $x = x_1^* = 2,5 \cdot 10^{-4}$, то $\varphi'(x) = 2,5 \cdot 10^{-8} < 1$. Но если

$x = x_2^* - 20\ 000$, то $\varphi'(x) = 2 > 1$. Следовательно, методом итерации корень x_1^* найти можно, а x_2^* нельзя.

Чтобы найти второй корень, нужно по-другому преобразовать уравнение (20.5). Перенесем в правую часть второй и третий члены и извлечем квадратный корень из обеих частей. Получим

$$x = \sqrt{20\ 000x - 5} \quad (20.7)$$

Можно показать, что производная правой части равна $\varphi'(x) = 10\ 000/\sqrt{20\ 000x - 5}$. При $x = x_2^* = 2,5 \times 10^{-4}$ знаменатель $\varphi'(x)$ превращается в нуль, т. е. при x , близком к x_1^* , $|\varphi'(x)| \gg 1$. При $x = x_2^* = 20\ 000$ $\varphi'(x) \approx 0,5 < 1$. Таким образом, используя уравнение в виде (20.7), можно методом итераций найти корень x_2^* , но нельзя найти x_1^* .

Вот и все, что я хотела тебе рассказать. А как составлять программу, соображай сам. Когда составишь, я дам тебе одно трансцендентное уравнение, которое мне сейчас часто приходится решать.

И я стал соображать. Алгоритм здесь очень простой, так что его даже незачем записывать. Вычисление $\varphi(x)$ оформим подпрограммой, а основная программа должна сравнивать полученное приближенное значение с предыдущим и останавливать вычисление, когда их разность даст машинный нуль. Вот что у меня получилось.

Программа 39. Решение уравнения $x = \varphi(x)$ методом простой итерации

П1 ПП 11 ИП1 ХУ П1 — x = 0 01 ИП1
С/П (подпрограмма вычисления $\varphi(x)$) В/О

Инструкция. Вписать начиная с шага 11 программу вычисления $\varphi(x)$, где $x = P1$, $x_0 = RX$ В/О С/П $RX = x^*$ (корень уравнения). Команды ИП1 ХУ П1 — x = 0 ... служат для того, чтобы ввести новое приближенное значение корня в P1 и сравнить его с предыдущим значением.

Пример. Для нахождения корней уравнения (20.5) приходится раздельно искать меньший корень по (20.6) и больший по (20.7). Для (20.6) я ввел следующую подпрограмму (начиная с шага 11):

ИП1 x² 5 + 2 ВП 4 ÷ В/О (20.8)

В качестве начального приближения я взял нарочно число 10, далекое от обоих корней. Вычисление продолжалось

17 с, и результат получился $2,5 \cdot 10^{-4}$, как и при решении по программе 36.

Для (20.7) подпрограмма получилась такая:

ИП1 2 ВП 4 × 5 — V⁻ В/О (20.9)

При том же начальном приближении $x_0 = 10$ вычисление продолжалось почти 2 мин и получился результат, отличающийся от результата по программе 36 восьмом знаке, 19 999, 998.

К этому времени папа освободился и пришел посмотреть, чем я занимаюсь.

— Отлично, — сказал он. — Простую итерацию ты освоил. Но программа, которую ты составил, по-моему, не подходит к твоему темпераменту. Да и я предпочитаю более веселые программы.

— При чем тут темперамент?

— Да очень при чем. Вычислитель должен быть очень спокойным, чтобы сидеть сложа руки и ждать 5...10 мин, пока итерации не сойдутся полностью, даже если не нужно получить корень с точностью до восьмого знака. Впрочем, это дело вкуса. Я же предпочитаю строить программу так.

Программа 40. Решение уравнения $x = \varphi(x)$ методом простой итерации.

П1 (вычисление $\varphi(x)$) С/П БП 00

Инструкция. Начиная с шага 01 вписать программу для вычисления $\varphi(x)$: $x_0 = RX$ В/О С/П ... $RX = x_n$ СП ... Наблюдать последовательные приближения x и закончить, когда требуемое число верных знаков не имеет тенденции изменяться.

Я быстро ввел эту программу в ПМК, включив в нее фрагмент (20.8) без первой команды ИП1, которая здесь, очевидно, не нужна, и последней В/О. В качестве начального приближения я взял снова $x_0 = 10$. Запустив программу, я стал считать число итераций, г. е. сколько раз я нажимаю клавишу С/П. Оказалось, что уже после четвертого нажатия повторился предыдущий результат $x^* = 2,5 \cdot 10^{-4}$. заняло это около 20 с. Затем я заменил фрагмент (20.8) на (20.9) и ввел то же значение $x_0 = 10$. На этот раз мне пришлось нажать клавишу С/П 29 раз, и ушло на это около 3 мин. Результат получился $x^* = 19 999, 998$.

— Интересно бы сравнить дихотомию и простую итерацию по скорости решения, — сказал я. — Попробуем решить уравнение

$$0,2x - 15 \lg x = 0, \quad (20.10)$$

которое мы вчера решили по методу дихотомии.

— Попробуй.

Я преобразовал это уравнение к виду (20.1):

$$x = 75 \lg x. \quad (20.11)$$

По-видимому, с помощью такого представления я смогу найти один из двух корней. А чтобы найти другой, нужно разделить обе части (20.11) на 75 и взять от обеих частей антилогарифмы. Это получается очень просто:

$$x = 10^{x/75}. \quad (20.12)$$

Соответственно (20.11) программа 40 получается такая:

$$\text{П1 } \lg 7 5 \times \text{С/П БП 00} \quad (20.13)$$

а соответственно (20.12) —

$$\text{П1 } 7 5 \div 10^x \text{ С/П БП 00} \quad (20.14)$$

Набрав программу (20.13) и введя наугад $x_0 = 50$ (примерно посередине между ожидаемыми корнями), нашел корень $x_1^* = 166,63184$, на что потребовалось 12 итераций, примерно 40 с. По программе (20.14) с тем же начальным значением $x_0 = 50$ корень $x_2^* = 1,0321971$ получен за 30 с при 8 итерациях. Конечно, это значительно быстрее, чем по методу дихотомии, когда затрачивалось 3 ... 4 мин на каждый корень. Я вошел во вкус и попросил папу дать мне еще какое-нибудь хитрое уравнение.

— Могу тебе предложить интересное трансцендентное уравнение. В журнале «Квант» № 11 за 1985 год (стр. 31) есть такая задача: «Приведите пример числа α такого, что $\{\alpha\} + \{1/\alpha\} = 1$, и докажите, что α — иррациональное число». Здесь $\{\alpha\}$ — твоя хорошая знакомая, мадам Фраксьон, т. е. дробная часть числа α . Конечно, ПМК вряд ли поможет тебе доказать иррациональность α . Но зато с его помощью, используя простую итерацию, ты должен легко найти хоть несколько десятков значений α . Однако для этого придется немного поломать голову.

Я охотно согласился, тем более что голова у меня довольно крепкая. Должен признаться, что на мои размыш-

ления ушло больше часа. Кстати, дорогие читатели, по пробуйте сами решить эту задачу, а уж потом прочтите, как это сделал я.

А сделал я это, в конце концов, так. Для применения простой итерации мне нужно уравнение

$$\{x\} + \{1/x\} = 1 \quad (20.15)$$

представить в виде

$$x = \varphi(x).$$

Но x в чистом виде в исходном уравнении отсутствует. Попробую его ввести так: $\{x\} = x - [x]$, где $[x]$ — целая часть (мсъе Антье) от x . Теперь можно (20.15) переписать в таком виде:

$$x - [x] + 1/x - [1/x] = 1. \quad (20.16)$$

Будем пока интересоваться только положительными корнями. Тогда из двух величин $[x]$ и $[1/x]$ одна обязательно равна нулю. Действительно, если $x > 1$, то $[x] > 0$, в этом случае $(1/x) < 1$ и, следовательно, $[1/x] = 0$. Наоборот, если $x < 1$, то $[x] = 0$. Между прочим, уже из самого исходного уравнения (20.15) видно, что если α является его корнем, то и $1/\alpha$ тоже является корнем. Поэтому можно ограничиться отысканием тех корней $\alpha_1, \alpha_2, \alpha_3, \dots$, которые больше единицы, а все корни, меньшие единицы, получим как $1/\alpha_1, 1/\alpha_2, \dots$ При $x > 1$ $[1/x] = 0$, и мы получим уравнение в виде

$$x - [x] + 1/x = 1 \text{ или } x = 1 + [x] - 1/x. \quad (20.17).$$

Вот в этом виде уравнение вполне пригодно для применения простой итерации: $x_n = 1 + [x_{n-1}] - 1/x_{n-1}$. Составляю программу

П1 П9 КИП9 1 ИП9 + ИП1 1/x — С/П
БП 00

Инструкция. $x_0 = РX$ В/О С/П РХ = x_1
С/П, ..., РХ = x_n С/П, ... Вычисления прекращаются,
когда $x_n = x_{n-1}$ (точнее, когда их разность окажется
машиным нулем).

Учитывая, что отыскиваемые этой программой корни должны быть больше единицы, я выбирал в качестве x_0 числа 2, 3, 4 и т. д. и при каждом из них через несколько итераций получил свой корень. Такими корнями оказались: 2,6180340; 3,7320508; 4,7912879; 5,8284271; 6,8541020;

7,8729834 и т.д. Беря из этих корней обратные величины, получаем значения корней, меньшие единицы: 0,381966; 0,26794919; 0,20871215; 0,17157287; 0,14589803; 0,12701665

... Невязки для всех этих корней не превышают $1 \cdot 10^{-7}$, а в большинстве случаев дают машинный нуль. Очевидно, что так можно вычислить сколько угодно положительных корней. Правда, с увеличением корня точность вычисления на ПМК несколько уменьшается.

Тут опять подошла мама и, убедившись, что я довольно быстро нахожу корни трансцендентного уравнения, заинтересовалась этим и сказала:

— Знаешь ли, твоя программа может мне пригодиться. Мне нужно вычислять собственные функции и собственные числа экспоненциального ядра. Эти страшные слова пусть тебя не пугают. Дело в том, что нужно найти много корней трансцендентного уравнения

$$x = a \operatorname{tg}(2\pi x) = \varphi(x) \quad (20.18)$$

при различных значениях постоянной a . Здесь, конечно, аргумент тангенса в радианах. График правой части при фиксированном a имеет известный тебе вид тангенсоиды (рис. 20.8). Ты, конечно, видишь, что имеется бесконечное число корней, один из которых равен нулю при любом a . Другие корни расположены симметрично относительно нуля, так что если существует корень x^* , то существует и корень $(-x^*)$. Поэтому мы можем ограничиться вычислением только положительных корней.

Я рассмотрел график и увидел, что только для корня $x^* = 0$ кривая $\varphi(x)$ идет менее круто, чем прямая $y = x$, и то только при $a < 1/2\pi$. Следовательно, уравнение (20.18) в исходной форме позволяет найти только один этот корень, что интереса не представляет. Поэтому преобразуем уравнение, разделив обе его части на a , взяв arctg от них и разделив еще на 2π :

$$x = (1/2\pi) \operatorname{arctg}(x/a). \quad (20.19)$$

В этом случае программа 40 будет иметь следующий вид:

П1 ИПА ÷ $\operatorname{arctg} \pi \div 2 \div \text{С/П БП 00}$

Причем значение a я записываю в РА, а начальное приближение x_0 — в РХ. Положив $a = 0,1$, $x_0 = 1$, я запустил программу и быстро нашел корень $x_1^* = 0,16196843$. Чтобы найти другие корни при том же a ,

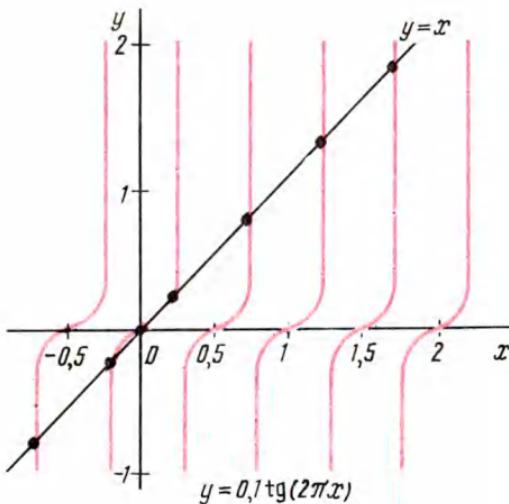


Рис. 20.8

я стал вводить другие положительные значения x_0 , но какие бы x_0 я ни вводил (даже 1000 и больше), я быстро приходил к тому же корню x_1^* . Только вводя отрицательные x_0 , я приходил к другому симметрично расположенному корню $x_2^* \approx -x_1^* = -0,16196845$. Я пошел к маме и сообщил ей:

— Оказывается, при $a = 0,1$ твое уравнение имеет только три корня: $-0,16196845,0$ и $+0,16196843$.

— Как ты можешь такое говорить? — возмутилась мама. — Ты же ясно видел на рис. 20.8, что должно быть бесконечное число корней. Вероятно, твоя программа врет. Хотя, постой. Кажется, я знаю, в чем дело. Ты совершил неправомерный переход от уравнения (20.18) к (20.19) и потерял при этом все корни, кроме трех. Это произошло, когда ты брал арктангенс.

— Почему же?

— Да потому, что арктангенс — функция многозначная, а ты учитываешь только одно значение, на интервале от $-\pi/2$ до $+\pi/2$. Вернее, это делает твой ПМК. Можешь убедиться в этом. Возьмем, например, $\operatorname{tg} 50^\circ$. Как видишь, он равен $-0,27189969$. А теперь возьмем

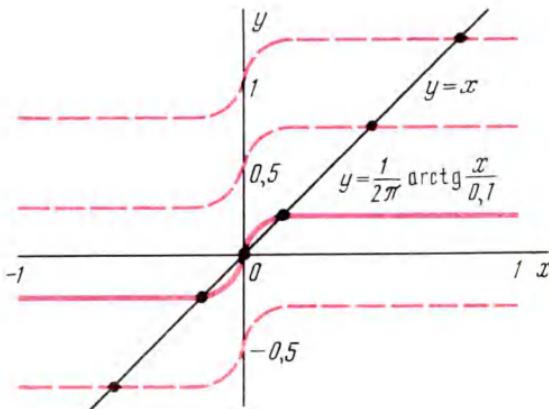


Рис. 20.9

arctg от этого числа. Ты, может быть, ожидаешь получить 50? Вот и нет. Ты получил $-0,2654816$, т. е. на 50,265482 меньше. Разделим это число на π . Получилось ровно 16. А все потому, что $\operatorname{tg} \alpha = \operatorname{tg}(\alpha + k\pi)$, где k — любое целое положительное или отрицательное число.

Что отсюда следует? Предположим, что мы понимаем операцию arctg так, как ее осуществляет ПМК. Тогда из (20.18) вовсе не обязательно следует, что $\operatorname{arctg}(x/a) = -2\pi x$. Равенство (20.18) возможно также, если $\operatorname{arctg}(x/a) = -2\pi x + k\pi$. И то значение x , для которого это выполнено, тоже является корнем уравнения (20.18). Понятно? Или не очень? Чтобы было яснее, перейдем опять к графику. Уравнение (20.18) привело нас к рис. 20.8, из которого мы и увидели, что оно имеет бесконечное число корней. Заметь, что функция $\operatorname{tg} 2\pi x$ однозначная — каждому значению x соответствует единственное значение $\operatorname{tg} 2\pi x$.

Теперь, перейдя к уравнению (20.19), с учетом нашего определения функции arctg , ты получишь то, что изображено непрерывной линией на рис. 20.9. Здесь ясно, что при некоторых значениях a существует три корня, т. е. три точки пересечения кривой с прямой $y = x$. При увеличении a крутизна кривой $y = (1/2\pi) \operatorname{arctg}(x/a)$ уменьшается и, как легко убедиться, при $a > 1/2\pi$ уравнение (20.19) имеет единственный корень $x^* = 0$.

Чтобы рис. 20.9 был эквивалентен рис. 20.8, необходимо добавить к нему еще штриховые кривые. Это отражает то, что в действительности функция arctg является много-значной. Но ПМК знает только одну кривую — непрерывную. Поэтому для получения уравнения, эквивалентного (20.18), нужно вместо $\operatorname{arctg} \alpha$ писать $\operatorname{arctg} \alpha + k\pi$. Таким образом, вместо (20.19) нужно написать

$$x = (1/2\pi) \operatorname{arctg} (x/a) + k/2, \quad (20.20)$$

где k — любое целое число или нуль. Фактически (20.20) это не одно уравнение, а множество, каждое из которых соответствует некоторому значению k . И корни каждого из них являются также корнями исходного уравнения (20.18). Можешь ли ты изменить свою программу в соответствии с (20.20)?

— Нет ничего проще. Величину k , начиная с $k=0$, введу в Р4, найдя очередной корень, буду командой КИП4 увеличивать его содержимое на единицу.

Теперь программа с добавлением управления параметром k приняла следующий вид.

Программа 41. Нахождение положительных корней уравнения $x = a \operatorname{tg}(2\pi x)$ методом простой итерации

ПА Сх П4 С/П ИПА \div $\operatorname{arctg} \pi \div$ ИП4
 $+ 2 \div$ С/П БП 04 КИП4 ХУ БП 04

Инструкция. ($a = РХ$) В/О С/П $x_0 = РХ$
 С/П ... РХ $= x_n$ С/П, ..., РХ $= x_i^*$ $\xrightarrow{\text{ШГ}}$ $\xrightarrow{\text{ШГ}}$
 С/П ... РХ $= x_{i+1}^*$ и т. д.

Вначале методом простой итерации отыскивается корень при $k=0$, а когда корень найден, вместо клавиши

С/П 2 раза нажимаю клавишу ШГ, в результате пропускаются команды БП 04, а затем нажимается клавиша С/П, запуская программу для отыскания корня при $k=1$ и т. д.

Для $a=0,1$ я получил такие значения корней:
 0,16196843; 0,72828235; 1,2371634; 1,7408677;
 2,2429088; 2,7442029; 3,2450971; 3,7457521; 4,2462525;
 4,7466475; 5,2469671; 5,7472310; 6,2474525; 6,7476415;
 7,247804; ...

— Вот это именно то, что мне нужно, — сказала ма- ма.

— За это я тебе завтра расскажу про метод Ньютона.

Ну, что ж, завтра, так завтра.

21

ПО КАСАТЕЛЬНОЙ

— Так что же это за метод Ньютона? — спросил я родителей за утренним чаем.

— А для чего он тебе нужен? — ответила мама вопросом на мой вопрос. — Ты ведь уже научился решать уравнения численными методами дихотомии и простой итерации.

— Все-таки дихотомия занимает много времени, а простая итерация не всегда получается. Я пробовал вчера применить ее к алгебраическому уравнению 5-й степени, но не смог его нужным образом представить.

— Конечно, — вмешался папа, — для алгебраических уравнений, да и для многих трансцендентных метод Ньютона удобнее. К тому же он достаточно прост. Чтобы понять его, нужно только знать, что такое производная. Ньютон впервые додумался до производной (одновременно с Лейбницем и независимо от него) и широко использовал ее в созданной им науке — теоретической механике. И тут же он заметил, что производная помогает численному решению уравнений.

А суть этого метода вот в чем.

Тут папа взял листок бумаги и нарисовал график (рис. 21.1).

— Вот график функции $y = F(x)$. Те точки, где этот график пересекает горизонтальную ось, определяют корни уравнения $F(x) = 0$. Предположим, что мы построили грубый график функции $F(x)$, и теперь знаем, где нужно искать корни нашего уравнения. Пусть x_0 — некоторая грубая оценка корня. Ему соответствует значение функции $y_0 = F(x_0)$. Проведем касательную к кривой $y = F(x)$ в точке (x_0, y_0) и найдем точку пересечения x_1 этой касательной с горизонтальной осью. Как правило, точка x_1 ближе к корню x^* , чем точка x_0 . Проведем теперь касательную в точке (x_1, y_1) , где $y_1 = F(x_1)$. Она пересечется с горизонтальной осью в точке x_2 , которая, как правило, еще ближе к корню x^* . Продолжая таким образом проводить касательные, мы быстро найдем корень x^* с допустимой погрешностью.

Теперь смотри. Производная $F'(x)$ в точке $x = x_0$ есть не что иное, как тангенс угла α , который равен отношению $y_0 / (x_0 - x_1)$. Отсюда следует

$$x_0 - x_1 = F(x_0)/F'(x_0) \text{ или } x_1 = x_0 - F(x_0)/F'(x_0).$$

Аналогичные зависимости справедливы для последующих итераций, так что в общем случае

$$x_k = x_{k-1} - F(x_{k-1})/F'(x_{k-1}). \quad (21.1)$$

Это и есть итерационная формула Ньютона. Продолжая этот процесс, мы, как правило, сможем найти значение корня с любой заданной точностью.

— Папа, ты все время говоришь «как правило». Значит, бывают исключения?

— Исключения бывают, если ты выберешь точку x_0 недостаточно близко к корню. Доказаны следующие свойства метода Ньютона. Если приближения x_0, x_1, \dots находятся на интервале (a, b) , содержащем корень, и на этом интервале первая и вторая производные $F'(x)$ и $F''(x)$ не меняют знаков, то по формуле (21.1) мы всегда будем приближаться к корню x^* . Вблизи корня это условие обычно выполняется. Если же мы удалились от корня настолько, что, например, знак производной $F'(x)$ изменился, то уже первая касательная не приблизит нас к корню, а удалит от него. Видишь на рисунке (рис. 21.2)?

Программу для решения уравнений методом Ньютона я советую тебе построить так же, как для метода дихотомии, т. е. чтобы вычисление функции $F(x)$ использовалось как при отделении корней, так и при уточнении.

— А как быть с производной $F'(x)$?

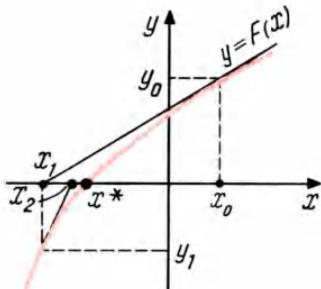


Рис. 21.1

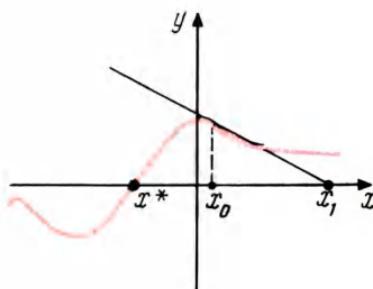


Рис. 21.2

— Подумай сам. Здесь возможны разные пути. Если ты умеешь написать выражение для производной, то ее можно вычислять по отдельной подпрограмме. Я думаю, для алгебраических уравнений, когда $F(x)$ и $F'(x)$ — многочлены, лучше всего так и сделать. А для трансцендентных уравнений это не всегда удобно.

— А если вычислять производную в соответствии с ее определением, хотя бы приблизительно:

$$F'(x) = \lim_{\Delta \rightarrow 0} \frac{F(x + \Delta) - F(x)}{\Delta} \quad (21.2)$$

и вместо предельного перехода взять в качестве Δ какую-то очень маленькую величину?

— Можно и так, — сказал пapa. — Но нужно внимательно отнестись к выбору Δ . Если взять ее слишком большой, то производная будет вычислена с большой погрешностью из-за отсутствия предельного перехода. А при очень малой Δ в числителе будет вычитание близких величин, а это, как ты знаешь, тоже ведет к погрешности.

— А нужно ли бояться этих погрешностей? — вставила мама. — Ведь они не повлияют на окончательное значение вычисленного корня, если только не нарушится сходимость алгоритма. Вообще же эта погрешность может только замедлить сходимость.

— Конечно, это так, — подтвердил пapa. — В общем, я тебе советую составить две основные программы: одну — для алгебраических уравнений высоких степеней, но, конечно, таких, чтобы их коэффициенты уместились в памяти ПМК, вторую — для произвольных уравнений $F(x) = 0$, если $F(x)$ можно вычислить на ПМК. Надеюсь, ты знаешь, как по многочлену $F(x)$ вычислить его производную?

— Конечно, — сказал я гордо (хотя только вчера разыскал эту формулу в книжке). — Если многочлен

$$F(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0,$$

то его производная

$$\begin{aligned} F'(x) = & n a_n x^{n-1} + (n-1) a_{n-1} x^{n-2} + \dots + \\ & + 2 a_2 x + a_1. \end{aligned} \quad (21.3)$$

— Ну, вот и отлично. Жду твои программы.

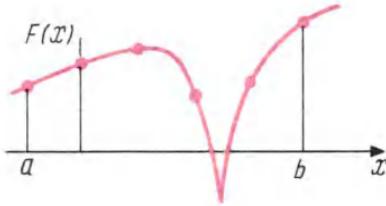


Рис. 21.3

— Только не забудь составить алгоритмы, — добавила мама. — Без этого тебе программы не сделать.

— Ну, это как сказать! — подумал я, но ничего не сказал.

Тем не менее я стал обдумывать алгоритм. И тут обратил внимание на одно обстоятельство. Ведь процесс отделения корней, который мы производим, опираясь на интуицию и случайные пробы, в сущности не описывается каким-либо алгоритмом. Алгоритм должен за определенное конечное число шагов дать определенное решение задачи. А здесь возможны такие хитрые функции $F(x)$, в которых корни где-то глубоко запрятаны. Например, если функция изменяется так, как на этом рисунке (рис. 21.3), и я буду прощупывать ее значения, обозначенные точками, то, конечно, я приду к выводу, что на интервале (a, b) она знака не меняет и корней не имеет. В действительности она имеет два близких корня. Этими сомнениями я поделился с папой.

Папе очень понравилось, что я стал размышлять о таких вещах.

— Действительно, такого алгоритма, который позволяет для любой функции $F(x)$ отделить корни или хотя бы определить количество вещественных корней, не существует. Тем не менее мы обычно справляемся с этой задачей. Это лишний раз доказывает, что и без алгоритма можно решать некоторые задачи. Впрочем, для алгебраических уравнений, когда $F(x)$ — степенной многочлен, алгоритм, позволяющий отделить все вещественные корни, существует. (Его открыл французский математик Ж. Ш. Ф. Штурм лет 150 назад). Но он, хотя и позволяет отделить все вещественные корни за конечное число шагов, очень сложен. Поэтому на практике им пользуются очень

редко, чаще предпочитают, как и мы, отделять корни методом проб. Что же касается твоих программ, то предположим, что корни уже отделены. Составь алгоритмы лишь для уточнения корней по методу Ньютона и по ним программы для ПМК. Только предусмотри в этих программах вычисление $F(x)$ для любых x , которыми можно пользоваться при отделении корней путем отыскания интервалов смены знаков $F(x)$. Надеюсь, ты ненаткнешься на такие неудобные функции, как на твоем рисунке (рис. 21.3).

— Это, конечно, облегчает задачу, — сказал я и стал составлять алгоритмы.

Первый алгоритм я составил для уточнения корня алгебраического уравнения n -й степени, полагая, что грубо приближенное его значение x_0 найдено в процессе отделения корней. Он получился таким:

```
алг НЬЮТОН 1 (таб вещества  $a[0:n]$ , вещества  $x_0, x^*$ )
арг  $a, x_0$ 
рез  $x^*$ 
нач вещества  $F, F', x, x_c, \varepsilon$ 
 $x := x_0; x_c := x_0 + 2\varepsilon$ 
пока  $|x - x_c| \geq \varepsilon$ 
нц
 $F := a[n]x^n + a[n-1]x^{n-1} + \dots +$ 
 $+ a[1]x + a[0]$ 
 $F' := na[n]x^{n-1} + (n-1)a[n-1]x^{n-2} +$ 
 $+ \dots + 2a[2]x + a[1]$ 
 $x_c := x; x := x - F/F'$ 
кц
 $x^* := x$ 
кон
```

В этом алгоритме я обозначил x_c — приближенное значение корня, полученное в предыдущей итерации (старое). Алгоритм заканчивается, когда новое значение x отличается от старого меньше чем на ε . Для того чтобы войти в цикл, я в самом начале присвоил x_c значение $x_0 +$

— 2e. Впрочем, в программе для ПМК можно будет сделать по-другому.

Для неалгебраического уравнения $F(x) = 0$ я составил другой алгоритм, полагая, что имеется вспомогательный алгоритм, которому я дал имя ФУНКЦИЯ, вычисляющий по x функцию $F(x)$.

```
алг НЬЮТОН 2 (вещ  $x_0$ ,  $x^*$ )
арг  $x_0$ 
рез  $x^*$ 
нач вещ  $x$ ,  $x_c$ ,  $\varepsilon$ ,  $\Delta$ 
 $x := x_0$ ;  $x_c := x_0 + 2\varepsilon$ 
пока  $|x - x_c| \geq \varepsilon$ 
    нц ФУНКЦИЯ ( $x$ ,  $F(x)$ )
        ФУНКЦИЯ ( $x + \Delta$ ;  $F(x + \Delta)$ )
         $x_c := x$ ;  $x := x - F(x)\Delta / [F(x + \Delta) - F(x)]$ 
    кц
 $x^* := x$ 
кон
```

Покончив с этими алгоритмами, я приступил к составлению программы для алгебраических уравнений по алгоритму НЬЮТОН 1. Здесь возникли свои проблемы. Хочется, чтобы программа позволяла решать уравнения высоких степеней. Однако на хранение коэффициентов потребуется много регистров памяти. Кроме того, нужны будут регистры для хранения текущих значений x , $F(x)$, $F'(x)$, x_c и ε — целых пять. Еще понадобится регистр для управления косвенными выводами коэффициентов, иначе не хватило бы программной памяти.

Поразмыслив, я решил прежде всего отказаться от хранения ε . Буду выводить очередные значения x на индикатор, останавливая ПМК, и принимать по ним решение о продолжении или прекращении уточнения, как в программе 40. Папа правильно сказал, что такой способ больше соответствует моему темпераменту. Конечно, всякий желающий может видоизменить мои программы, введя значение ε и осуществляя условный переход по

правилу $(x - x_c)^2 - r^2 < 0$. Но тогда придется допустимую степень n уменьшить на единицу. Итак, прикину, на какую степень уравнения я могу рассчитывать.

Помимо коэффициентов уравнения мы должны хранить в памяти значение x , $F(x)$, пока вычисляется производная $F'(x)$. Нужен еще регистр для управления выводом коэффициентов. Я, конечно, вспомнил о своей программе вычисления многочлена 12-й степени (программа 25). Там значение x хранится в операционном стеке, 13 регистров памяти использованы для хранения коэффициентов многочлена, а один Р0 — для управления остальными регистрами.

Если мне удастся и в этой программе сохранить x в регистрах операционного стека и один регистр использовать для хранения $F(x)$, то смогу решать уравнения 11-й степени. Можно было бы, конечно, сохранить в операционном стеке и значение $F(x)$. Тогда бы удалось решить уравнения 12-й степени. Может быть, кому-нибудь это и удастся, но у меня не получилось. Поэтому я решил ограничиться уравнениями 11-й степени. Тогда $F(x)$ я буду записывать в РС, a_{11} — в РВ, a_{10} — в РА, a_9 — в Р9,..., a_1 — в Р1. Для a_0 я сохранил регистр РД, так как Р0 нужен для управления косвенными обращениями к остальным регистрам. Я, конечно, использовал возможность обращаться к Р0 «с черного хода» командой КИП ↑, как и в программе 25.

Для отделения корней я оформил вычисление $F(x)$ в виде подпрограммы. Я решил сначала вычислить $F'(x)$ и запомнить его в РС, а затем вычислить $F(x)$. Кроме того, я применил автоматизацию ввода коэффициентов (кроме a_0) в соответствующие регистры. В результате у меня получилось вот что.

Программа 42. Решение алгебраических уравнений
 $F(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 = 0$ степени $n \leq 11$ методом Ньютона

ПИ	29	С/П	↑	1	1	П0	XY	↑	0
КИП↑	ИП0	×	+	×	L0	10	XY	÷	ПС
→	ПП	29	ИПС	÷	—	С/П	БП	03	1
1	П0	XY	↑	↑	0	КИП↑	+	×	L0
36	ИПД	+	В/О	1	2	П0	С/П	КП0	БП

И н с т р у к ц и я. Для ввода коэффициентов БП 44 С/П $a_{11} = \text{РХ С/П}$, $a_{10} = \text{РХ С/П}$, $a_9 = \text{РХ С/П}$, ..., $a_2 = \text{РХ С/П}$, $a_1 = \text{РХ СП}$, $a_0 = \text{РД}$.

Если степень уравнения меньше 11, заменить недостающие коэффициенты нулями.

Для отделения корней испытуемое значение x ввести в РХ В/О С/П РХ = $F(x)$. Отыскивать области, в которых происходит смена знака $F(x)$.

Для уточнения корней ввести предполагаемое приближенное значение корня $x_0 = \text{РХ БП 03 С/П РХ} = x_1 \text{ С/П}$, $\text{РХ} = x_2 \text{ С/П}$, ... (продолжать до тех пор, пока необходимое количество верных знаков не будет повторяться в последовательных приближениях), $\text{РХ} = x^*$. Для нахождения невязки после определения корня x^* В/О С/П РХ = $F(x^*)$.

Примеры. 1. $F(x) = 2x^{11} - 10x^{10} + 9x^9 - 7x^7 + 6x^6 - 5x^5 + 4x^4 - 8x^3 + 2x^2 + 80x - 8$. Для отделения корней подавались значения $x = -10, -5, -1, 0, +1, +5, +10$. Установлено, что перемена знака $F(x)$ происходит только на интервале $(0, +1)$. По-видимому, уравнение имеет единственный корень. Подав в режиме уточнения корня $x = 0,5$, после трех итераций получим $x^* = 0,099845894$ с невязкой $F(x^*) = 1 \cdot 10^{-7}$.

Для того чтобы сравнить метод Ньютона с методом дихотомии, я ввел в новую программу то же уравнение 5-й степени, которое я решил по программе 38,

$$F(x) := 2x^5 + 3x^4 - 11x^3 - 10x^2 + x - 7 = 0.$$

При вводе коэффициентов я не забыл ввести нули для a_6, \dots, a_{11} . Отделяя корни, нахожу интервалы, в которых они находятся: $(-3; -2,5), (-1,5; -1,2), (2; 2,3)$.

Подавая на вход при уточнении корней соответственно $-2,7, -1,3$ и $2,2$, получаем следующие результаты:

$$\begin{aligned} x_1^* &= -2,7531842 \text{ (3 итерации)}, \\ x_2^* &= -1,2958691 \text{ (3 итерации)}, \\ x_3^* &= 2,1805217 \text{ (3 итерации)}. \end{aligned}$$

Каждая итерация длится около 40 с. Все решение уравнения заняло около 12 мин., что примерно в 3,5 раза меньше, чем по методу дихотомии.

Другая программа по алгоритму НЬЮТОН 2 для решения произвольных уравнений с одной неизвестной оказалась еще проще.

Программа 43. Решение уравнения $F(x) = 0$ методом Ньютона

ПС ПП 23 С/П ПВ ИПД ИПС + ПП 23
ИПВ — ИПВ XY ÷ ИПД × ИПС XY —
С/П БП 00 (подпрограмма вычисления $F(x)$) В/О

Инструкция. С 23-го шага записать подпрограмму для вычисления $F(x)$. При составлении подпрограммы считать, что $x = RX$, и $F(x)$ также вывести в RX . Можно использовать до 75 шагов и регистры памяти от Р0 до Р9 и РА. Величину Δ выбирать в пределах $10^{-4} \dots 10^{-2}$ от ожидаемого значения корня.

Для отделения корней (построения грубой зависимости $y = F(x)$) $x = RX$ В/О С/П $RX = F(x)$.

Для уточнения корня: ($\Delta = R\Delta$) $x_0 = RX$ В/О С/П $RX = F(x_0)$ С/П, $RX = x_1$ С/П, $RX = F(x_1)$ С/П, $RX = x_2 \dots$, получая по очереди приближения x_k и невязки $F(x_k)$ до получения нулевой невязки или до повторения значений x_k . При этом $x^* = x_k = PC$.

Пример. Перепишем уравнение (20.8) $0,2x - 15 \lg x = 0$ в более удобной форме $75 \lg x - x = 0$. Тогда можно составить такую подпрограмму:

П1 $\lg 7 5 \times$ ИП1 — В/О

При отделении корней получили отрицательные значения $F(x)$ для $x = 0,5$, $x = 1$, $x = 200$ и положительные — для $x = 2, x = 10$, $x = 100$. Корни следует искать на интервалах $(1, 2)$ и $(100, 200)$.

При $x_0 = 1$ и $\Delta = 0,001$ после трех итераций получаем $x^* = 1,0321971$ с невязкой $F(x^*) = 5 \cdot 10^{-7}$, при $x_0 = 150$ после четырех итераций $x^* = 166,63184$ с невязкой $F(x^*) = 0$ (машинный нуль). На каждую итерацию тратится около 20 с. В целом решение получается быстрее, чем по методу дихотомии, но медленнее, чем при простой итерации. Впрочем, при простой итерации для каждого корня приходилось вводить свою подпрограмму, тогда как при методе Ньютона одна подпрограмма позволяет найти все корни данного уравнения.

После обеда я показал свои программы папе.

— Программы неплохие. Во всяком случае решать уравнения с их помощью можно. К программе 43 у меня вообще нет претензий. А вот в программе 42 ты кое-что

недоработал. Во-первых, мне не нравится, что при степени уравнения $n < 11$ нужно «недостающие коэффициенты» заменять нулями. Конечно, я не возражаю против вводов нулей, это много времени не занимает. Хуже, что при работе программы много времени тратится на умножения и сложения нулей. Если у тебя уравнение 5-й или 6-й степени, то около половины времени уходит впустую. Я однажды составил аналогичную программу, в которой этого недостатка нет. Правда, она пригодна только для уравнений до 9-й степени. У этой программы есть еще одно достоинство, ускоряющее вычисления, облегчающее отделение корней и позволяющее более надежно решить вопрос о наличии ненайденных действительных корней. Это достигается последовательным понижением степени уравнения.

Пусть, например, $F_n(x)$ — многочлен n -й степени и мы нашли один корень x_1^* уравнения $F_n(x) = 0$. Я не знаю, проходили ли вы в школе теорему Безу?

— Я что-то такой не припомню.

— Теорема очень простая. Она найдена еще в XVIII в. французским математиком Безу и заключается в следующем. Если многочлен $F_n(x)$ разделить на двучлен $x - a$, то остаток от деления равен $F_n(a)$. Отсюда следует, что если a — корень уравнения $F_n(x) = 0$, то остаток равен нулю. Другими словами, $F_n(x)$ делится без остатка на $x - x_1^*$, где x_1^* — любой из корней нашего уравнения. Результатом деления является многочлен $F_{n-1}(x)$ степени $n - 1$. Уравнение $F_{n-1}(x) = 0$ имеет корни, такие же, как и корни исходного уравнения $F_n(x)$, за исключением корня x_1^* . Таким образом, мы можем постепенно исключать находимые корни и снижать степень уравнения. Благодаря этому каждый следующий корень находится быстрее, чем предыдущий. Ясно?

— Да. Только как ты делишь многочлен на двучлен?

— Очень просто. Коэффициенты многочлена

$$F_n(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

записаны в регистры памяти:

$$\begin{aligned} a_0 &= P6, & a_1 &= P7, & a_2 &= P8, & a_3 &= P9, & a_4 &= PA, \\ a_5 &= PB, & a_6 &= PC, & a_7 &= PD, & a_8 &= PO, & a_9 &= P1. \end{aligned} \quad (21.4)$$

После деления на $x - x_1^*$ получаем многочлен $F_{n-1}(x) = b_0 + b_1x + \dots + b_{n-2}x^{n-2} + b_{n-1}x^{n-1}$. Программа

составлена так, чтобы коэффициенты b_i заменили коэффициенты a_i в тех же регистрах памяти. Для этого исходим из равенства

$$(b_0 + b_1 x + b_2 x^2 + \dots + b_{n-1} x^{n-1})(x - x^*) = \\ = a_0 + a_1 x + a_2 x^2 + \dots + a_{n-1} x^{n-1} + a_n x^n. \quad (21.5)$$

Раскрывая скобки, легко получить следующие соотношения:

$$\begin{aligned} b_0 &= -a_0/x^*, \\ b_1 &= (b_0 - a_1)/x^*, \\ b_2 &= (b_1 - a_2)/x^*, \\ &\dots \\ b_k &= (b_{k-1} - a_k)/x^*, \\ &\dots \\ b_{n-1} &= (b_{n-2} - a_{n-1})/x^*, \\ b_n &= (b_{n-1} - a_n)/x^* = 0. \end{aligned} \quad (21.6)$$

Последнее равенство следует из того, что согласно (21.5) $b_{n-1} = a_n$. По этим формулам нетрудно составить программу.

Программа 44. Решение алгебраических уравнений $F_n(x) = 0$ при $n \leq 9$ методом Ньютона с понижением степени

ПП	50	\times	КИП3	$+$	L2	02	П4	C/П	XY
ПП	50	L2	14	\times	КИП3	ИП2	\times	+	L2
14	ИП4	XY	\div	—	С/П	БП	00	П3	ИП5
6	—	П2	5	П4	П5	0	КИП5	—	ИП3
\div	КП4	L2	37	0	КП4	ИП5	6	—	C/П
ИП5	1	+	П3	6	—	П2	XY	\uparrow	\uparrow
↑	0	B/O	5	P5	C/П	КП5	БП	65	

И н с т р у к ц и я.

1. Для ввода коэффициентов уравнения: БП 63 С/П RX = 5, $a_0 = RX$ СП, $a_1 = RX$ C/П, $a_2 = RX$ C/П, ... $a_n = RX$ СП.

2. Для вычисления $F(x)$ в процессе отделения корней: $x = RX$ В/О С/П $RX = F(x)$, $RY = x$.

3. Для уточнения приближенного значения корня x_0 : $x_0 = RX$ (В/О) С/П $RX = F(x_0)$ С/П, $RX = x_1$ С/П, $RX = F(x_1)$ С/П, $RX = x_2$ С/П, ..., $RX = F(x_k)$ С/П, $RX = x_{k+1}$, ... Уточнение заканчивается, когда невязка $F(x_k) = 0$ или разность $x_k - x_{k+1}$ меньше требуемой точности определения корня (в частности, равна машинному нулю). Здесь x_1, x_2, \dots, x_k — последовательные приближения корня x^* .

4. Для понижения степени многочлена $F(x)$ путем деления на $x - x^*$ после получения уточненного корня

ШГ ЦГ С/П $RX = n - 1$ (степень полученного в результате деления многочлена). Коэффициенты полученного многочлена размещены в тех же регистрах, что и коэффициенты исходного многочлена: $a_0 = P6$, $a_1 = P7$, $a_2 = P8$, $a_3 = P9$, $a_4 = PA$, $a_5 = PB$, $a_6 = PC$, $a_7 = PD$, $a_8 = P0$.

После этого можно заняться поиском и уточнением следующего корня, что сделать проще, так как степень полинома снизилась.

Тут я попросил папу помочь мне разобраться в этой программе, так как без подсказки это сделать трудно.

— Что же тут трудного? Все очень ясно. Давай рассмотрим каждую часть инструкций и соответствующие ей шаги программы. Прежде всего шаги с 63 по 68 служат для ввода коэффициентов. Сначала в Р5 записывается число 5. Затем, используя косвенную адресацию, командой КПБ записываются коэффициенты в Р6, Р7, Р8 и т. д. до Р1. Коэффициенты вводятся начиная с a_0 , и после ввода каждого коэффициента нажимается клавиша С/П. При этом не забудь, что коэффициенты вводятся со своими знаками. Здесь использовано то, что Р5 — регистр с нарастающим содержимым, а также свойства «переноса» регистров, по которому числу 14 соответствует Р0, а числу 15 — Р1 (см. выше, стр. 110). После ввода коэффициентов в Р5 записано число $n + 6$, где n — степень введенного многочлена, которая может быть любой до 9 включительно.

Обрати внимание на подпрограмму, занимающую шаги с 50 по 62. Она устанавливает числа в управляющих регистрах Р2 и Р3, причем в Р3 число $n + 7$, а в Р2 число $n + 1$, и, кроме того, вводит число x в операционные регистры Y, Z и T. Это позволяет хранить там текущее значение x , пока оно не будет сменено. Эта подпрограмма

очень важна, так как в ходе вычислений содержимое Р2 и Р3 изменяется и их нужно каждый раз восстанавливать по эталону $n + 6$, хранящемуся в Р5, а также приходится изменять значения x .

Дальше тебе, вероятно, ясно, что шаги со 2 по 8 позволяют вычислить $F(x)$ и записать его значение в Р4. Вводя различные значения x и нажимая каждый раз клавиши В/О С/П, можно вычислять $F(x)$ для разных x . Здесь вычисление производится по схеме Горнера, а коэффициенты вызываются косвенно через Р3. Счетчиком служит Р2.

Если после вычисления $F(x)$ нажать клавишу С/П, не нажимая В/О, то включится часть программы (шаги с 10 по 25), в которой вычисляется $F'(x)$, и производится уточнение корня по формуле Ньютона (21.1). Если продолжать каждый раз нажимать клавишу С/П, то будут осуществляться последующие итерации, причем на индикаторе поочередно будут появляться невязка $F(x)$ и уточненное значение x^* . Для того чтобы при этом не думать, нужно ли нажимать клавишу В/О (для вычисления $F(x)$) или только С/П (для уточнения корня), я ввел в программу шаги 26 и 27, возвращающие к началу программы.

Если уточненный корень x^* находится на индикаторе, т. е. программа остановилась на шаге 25, то можно перейти к делению многочлена на $x - x^*$. Для этого нажимаем

дважды клавишу ШГ, чтобы обойти шаги 26 и 27, и продолжаем считать с шага 28. Здесь корень x^* записывается в Р3, число n — в Р2, а в Р4 и Р5 число 5. Далее, косвенная команда КИП5 вызывает по очереди коэффициенты многочлена n -й степени, которые обрабатываются по формулам (21.6) получения коэффициентов нового многочлена ($n - 1$)-й степени. Они засыпаются в нужные регистры косвенной командой КП4. В последний регистр вместо коэффициента n -й степени засыпается нуль. В результате в Р5 будет записано число $(n - 1) + 6$, соответствующее новой степени многочлена $n - 1$. Число $(n - 1)$ выводится на индикатор.

Обрати внимание, что ввод коэффициентов начинается с a_0 и продолжается вверх до a_n . Точно так же программа деления на двучлен начинается с определения коэффициентов низших степеней многочлена-частного. В то же время вычисления $F(x)$ и $F'(x)$ начинаются с верхней степени n и спускаются до a_0 . Заметь еще, что если один из корней $x^* = 0$, то разделить многочлен на $x - x^*$ по формулам (21.6) нельзя. При попытке разделить на $x^* = 0$

(шаг 72) ПМК выдаст ЕГГОГ. Но это не страшно. Случай $x^* = 0$ легко обнаруживается по нулевому свободному члену ($a_0 = 0$) в $F(x)$, и деление на x вручную до ввода в ПМК не составляет никакого труда, так как сводится к уменьшению на единицу степеней x во всех членах.

Теперь нужно привести пару примеров решения уравнений. Это я поручаю тебе.

Я ввел программу в ПМК и начал с ней экспериментировать. Взял уравнение (19.9) 5-й степени, с которым многое провозился позавчера, осваивая метод дихотомии, $F(x) = 2x^5 + 3x^4 - 11x^3 - 10x^2 + x - 7 = 0$.

Я мог бы облегчить себе задачу: корни этого уравнения мне известны, я их вычислил позавчера. Но я действовал по-честному, забыл о значениях корней и провел все решение по программе 44.

Сначала я ввел коэффициенты. Это заняло 25 с. Затем занялся отделением корней, подавая разные значения x и записывая знаки результатов. Нашел интервал $(2, 3)$, в котором происходит смена знаков. На это у меня ушла 1 мин. В качестве начального приближения x_0 я взял середину этого интервала $x_0 = 2,5$ и стал уточнять корень. После четырех итераций, каждая из которых длится около 40 с, нашел $x_1^* = 2,1805217$ при невязке $\overrightarrow{F(x_1^*)} = 2,5 \cdot 10^{-6}$. Затем, нажав дважды клавишу $\overset{\rightarrow}{\text{ШГ}}$, занялся понижением степени многочлена. Эта операция заняла 25 с.

Для нового уравнения 4-й степени я отдал еще один корень в интервале $(-1, -1,5)$, принял за $x_0 = -1,3$ и, проделав четыре итерации, нашел корень $x_2^* = -1,2958691$ и вычислил его невязку $F(x_2^*) = -4 \cdot 10^{-7}$.

После этого произвел еще одно понижение степени многочлена. В полученном кубическом уравнении я очень легко выделил интервал $(-2,5; -3)$, содержащий корень. Приняв за x_0 примерно середину этого интервала и проведя три итерации, я уточнил корень $x_3^* = -2,7531837$ с невязкой $1 \cdot 10^{-6}$. Как и следовало ожидать, продолжительность итерации с понижением степени уменьшается и при $n = 3$ составляет около 20 с.

Снизив степень делением $F_3(x)$ на $x - x_2^*$, я получил квадратное уравнение с коэффициентами $a_2 = P8 = 2$, $a_1 = P7 = -0,737063$ и $a_0 = P6 = 0,89979$

$$2x^2 - 0,737063x + 0,89979 = 0. \quad (21.7)$$

Дискриминант этого уравнения $a_1^2 - 4a_0a_2 = -6,655 < 0$, следовательно, это уравнение действительных корней не имеет. Таким образом, я нашел все три корня исходного уравнения (19.9). В целом все это заняло около 10 мин, т. е. в 4 раза меньше, чем при методе дихотомии. Результаты отличаются от полученных по методу дихотомии только в последнем знаке. Несколько более точные результаты получены по методу Ньютона. Это следует из сравнения невязок.

Чтобы проверить предельные возможности программы, я решил использовать уравнение 9-й степени. Попросил папу дать мне какое-либо уравнение 9-й степени, у которого все 9 корней действительные. Папа немного поколдовал и записал такое уравнение:

$$F_9 = x^9 - 4x^8 - 10x^7 + 40x^6 + 35x^5 - 140x^4 - 50x^3 + 200x^2 + 24x - 97 = 0. \quad (21.8)$$

Я ввел коэффициенты этого уравнения согласно инструкции, не забывая вводить, где это нужно, знак минус. Затем проверил знак $F(x)$ для нескольких значений x и быстро обнаружил, что $F(1,2) > 0$, а $F(1,5) < 0$. Поэтому грубой оценкой одного из корней может служить $x_0 = 1,3$. Проведя уточнение по методу Ньютона, я после пяти итераций нашел $x_1^* = 1,3494895$. Разделив F_9 на $x - x_1^*$, получил новый многочлен F_8 . Для него я тоже достаточно быстро отделил корень в окрестности 1, который после уточнения оказался равным $x_2^* = 1,0311230$. Продолжая таким образом, я нашел остальные корни: $x_3^* = -3,9999269$, $x_4^* = 1,7928245$, $x_5^* = -1,7562287$, $x_6^* = -1,3821416$, $x_7^* = -1,0175859$ и $x_8^* = -1,9750738$. При этом степень уравнения понижалась каждый раз на единицу и, наконец, дошло до 1-й степени. Коэффициенты этого линейного уравнения хранятся в Р6 и Р7 и легко извлекаются. В результате я получил уравнение

$$F_1(x) = 1,0000184x + 1,9975707 = 0, \quad (21.9)$$

откуда уже элементарно нахожу 9-й корень $x_9^* = -1,9975767/1,0000184 = -1,992534$.

Тут подошел папа и посмотрел на мои результаты.

— Конечно, — сказал он, — в процессе вычисления твоей ПМК допускал погрешности, да и уточнения корней, вероятно, не всегда доводил до максимальной точно-

сти. Поэтому вряд ли можно ожидать, что в полученных корнях больше пяти-шести верных знаков.

— А откуда это видно? — спросил я.

— Очень просто. Помнишь, когда мы составляли формулы для понижения степени (21.6), я обратил твоё внимание, что значение коэффициента при старшей степени не изменяется. Только его степень уменьшается на единицу. А у тебя в исходном уравнении (21.8) старший коэффициент равен 1, а в последнем уравнении (21.9) он немного вырос и стал равным 1,0000184. Правда, это не такая уж большая разница, а значит, и погрешности в вычисленных корнях невелики. Но все же в уравнении 5-й степени (19.9), которое ты решил перед этим, старший коэффициент, равный 2, не изменился и в окончательном квадратном уравнении (21.7). Поэтому можно полагать, что его корни вычислены точнее, чем корни уравнения (21.8).

— Что же теперь делать?

— Вероятнее всего, ничего особенного делать не надо, нужно только представить результаты, сохранив в них не более четырех знаков. Для большей части практических задач такая точность более чем достаточна. Большую точность можно получить, уточнив корни по тому же методу Ньютона и той же программе, но с помощью исходного уравнения (21.8), без понижения степени. Это теперь, сделать очень легко: ты уже знаешь значения корней с очень малой погрешностью и тебе хватит одной-двух итераций для их уточнения.

— Давай попробуем.

Введя опять коэффициенты уравнения (21.8), я стал подавать на вход поочередно все вычисленные значения корней и уточнять их. Это заняло около 15 мин, и получились такие результаты:

Корни, вычисленные с понижением степени	Корни, уточненные по исходному уравнению	Корни, вычисленные с понижением степени	Корни, уточненные по исходному уравнению
—1,992534	—1,9926435	1,3494895	1,3494895
—1,7562287	—1,7561189	1,7928245	1,7928231
—1,3821416	—1,3821602	1,9750738	1,9749743
—1,0175859	—1,0175855	3,9999269	4,0000302
1,0311230	1,0311225		

— Молодец, — сказал папа. — Теперь ты владеешь многими разнообразными методами решения уравнений, как алгебраических, так и трансцендентных. Надеюсь, тебе это пригодится.

— Но это все уравнения с одним неизвестным. А как быть, когда их несколько?

— Ты уж хочешь все сразу. Подожди немного, завтра поговорим и о системах уравнений с несколькими неизвестными. А сейчас пора спать.

— Есть подождать до завтра.

22

СО МНОГИМИ НЕИЗВЕСТНЫМИ

— Так как же решать уравнения со многими неизвестными? — спросил я папу, как только он вернулся с работы.

— Вопрос не такой простой. Если речь идет о системе из двух уравнений с двумя неизвестными или даже из трех уравнений с тремя неизвестными, то особых трудностей нет, и ПМК с этими задачами справляется. Это относится к системам как линейных, так и нелинейных уравнений, в том числе трансцендентных. А вот если неизвестных больше, то привлечь для их решения существующие ПМК довольно трудно: памяти ПМК не хватает для записи всех коэффициентов.

— А как же быть, если нужно решать очень большую систему? Вот я помню, мама как-то хвалилась, что ей пришлось составлять программу для системы с сотней неизвестных. Я даже не поверил тогда, что такие бывают.

— В современной прикладной математике всяческое бывает. Но не все можно решать на микрокалькуляторе. Не будем отбивать хлеб у больших ЭВМ. Попробуем решить не очень большие системы уравнений, как линейных, так и нелинейных, с помощью ПМК.

Рассмотрим сначала системы линейных уравнений, например систему с двумя неизвестными

$$\left. \begin{array}{l} a_1 x + b_1 y = c_1, \\ a_2 x + b_2 y = c_2. \end{array} \right\} \quad (22.1)$$

Для нее известны очень простые формулы, дающие однозначное решение, если, конечно, оно существует:

$$\begin{aligned}x &= (c_1b_2 - b_1c_2)/(a_1b_2 - b_1a_2), \\y &= (a_1c_2 - c_1a_2)/(a_1b_2 - b_1a_2).\end{aligned}\quad (22.2)$$

Я не сомневаюсь, что по таким простым формулам ты очень легко и быстро составишь удобную программу.

— Конечно. Кое-чему я уже научился. Я буду коэффициенты вводить автоматически, применив счетчик на регистре Р0, в который введу число 6 (количество коэффициентов), используя «черный ход», т. е. команду КП↑, вводящую число из регистра Х в регистр, номер которого находится в Р0, не изменяя содержимого этого регистра. Затем вычислю общий знаменатель двух формул (22.2) и запомню его, наконец, вычислю числитель и сами результаты x и y . Вот что получится.

Программа 45. Решение системы линейных уравнений с двумя неизвестными

6	П0	С.П	КП↑	L0	02	ИП6	ИП2	×	ИП5
ИП3	×	—	ПД	ИП4	ИП2	×	ИП1	ИП5	×
—	ИПД	÷	С П	ИП6	ИП1	×	ИП4	ИП3	×
—	ИПД	÷	С П						

Инструкция. В/О С/П, a_1 С/П, b_1 С/П, c_1 С/П, a_2 С/П, b_2 С/П, c_2 С/П РХ = x С/П, РХ = y .

Пример. $2x + 3y = 4$, Ответ: $x = 1$, $y = 0,66666666$.
 $5x - 6y = 1$.

Продолжительность ввода коэффициентов 20 с, вычисления корней 15 с.

— Отлично, — сказал папа. — Ты очень быстро составил программу и придраться не к чему. Впрочем, попробуй решить такую простую систему

$$\begin{cases} 2x + 3y = 4, \\ 6x + 9y = 12. \end{cases}$$

— Пожалуйста, — сказал я, ввел коэффициенты и вдруг ... получил ответ ЕГГОГ. Что бы это значило?

— Это значит, что твои уравнения линейно зависимы и поэтому имеют не единственное решение. Ты можешь

убедиться, что им удовлетворяют различные пары значений (x, y) , например $(2, 0)$ или $(0, 4/3)$, или $(1/2, 1)$, и множество других. Возможен также случай, когда уравнения несовместны, т. е. не имеют вовсе решений, например

$$\left. \begin{array}{l} 2x + 3y = 4, \\ 6x + 9y = 1. \end{array} \right\}$$

Для этой системы решений не существует. В этом случае ПМК тоже укажет на ошибку.

Ты легко справился с системой линейных уравнений с двумя неизвестными, уложив всю программу в 34 шага. Сколько шагов тебе потребуется для решения системы с тремя неизвестными, например

$$\left. \begin{array}{l} a_1 x + b_1 y + c_1 z = d_1, \\ a_2 x + b_2 y + c_2 z = d_2, \\ a_3 x + b_3 y + c_3 z = d_3. \end{array} \right\} \quad (22.3)$$

— Конечно, побольше. Шагов 60 ... 70. А может быть, и 80.

— Попробуй. Известны формулы, дающие решение, если, конечно, оно существует:

$$\begin{aligned} x &= (d_1 b_2 c_3 + b_1 c_2 d_3 + c_1 d_2 b_3 - c_1 b_2 d_3 - c_2 b_3 d_1 - \\ &\quad - c_3 b_1 d_2) / D, \\ y &= (a_1 d_2 c_3 + d_1 c_2 a_3 + c_1 a_2 d_3 - c_1 d_2 a_3 - c_2 d_3 a_1 - \\ &\quad - c_3 d_1 a_2) / D, \\ z &= (a_1 b_2 d_3 + d_1 b_2 a_3 + d_1 a_2 b_3 - d_1 b_2 a_3 - d_2 b_3 a_1 - \\ &\quad - d_3 a_1 b_2) / D, \end{aligned} \quad (22.4)$$

где $D = a_1 b_2 c_3 + b_1 c_2 a_3 + c_1 a_2 b_3 - c_1 b_2 a_3 - c_2 b_3 a_1 - c_3 b_1 a_2$.

Попытайся составить программу.

Я стал наращивать свою программу 45. На ввод коэффициентов у меня ушло на один шаг больше, т. е. 7 вместо 6, так как в Р0 пришлось ввести не 6, а 13. Ну, это совсем не страшно. Затем я стал вычислять знаменатель D . В предыдущей программе это занимало 8 шагов. Здесь, конечно, придется потратить больше. Оказалось, значительно больше — 36 шагов. Но столько же мне придется потратить на каждый из трех числителей. Итого 144 шага!

А программная память моего ПМК всего 98 шагов. Так что ничего не получится. Об этом я и доложил папе.

— Что-то рано ты сдаешься, — заметил он. — То ты собирался уложитьсь в 60 ... 70 шагов, а сейчас уже и сотни мало? Нужно немного подумать, кое-где схитрить, кое-чем пожертвовать и, может быть, что-нибудь получится.

Я стал размышлять. Чем можно пожертвовать, ясно. Нужно отказаться от автоматического ввода коэффициентов, а вводить их вручную. На этом я сэкономлю 7 шагов. Но это ведь капля в море. Нужно что-то придумать. Косвенное обращение при вычислении числителей и знаменателя в (22.4) не давало никакой экономии. Уж больно сложно входят коэффициенты в эти формулы. Но тут я заметил одну интересную закономерность. Если в выражении для знаменателя заменить a_1 , a_2 , и a_3 соответственно на d_1 , d_2 и d_3 , то получится числитель в формуле для x . Если в этом числителе заменить b_1 , b_2 и b_3 на a_1 , a_2 и a_3 , то получится (правда, с обратным знаком) числитель для y . А если в последнем заменить c_1 , c_2 и c_3 на b_1 , b_2 и b_3 , то получим числитель для z . Ура! Можно вычисление знаменателя и всех трех числителей оформить подпрограммой, а в промежутках между обращениями к подпрограмме заниматься перестановкой коэффициентов. Это, конечно, даст большую экономию.

Я быстро прикинул такую программу. На подпрограмму, как я уже раньше считал, уходит 36 шагов. На перестановку коэффициентов 15 шагов. Таких перестановок три, на них тратится 45 шагов. Вместе с подпрограммой 81 шаг, 9 шагов уходит на то, чтобы разделить числители на знаменатель и вывести результаты. Да четыре раза нужно обращаться к подпрограмме, еще по два шага (ПП и адрес), всего 8. Итого ровно 98 шагов. Значит, можно, уложиться в программную память.

Но я рано обрадовался. Когда я стал программу записывать, то обнаружил пробелы в своей бухгалтерии. Я не учел шаги, необходимые для записи знаменателя в память и для смены знака у числителя второй формулы. В общем, оказалось, что двух шагов мне все-таки не хватает.

Я немного приуныл и стал подумывать о том, чтобы заставить пользователя вручную делить числители на знаменатель, хотя это и неудобно и некрасиво. Но тут вспомнил, как папа добывал несколько лишних шагов, используя косвенное обращение к подпрограмме, на которое затрачивается один шаг, а не два. Таких обращений здесь

четыре, значит, четыре шага я сэкономлю, и будет все в порядке. Теперь я могу записать программу. Чтобы проще оперировать регистрами, я решил ввести a_1 , a_2 и a_3 в Р1, Р2, Р3, b_1 , b_2 и b_3 в Р4, Р5, и Р6, c_1 , c_2 и c_3 в Р7, Р8 и Р9. Теперь, если смотреть на пульт с правого боку, то коэффициенты расположатся так, как они записаны в (22.3). Для d_1 , d_2 и d_3 я использую РА, РВ и РС. Знаменатель записываю в Р0, а РД использую для косвенного обращения к подпрограмме. Таким образом, получилась

Программа 46. Решение системы линейных уравнений (22.3) с тремя неизвестными

```

КППД П0 ИПА ИП1 ПА ХУ П1 ИПВ ИП2 ПВ
ХУ П2 ИПС ИП3 ПС ХУ П3 КППД ИП0 ÷
С/П ИПА ИП4 ПА ХУ П4 ИПВ ИП5 ПВ ХУ
П5 ИПС ИП6 ПС ХУ П6 КППД /—/ ИП0 ÷
С/П ИПА ИП7 ПА ХУ П7 ИПВ ИП8 ПВ ХУ
П8 ИПС ИП9 ПС ХУ П9 КППД ИП0 ÷ С/П
ИП1 ИП5 ИП9 × × ИП2 ИП6 ИП7 × ×
+ ИП3 ИП4 ИП8 × × + ИП7 ИП5 ИП3
× × — ИП8 ИП6 ИП1 × × — ИП9
ИП4 ИП2 × × -- В/О

```

Инструкция. $a_1 = P1$, $a_2 = P2$, $a_3 = P3$, $b_1 = P4$, $b_2 = P5$, $b_3 = P6$, $d_1 = PA$, $d_2 = PB$, $d_3 = PC$, $c_1 = P7$, $c_2 = P8$, $c_3 = P9$ ($60 = RD$) В/О С/П
 $PX = x$ С/П, $PY = y$ С/П, $PZ = z$.

Пример.

$$\left. \begin{array}{l} 2x + 10y + 6z = 32, \\ 3x + 9y + 6z = 33, \\ 5x + 15y + 20z = 65. \end{array} \right| \quad (22.5)$$

Ответ: $x = 3$, $y = 2$, $z = 1$. Время вычислений 100 с.

— Ну, вот видишь, — сказал папа, проверив программу. — Если подумать, можно уложить формулы (22.4) в ПМК. Однако можно эту же систему уравнений решить и по-другому, применив итерационный метод. Более того, этим итерационным методом можно решать и более обшир-

ные системы линейных уравнений, а также некоторые нелинейные, в том числе и трансцендентные, системы уравнений с несколькими неизвестными.

Этот итерационный метод по существу такой же, как и тот, что ты использовал для уравнений с одним неизвестным. Он очень прост и быстро работает, но предъявляет высокие требования к форме уравнений. Поясню его на примере линейной системы с тремя неизвестными, но не той системы (22.5), которую ты только что так блестяще решил, а другой:

$$\left. \begin{array}{l} 17x + y - 5z = 10, \\ 4x + 20y + 5z = 12, \\ -7x + 3y + 18z = -11. \end{array} \right\} \quad (22.6)$$

Прежде чем решать ее итеративным методом, воспользуемся тем, что у тебя набрана замечательная программа 46, позволяющая решить эту систему точно с одного раза.

Я ввел эти коэффициенты в регистры ПМК и меньше чем через 2 мин получил результат:

$$x = 0,37995198, \quad y = 0,66766706, \quad z = -0,57462985.$$

— Хорошо, — сказал пapa. — Теперь посмотрим, как получить это же решение итерационным методом. Представим, как мы это делали в случае одного неизвестного, наши уравнения в виде

$$\left. \begin{array}{l} x = \Phi_1(x, y, z), \\ y = \Phi_2(x, y, z), \\ z = \Phi_3(x, y, z). \end{array} \right\} \quad (22.7)$$

Найдем (я потом покажу тебе как) исходное (нулевое) приближение корней x_0 , y_0 , и z_0 и вычислим

$$\begin{aligned} x_1 &= \Phi_1(x_0, y_0, z_0), \quad y_1 = \Phi_2(x_1, y_0, z_0), \quad z_1 = \\ &= \Phi_3(x_1, y_1, z_0). \end{aligned}$$

Затем продолжаем таким же образом: $x_2 = \Phi_1(x_1, y_1, z_1)$ и т. д., пока значения x , y и z не перестанут изменяться (или будут изменяться очень мало, если нам достаточно получить не очень точные приближения).

Конечно, как и в случае с одной неизвестной, не всякие уравнения вида (22.7) обеспечивают сходимость ите-

рационного алгоритма. Обычно сходимость обеспечивается, если в одном уравнении коэффициент при x значитель- но больше коэффициентов при остальных неизвестных, в другом — коэффициент при y больше других коэффициен- тов и т. д. Здесь всюду имеется в виду абсолютное значе- ние коэффициентов. Исходя из этого, я и выбрал для при- мера систему (22.6). Теперь смотри, как она преобразуется к виду (22.7):

$$\left. \begin{array}{l} x = (10 - y + 5z)/17, \\ y = (12 - 4x - 5z)/20, \\ z = (-11 + 7x - 3y)/18. \end{array} \right\} \quad (22.8)$$

Чтобы получить начальные приближения, будем иг- норировать неизвестные с малыми коэффициентами, т. е. положим $x_0 = 10/17$, $y_0 = 3/5$ и $z_0 = -11/18$. Эти зна- чения я ввожу в РА, РВ и РС и составляю программу по формулам (22.8):

$$\begin{array}{rcl} 1 & 0 & \text{ИПВ} - \text{ИПС} \ 5 & \times & + & 1 & 7 \\ \hline \div & \text{ПА С/П} & 1 & 2 & \text{ИПА} & 4 & \times & - & \text{ИПС} \\ 5 & \times & - & 2 & 0 & \div & \text{ПВ С/П ИПА} & 7 \\ \times & 1 & 1 & - & \text{ИПВ} & 3 & \times & - & 1 & 8 \\ \hline \div & \text{ПС С/П БП} & 00 \end{array}$$

Инструкция. $x_0 = \text{РА}$, $y_0 = \text{РВ}$, $z_0 = \text{РС}$
 В/О С/П РХ = x_1 С/П, РХ = y_1 С/П, РХ = z_1 С/П, РХ =
 $= x_2$ С/П, ... Вычисления продолжаются до получения
 значений x , y , z , практически не изменяющихся на сле-
 дующих итерациях.

Эта программа получилась значительно короче, чем программа 46, только 45 шагов. При вычислении корней пришлось проделать 10 итераций (т. е. 10 раз получать последовательные приближения значений x , y и z). На каждую итерацию затрачивалось около 25 с. В результате мы получили

$$x = 0,37995198, \quad y = 0,66766705, \quad z = -0,57462983,$$

что с точностью до двух единиц последнего разряда сов- падает с результатом, полученным по программе 46.

— Все это очень хорошо, папа, но годится не для всех систем уравнений. Ты ведь специально выбрал такие урав-

нения, чтобы в каждом из них один коэффициент был значительно больше остальных.

— Я это сделал только для того, чтобы не отвлекаться на мелочи. Ведь любую систему можно привести к такому виду, если немного «покрутить». Рассмотрим теперь систему нелинейных уравнений

$$\left. \begin{array}{l} x = \ln(y/z), \\ y = 3 + z^2 - x^2, \\ z = 1 + xy/10. \end{array} \right\} \quad (22.9)$$

Пока мы не знаем ответа, нельзя сказать, будет ли последовательность итераций сходиться. Что же делать? Нужно попробовать. Очень может быть, что итерации будут сходиться. В противном случае будем преобразовывать уравнения и опять пробовать.

Я легко составил программу, полагая $x = PA$, $y = PB$, $z = PC$:

$$\begin{array}{rcl} \text{ИПВ ИПС} & \div & \text{ln ПА С/П ИПС } x^2 \quad \text{ИПА } x^2 \\ - & 3 & + \text{ПВ С/П 1} \quad \text{ИПА ИПВ } \times \quad 1 \\ 0 & \div & + \text{ПС С/П БП } 00 \end{array}$$

Труднее было выбрать исходное приближение. Я рассуждал так. В третьем уравнении член $xy/10$, вероятно, мал по сравнению с единицей. Поэтому выберем $z_0 = 1$. Далее из второго уравнения с учетом выбранной оценки z можно положить $y_0 = 4$. А для x вообще начальное приближение не нужно: его найдем по программе из первого уравнения как $\ln(y_0/z_0)$. Итак, начинаю.

Мне повезло, итерации сходились, хотя и очень медленно. После 20 итераций первые три цифры в значениях неизвестных стабилизировались. Но чтобы получить хотя бы по пять верных цифр, пришлось проделать 50 итераций. Окончательно я получил $x \approx 1,026251$, $y \approx 3,910493$, $z \approx 1,401314$. Подставив эти значения в уравнения, я получил относительные погрешности порядка 10^{-5} .

Конечно, это вычисление заняло много времени. Но, с другой стороны, как бы можно было быстрее решить такую систему уравнений? Я спросил об этом у папы.

— Есть и другие методы численного решения систем трансцендентных уравнений. Но все они еще более сложные. Так что ограничимся этим.

И я тоже считаю мои записи об уравнениях законченными. Тем более, что существует еще немало интересных задач для ПМК.

23

НЕМНОГО СТАТИСТИКИ

Сегодня у нас неожиданная гостья — тетя Марина. Раньше она приходила очень часто, во всяком случае на праздники и почти каждое воскресенье. Но сейчас она пишет диссертацию. Я очень удивился, открыв дверь и увидев ее на пороге: сегодня день не праздничный и не воскресенье.

— Папа дома? — спросила она.

— Скоро должен вернуться, — сказал я. — Заходи, присаживайся, рассказывай, как твои дела?

— Да какие у меня могут быть дела? Единственное сейчас дело — сижу над диссертацией, будь она неладна. Да и сейчас мне нужна срочная консультация твоего папы.

— А чем же папа тебе может помочь? Ты ведь врач, а папа медицинской никогда не занимался и, наоборот, всячески избегает врачей.

— Ну, конечно, чем реже встречаешься с врачами, тем здоровее будешь, — сказал появившийся в дверях папа. — Но к тебе, Мариночка, это, разумеется, не относится. Тебя я всегда рядом видеть. Сейчас давай пообедаем, и рассказывай, как твои успехи.

— Да нет у меня никаких успехов и даже об обеде думать не хочется. У меня застопорилась диссертация. Я набрала огромный статистический материал, обследовала более сотни больных, составила таблицу, из которой все видно. А с меня требуют, чтобы я эти данные статистически обработала. А как это сделать, я понятия не имею. Вот и пришла к тебе.

— Ладно, — сказал папа. — Сейчас подберу тебе пару книжек по математической статистике, самых популярных, по которым ты дней за десять сумеешь в этом разобраться, а если что-то будет непонятно, то объясню.

— Как за десять дней? — воскликнула тетя Марина.
— Я должна завтра представить своему руководителю основные результаты. Мне нужно для начала хотя бы вычислить средние значения и стандартные отклонения содержания кальция у больных с заболеванием окколощитовидных желез (гипопаратиреозом). Вот у меня две группы больных по 20 человек в каждой. Диагноз у них одинаковый. Первая группа получала в течение 1,5 месяцев препарат оксидейвит, эффективность которого я проверяю, а вторая группа — контрольная — традиционные лекарства, в частности витамин Д. Мне нужно убедиться, что оксидейвит помогает лучше.

— Это проще простого. Сейчас Миша на своем микроКалькуляторе тебе все это подсчитает. Я только напомню ему, какими формулами нужно пользоваться. Смотри, Миша, у тебя есть некоторый массив данных, короче говоря, измеренные значения величины x_i , где i принимает значения от 1 до n . Вычислим, прежде всего, среднее выборочное значение a_x , которое выражается формулой

$$a_x = (x_1 + x_2 + x_3 + \dots + x_n)/n. \quad (23.1)$$

— Так ведь это же среднее арифметическое n чисел, — заметил я.

— Ну, конечно. Я только хочу подчеркнуть, что это не просто какие-то числа, а статистическая выборка, т. е. результаты измерения в нашем статистическом эксперименте. Если n — достаточно большое число, то такое среднее выборочное значение может служить хорошей оценкой математического ожидания случайной величины x . Этим я хочу сказать, что если я буду продолжать эксперимент, т. е. измерять значения x_{n+1}, x_{n+2}, \dots и вычислять их средние значения по формуле (23.1), то получу практически тот же результат.

— Считать по этой формуле очень просто. Нужно только вводить по очереди значения x_i , а ПМК вычислит их сумму и число n , а потом осуществит деление. Вот по такой программе:

ИП9 + П9 КИП4 С/П БП 00 ИП9 ИП4 ÷ С/П

Инструкция. $(0 = P4 = P9 B/O) \quad x_1 =$
 $\leftarrow PX\ C/P, \quad x_2 = PX\ C/P, \dots, \quad x_n = PX\ C/P \quad \overrightarrow{SHG}\ C/P$
 $PX = a_x.$

Я использую Р9 для накопления суммы, а Р4 — для подсчета числа n , пользуясь тем, что при косвенном обращении КИП4 содержимое Р4 возрастает на единицу. После окончания списка я нажимаю клавишу ШГ, чтобы обойти команду безусловного перехода, и \rightarrow делю сумму на n .

— Разумеется, такая программа работать будет, хотя можно было бы ее сделать более удобной, подсказывая пользователю, какое значение x_i ему нужно вводить. Для этого можно было бы добавить после шага 03 команду ИП4, и тогда на индикаторе будет высвечиваться каждый раз номер i числа x_i , которое ты ввел. Но не в этом суть. Ты ведь слышал, что тете Марине нужно вычислить не только средние значения для двух выборок (принимавших препарат и контрольной группы), но и *стандартное отклонение* s_x , квадрат которого вычисляется по такой формуле:

$$s_x^2 = [(x_1 - a_x)^2 + (x_2 - a_x)^2 + \dots + (x_n - a_x)^2]/n. \quad (23.2)$$

Величину s_x^2 называют еще средним квадратическим отклонением (СКО).

— А для чего это нужно?

— Это вопрос в данном случае следовало бы адресовать тете Марине. Я думаю, что это нужно здесь для того, чтобы можно было судить о том, насколько различия средних значений в двух выборках следует приписать приему препарата, а не случайному колебаниям. Дело в том, что СКО s_x^2 может служить оценкой дисперсии случайной величины x . Слово дисперсия в переводе на русский язык означает разброс. Ясно, что если этот разброс велик, то небольшое различие в значениях a_x для двух выборок ни о чем не говорит, тогда как большое различие в значениях a_x при малом s_x безусловно свидетельствует о пользе (или, может быть, о вреде) лекарства. Так, что ли?

Тетя подтвердила.

— Конечно, это самый примитивный способ проверки статистической гипотезы (о том, что лекарство полезно), — продолжал папа. — Существуют более надежные статистические критерии. Пока замечу, что для оценки дисперсии s_x^2 величины x лучше пользоваться не значением s_x^2 , а $[n/(n - 1)] s_x^2$. Впрочем, при больших n это не существенно. Для величины a_x , которая ведь тоже случайная, дисперсия s_a^2 в n раз меньше, чем для x , т. е. ее можно оце-

нить значением $s_x^2 / (n - 1)$. И, наконец, статистики для предварительных прикодок пользуются так называемым правилом «трех сигм». Это значит, если значения a_x для двух выборок будут различаться больше чем на $3\sigma_a = 3s_x / \sqrt{n - 1}$, то имеются веские основания считать, что это различие вызвано не случайными причинами (например, неточностями измерений), а действительным различием выборок.

Я внимательно слушал и в то же время думал, как формулу (23.2) воплотить на ПМК. Здесь одно существенное препятствие. В каждый член суммы входит среднее значение a_x . Чтобы его вычислить, нужно поочередно ввести все n значений x_i . Если бы у меня был не ПМК, а большая ЭВМ с мощной оперативной памятью, я бы ввел в нее все x_i и быстро вычислил сначала a_x , а затем и s_x , но у нас такой памяти нет.

— Придется, — сказал я папе, — два раза вводить все значения x_i : в первый раз для вычисления a_x по формуле (23.1), во второй — для вычисления s_x^2 по формуле (23.2).

— Конечно, если не хочешь поработать немножко головой, то пальцам придется делать двойную работу. А все-таки подумай еще над формулой (23.2) и составь программу.

Мне было неприятно, что такие слова папа произнес в присутствии моей любимой тети. Но, с другой стороны, она ведь сама не знает, как подойти к микрокалькулятору. Как бы то ни было, нужно поскорее составить программу. Взглянув еще раз на формулу (23.2), я решил попробовать раскрыть в ней скобки и перегруппировать слагаемые. Вот что получилось:

$$\begin{aligned}
 s_x^2 &= \frac{1}{n} [x_1^2 - 2a_x x_1 + a_x^2 + x_2^2 - 2a_x x_2 + a_x^2 + \dots \\
 &\quad + x_n^2 - 2a_x x_n + a_x^2] = \frac{1}{n} (x_1^2 + x_2^2 + \dots + x_n^2) - \\
 &\quad - 2a_x \frac{1}{n} (x_1 + x_2 + \dots + x_n) + \frac{1}{n} n a_x^2 = \\
 &= \frac{1}{n} \sum_{i=1}^n x_i^2 - 2a_x a_x + a_x^2 = \frac{1}{n} \sum_{i=1}^n x_i^2 - a_x^2. \quad (23.3)
 \end{aligned}$$

Теперь все очень просто. Ввожу поочередно x_i и накапливаю их сумму, как в программе для a_x , а также сумму их квадратов, вычисляю a_x и s_x . Через 5 мин моя программа была готова.

Программа 47. Вычисление среднего выборочного значения a_x и стандартного отклонения s_x по выборке x_i ($i = 1, \dots, n$)

```
0 П4 П0 П1 С П x2 Вх ИП0 + П0
ХУ ИП1 + П1 КИП4 ИП0 ИП4 БП 04 ÷
П0 С П ИП1 ИП4 ÷ ИП0 x2 — П1 С/П
ν С.П
```

Инструкция. В/О С/П РХ = 0, $x_i = \text{РХ С/П}$ РХ = i , $x_{i+1} = \text{РХ С/П} \dots \text{РХ} = n - 1$, $x_n = \text{РХ С/П}$ РХ = n , БП 19 С/П, РХ = Р0 = a_x С/П, РХ = Р1 = s_x^2 С/П, РХ = s_x . Здесь в качестве подсказки высвечивается номер i уже введенного x_i , так что после этого нужно вводить x_{i+1} .

— Вот и отлично, — сказал пapa. — Теперь бери тетины таблицы, вводи их в ПМК и считай.

Я так и сделал. Для группы из 20 больных, получивших препарат, и для контрольной группы данные x_i были такие (табл. 23.1, 23.2).

Таблица 23.1

Содержание кальция в крови (в миллимолях на литр) для больных, получавших оксидевит

i	x_i	i	x_i	i	x_i
1	2,4	8	2,7	15	2,7
2	2,76	9	2,45	16	2,55
3	2,5	10	2,5	17	2,45
4	2,4	11	2,75	18	2,25
5	2,9	12	2,55	19	2,55
6	2,65	13	2,45	20	2,6
7	2,4	14	2,3		

Таблица 23.2

Содержание кальция в крови (в миллимолях на литр) для контрольной группы больных, получавших обычные лекарства

i	x_i	i	x_i	i	x_i
1	2,2	8	2,05	15	2,0
2	2,3	9	2,35	16	2,05
3	2,2	10	2,2	17	2,15
4	2,25	11	2,25	18	2,1
5	2,1	12	2,45	19	2,25
6	2,05	13	1,95	20	2,35
7	2,0	14	2,15		

При меч ани е. Нормальное содержание кальция в крови от 2,25 до 2,75 ммоль/л. До начала лечения у обеих групп больных было от 1,6 до 2,1 ммоль/л.

В результате вычисления я получил для первой выборки $a_x = 2,5405$, $s_x^2 = 0,025864$, $s_x = 0,16083$, для второй (контрольной) $a_x = 2,170$, $s_x^2 = 0,0171$, $s_x = 0,1308$. При этом $\sigma_a \approx s_x \sqrt{n - 1} \approx 0,03$, т. е. разность a_x для двух выборок, равная 0,3705, не в 3, а в 12 раз больше, чем σ_a . Это позволяет с достаточной убедительностью утверждать, что оксидевит значительно эффективнее, чем градиционные лекарства. На все это вместе с вводом программ и данных ушло около 15 мин.

— Между прочим, — заметил папа, — ты мог бы немного ускорить ввод данных, если бы увеличил все значения x_i в 100 раз. Тогда вместо того, чтобы вводить 2,76, ты бы ввел 276, т. е. нажимал бы не четыре клавиши, а только три. Конечно, полученные при этом значения a_x и s_x^2 нужно разделить на 100, а s_x — на 10 000. Понятно?

Я подумал и понял. Тетя Марина была нескованно счастлива.

— Надо же, — сказала она, — такой малюсенький микрокалькулятор, а так быстро все сосчитал! А я думала, что он пригоден только для того, чтобы играть в «морской бой» и в «космический полет с мягкой посадкой». У меня

в больнице многие больные развлекаются. Ничего, теперь я их заставлю обрабатывать мою статистику.

Тетя Марина быстро переписала формулы и программу 47.

— Я потом как-нибудь разберусь в этом, а пока побегу.

— Смотри, Марина, не останавливайся на этих результатах. Приходи, когда у тебя будет больше времени, и мы посмотрим, какие численные характеристики и статистические критерии подойдут для обработки твоего материала.

— Хорошо, хорошо, обязательно приду, а пока удираю. Времени совсем нет.

И тетя Марина отправилась дописывать свою диссертацию. А я остался в некотором недоумении. Во-первых, какие существуют численные характеристики и статистические критерии? Я спросил об этом папу, но он сказал, что при случае все объяснит. Что ж, будем ждать случая!

Другой вопрос меня заинтересовал еще больше. А как играют в «морской бой» тетины больные? Это, вероятно, очень интересно.

— Честное слово, насчет морского боя ничего не знаю, — сказал папа. — Знаю, что есть много игр с участием микрокалькулятора, но интересных довольно мало. Если очень хочешь, я завтра немного расскажу о них. Кстати, на них ты можешь тоже поупражняться в составлении программ. Вот, если хочешь, посмотри книжку «Игры с микроЭВМ» [4], в которой описано много разных игр. Впрочем, сегодня этим заниматься не будем, а завтра я тебе все объясню.

24

ПОИГРАЕМ?

— Ну, как поиграем сегодня?

Такими словами я встретил папу, вернувшегося с работы.

— Можно и поиграть, — ответил он. — Во что же мы будем играть? В шахматы, в орлянку, в дурака или в балду? А может быть в волейбол?

— Что ты, папа, с микрокалькулятором!

— Можно и с микрокалькулятором. Конечно, играть в волейбол, используя вместо мяча микрокалькулятор, довольно опасно.

— Ты все шутишь, а я серьезно спрашиваю, во что и как можно играть с микрокалькулятором.

— А я серьезно отвечаю, можно играть в разные игры, но не всегда это интересно. Если, например, ты захочешь соревноваться с микрокалькулятором в том, кто скорее разложит большое натуральное число на простые множители, то вероятнее всего ты проиграешь. Вспомни, мама однажды уже пыталась это сделать, но сильно отстала. Это примерно то же самое, что соревноваться с башенным краном в поднятии тяжестей. Поэтому далеко не всякая игра с микрокалькулятором интересна.

Я бы, — продолжал папа, — подразделил игры, в которых участвует микрокалькулятор, на следующие группы: а) игры, в которых ПМК является партнером; б) игры, в которых ПМК играет роль «судьбы»; в) игры, в которых ПМК играет роль арбитра. Кроме того, игры можно подразделить и на другие три группы:

1. Игры, в которых выигрыш определяется только умением игрока. Это практически все спортивные игры, в том числе шашки и шахматы, а также так называемые интеллектуальные игры, например игра в слова. Эта группа игр, в свою очередь, делится на две подгруппы: игры с полной и неполной информацией. В играх с полной информацией каждый играющий знает, что сделал его партнер (или партнеры). Например, в шахматах твой противник не скрывает, какие ходы он сделал. В играх с неполной информацией ход противника до поры-до времени тебе не известен. Так, в игре «чет-нечет» один из соперников задумывает и записывает 0 или 1, а другой пытается отгадать, выигрывая одно очко при удаче и проигрывая при неудаче.

2. Игры, в которых выигрыш того или иного игрока зависит только от случая и не зависит от умения, это так называемые азартные игры. Название «азартная игра» происходит от французского слова *hasard* — случай, риск, и первоначально не имело никакого осуждающего смысла. Но поскольку в такие азартные игры, как «кости», «roulette», многие карточные, играли обычно на деньги, выигрывали или проигрывали иногда целые состояния, то в настоящее время термин «азартный», да и производные от него «азарт», «азартность» приобрели весьма нехоро-

ший смысл. Человек, покупающий карточку «Спортлото», несомненно, обидится, если ты ему скажешь, что он играет в азартную игру, хотя по первоначальному смыслу слова это, безусловно, верно.

Конечно, азартные игры не помогают ни физическому, ни интеллектуальному развитию человека, приносят больше вреда, чем пользы. Однако нет худа без добра. Некогда изучение азартных игр привело к формулировке основных положений теории вероятностей. Здесь основную роль сыграли Паскаль и Ферма. Да и сейчас азартные игры иногда служат моделями для решения серьезных вероятностных задач.

3. Игры смешанные, в которых выигрыш зависит и от умения игрока, и от удачи. Это большая часть карточных игр, которые в старину называли коммерческими, в отличие от азартных, например преферанс, вист, бридж, покер, а также подкидной дурак и домино.

Так вот, игры, в которых ПМК с успехом играет роль партнера, почти всегда относятся к первой группе игр. Поэтому мы и займемся ими.

Начнем с самой простой игры, которую придумал еще в начале XVII в. французский математик Баше де Мезирьяк. Ее с тех пор называют игрой Баше. Есть несколько разновидностей этой игры, мы ограничимся одной. Имеется куча камешков, число которых обозначим S . Каждый из двух игроков по очереди забирает из этой кучки от 1 до N камешков. Выигрывает тот, кто сумеет забрать последний. Поскольку у нас в квартире камешки не водятся, заменим их спичками. Пусть для начала $S = 10$ спичек, а $N = 3$. Итак, ты можешь взять от 1 до 3 спичек. Сколько ты выберешь?

— Ну, скажем, 3.

— Хорошо. Я возьму тоже 3 из оставшихся 7 спичек. Теперь твой ход.

Я задумался. Осталось всего 4 спички. Все взять не могу, хотя бы одну должен. Тогда папа забирает все остальные спички и выигрывает.

— Давай еще раз сначала. На этот раз я возьму одну спичку и оставлю 9.

— Хорошо, — сказал папа. — я тоже возьму одну спичку. Теперь осталось 8.

Я взял 2 спички и оставил 6, папа тоже взял 2 спички, и я снова проиграл.

— Ага, кажется, я понял, в чем секрет. Тот, кто начинает игру, проигрывает, если второй игрок повторяет его ходы.

— Не делай поспешных выводов, — сказал папа. — Попробуем. Я сделаю первый ход и возьму, скажем, 2 спички.

Я, конечно, тоже взял 2 спички, папа взял опять 2 спички, я повторил его ход, папа взял оставшиеся 2 спички и выиграл. Я был в недоумении.

— Ты прав только в одном, — сказал папа. — Во всех играх с полной информацией существует для каждого игрока «оптимальная стратегия» — такой способ игры, который позволяет ему рассчитывать на успех. Он наверняка выиграет, если его партнер сделает ошибку, отклонится от своей оптимальной стратегии.

— А что будет, если оба игрока придерживаются оптимальной стратегии?

— Это зависит от игры. В некоторых играх будет ничья. Я чуть позже покажу тебе такой пример. Но в игре Баше ничьей быть не может: кто-нибудь из игроков возьмет последнюю спичку. Преимущество может иметь начинающий игру или его соперник опять-таки в зависимости от условий игры. В игре Баше, если S не делится на $N + 1$, при оптимальных стратегиях обоих игроков выигрывает первый, а если делится, то второй. Вот в таких играх можно «научить» ПМК играть почти без проигрыша. Практически, если партнер не владеет оптимальной стратегией, ПМК проигрывать не будет. Позже я тебе покажу эту оптимальную стратегию, а пока дам программу для игры Баше и ты сможешь убедиться, что выиграть у ПМК тебе будет очень трудно.

Программа 48. Игра Баше

ИПА	ПД	ИПВ	1	+	ПС	÷	1	+	ПЗ
КИПЗ	ИПД	ИПС	ИПЗ	×	—	П1	$x \neq 0$	41	ИПД
ХY	—	ПД	$x \neq 0$	45	ИП1	С/П	П2	$x \neq 0$	50
ИПВ	ХY	—	$x \geq 0$	51	ИПД	ИП2	—	$x \neq 0$	48
В/О	1	П1	БП	19	7	7	С/П	0	С/П
In	С/П	БП	27	П2	ИПА	ПД	ИПВ	1	+
ПС	ИП2	БП	28						

И н с т р у к ц и я. ($S = PA$, $N = PB$). Если первый ход предоставляетя ПМК, то В/О С/П; высвечивается количество спичек, взятых ПМК. Вводится в РХ количество спичек, взятых соперником ПМК, и нажимается клавиша С/П, опять ПМК выдает количество взятых им спичек. Если ПМК выиграл, он объявляет об этом, высвечивая радостный клич 77. При этом число взятых в последний раз спичек находится в Р1. Если же ПМК проиграл, то выдает 0. Если первый ход делает противник ПМК, то взятое им количество спичек ввести в РХ и нажать клавиши БП 54 С/П, а дальше все, как и в предыдущем случае. В любой момент можно проверить, сколько спичек осталось в кучке. Это число находится в РД. Чтобы ты не смог смошенничать, например не взять ни одной спички или взять больше N , ПМК тебя проверяет. Если ты не взял ни одной спички, на индикаторе увидишь ЕГГОГ, если взял лишние,— количество лишних спичек со знаком минус. После этого ты можешь исправить ошибку —ввести правильное число спичек, нажать клавишу С/П и продолжать игру.

Позабавляйся пока этой программой и пострайся найти алгоритм оптимальной стратегии.

Я набрал программу и ввел $S = 100$, $N = 10$. При первом ходе ПМК я несколько раз проиграл. При моем первом ходе я тоже не смог ни разу выиграть. Вводил различные значения S и N и по-прежнему не мог выиграть, пока не догадался ввести $S = 10$ и $N = 9$. Тогда ПМК своим первым ходом взял одну спичку, а я сразу все оставшиеся 9, на индикаторе увидел 0 — ПМК признал себя побежденным. Но этот выигрыш мне радости не доставил: слишком уж короткой получилась партия.

Тогда я решил проанализировать программу и понять, какая стратегия ПМК в ней заложена. Это оказалось совершенно не трудно. В первых двух строках программы производится запись количества спичек в кучке S в РД, а числа $N + 1$ — в РС, отыскивается остаток от деления содержимого РД на $N + 1$, который записывается в Р1. Затем проверяется, не равен ли этот остаток нулю, и если нет, то он и определяет количество спичек, отираемых ПМК. Если же Р1 = 0, ПМК берет одну спичку. После хода противника ПМК проверяет, не смошенничал ли он (шаги 28 и 33) и не выиграл ли кто-нибудь (шаги 23 и 38), снова определяет свой ход как остаток от деления оставшегося в кучке (РД) количества спичек на $N + 1$. Последние 10 шагов программы служат для того, чтобы при пер-

вом ходе противника заполнить РД и РС. Остальные операции в программе легко понять. Я обратил внимание на использование команды В/О на шаге 40, которая заменяет здесь два шага БП 01.

Итак, можно сформулировать правило, по которому играет почти всегда выигрывающий ПМК. Каждый раз количество взятых спичек равно остатку от деления числа спичек в кучке на $N + 1$, если этот остаток не равен нулю. Если же остаток равен нулю, то ПМК забирает одну спичку и ждет, чтобы его соперник ошибся. Когда к первому ходу ПМК S делится на $N + 1$, а противник соблюдает оптимальную стратегию, ПМК выиграть не сможет, так как все время содержимое кучки D будет нацело деляться на $N + 1$, а это значит, что к последнему ходу ПМК $D = N + 1$ и, сколько бы спичек не взял ПМК, его противник следующим ходом заберет все остальные.

Поняв эту закономерность, я стал выигрывать у ПМК, когда начинал партию, выбирая S , не делящееся на $N + 1$. Я выигрывал и тогда, когда первый ход делал ПМК, но для этого выбирал S , делящееся на $N + 1$.

Через час я подошел к папе и сказал ему, что все понял и теперь в Баше мне играть неинтересно.

— Ладно. Попробуем другую игру с полной информацией. Ты ее, конечно, хорошо знаешь — крестики-нолики.

— Что ты, папа! О ней даже в песне поется, что это детская игра. Я еще в 4-м классе освоил оптимальную стратегию этой игры. Если ты начинаешь игру, ставь свой крестик в среднюю клетку. Если этим ходом начал игру твой противник, то в ответ ставь нолик в одну из угловых клеток (рис. 24.1, а). Дальше нужно только следить, чтобы не дать противнику заполнить какой-либо ряд или диагональ. Если оба игрока придерживаются этих правил, то

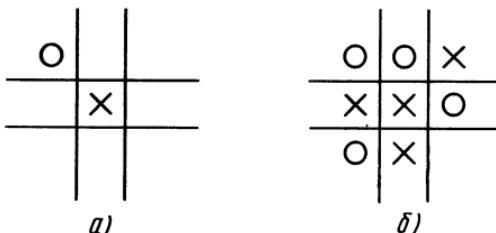


Рис. 24.1

игра обязательно заканчивается вничью (см., например, рис. 24.1, б).

— Ты прав. Как игра «крестики-нолики» такого серьезного человека, как ты, не увлечут. Но как задача для программирования твоя стратегия далеко не проста. Попробуй научить ПМК играть в «крестики-нолики» хотя бы в роли зачинателя (делающего начальный ход). Твоя программа должна быть удобной, обеспечивать отсутствие проигрыша, а также использовать всякую ошибку в стратегии для выигрыша. Подумай над этим.

Я сел и стал думать. Время от времени папа подходил ко мне и подбрасывал некоторые идеи. Прежде всего, чтобы программу можно было представить в обычной форме, нужно придумать удобные обозначения для девяти клеток игрового поля. Я предложил совместить его с цифровыми клавишами 1—9 панели ПМК (рис. 24.2). Тогда центральная клетка имеет номер 5. Именно в нее в начале игры ПМК должен поставить свой крестик.

Затем папа посоветовал выписать наилучшие ходы при различных ответах на этот первый ход ПМК и посмотреть, нельзя ли сгруппировать различные ответы так, чтобы можно было использовать одни и те же последовательности ходов. Я прикинул несколько вариантов и убедился, что это действительно так. Пусть, например, на первый ход ПМК 5 противник ответил 2. Тогда к выигрышу приводит такая последовательность ходов: 6 (4) 9 (в скобках показан вынужденный ход противника). Образуется расположение крестиков-ноликов, при котором выигрыш ПМК обеспечен (рис. 24.3, а). Но точно такая же последовательность ходов будет оптимальной, если на первый ход 5 противник ответил 8 (рис. 24.3, б). В обоих случаях ПМК выигрывает ходом 1 или 3. Для двух других ответов на первый ход ПМК 5, а именно 4 или 6, выигрыш обеспечат следующие ходы: 8 (2) 7.

При ответах противника 1 или 9 гарантировать выигрыш ПМК нельзя. Но вот последовательность ходов, гарантирующих ничью: 6 (4) 7 (3) 2 (8). Если же противник отклонился от своих вынужденных ходов, ПМК получает возможность выиграть.

Наконец, при первых ответах противника 7 или 3 оптимальной (в том же смысле) будет следующая последовательность: 6 (4) 1 (9) 2 (8).

Таким образом, существует всего четыре последовательности ходов, которые должны быть заложены в програм-

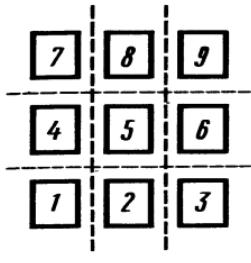


Рис. 24.2

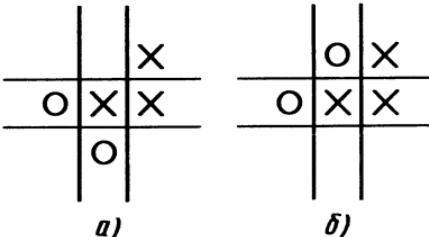


Рис. 24.3

му ПМК. Нужно заложить в программную память четыре отдельные программы и адресовать к ним в соответствии с первым ответным ходом противника. И тут мы вместе с папой придумали способ адресации, использующий существующую симметрию этих ответов.

Обозначим клетку, на которую ставит нолик противник в ответ на первый ход ПМК 5, буквой x . Сформируем величину $D = 5(5 - x)^2$. Получается

$$\begin{array}{llll} D = 5 \cdot 4^2 = 80 & \text{при } x = 1 & \text{или } 9, \\ D = 5 \cdot 3^2 = 45 & \text{при } x = 2 & \text{или } 8, \\ D = 5 \cdot 2^2 = 20 & \text{при } x = 3 & \text{или } 7, \\ D = 5 \cdot 1^2 = 5 & \text{при } x = 4 & \text{или } 6. \end{array}$$

Эти значения D будут адресами каждой из четырех частных программ. Для обращения к ним после ввода ответа противника ПМК вычисляет D и вводит его значение в РД, после чего команда КБПД устанавливает нужный шаг программы.

Мы решили также ввести подпрограмму, которая анализирует ходы противника и наказывает его, как только он отклоняется от своей оптимальной стратегии. Теперь ПМК может выигрывать и в тех случаях, когда оптимальная стратегия гарантирует только ничью. При таком способе адресации некоторые шаги не используются.

При запуске ПМК высвечивается его первый ход — всегда 5. Противник вводит ответный ход и нажимает клавишу С/П, высвечивается очередной ход ПМК. В случае выигрыша ПМК высвечивает свой последний ход, умноженный на миллион. При ничьей высвечивается 0.

В результате получилось
Программа 49. Игра в крестики-нолики

5	С/П —	БП	60	8	С/П ↑	2	ПП	
65	9	С/П ↑	7	ПП	65	↑	БП	70
6	С/П ↑	4	ПП	65	1	С/П ↑	9	
ПП	65	8	С/П ↑	2	ПП	65	Сх	С/П
...	6	С/П ↑	4	ПП	
65	9	С/П ↑	3	ПП	65	↑	БП	70
x ²	5	×	ПД	КБПД	—	x=0 69	В/О	Вх
ВП	6	С/П
6	С/П ↑	4	ПП	65	7	С/П ↑	3	
ПП	65	2	С/П ↑	8	БП	36		

Инструкция. В/О С/П РХ = 5. Ввести номер клетки, на которую Вы ставите нолик, нажать клавишу С/П и прочесть на индикаторе номер клетки, на которую ПМК ставит крестик, и т. д. Высвечивание 0 означает ничью, номер клетки с шестью нулями — выигрыш ПМК. Вместо точек можно ввести в программу НОП либо оставить нули.

Я проверил программу при всех вариантах ходов противника и убедился в ее беспроигрышности для ПМК. Всякое отклонение противника от хода, обозначенного выше как вынужденный, приводит к его немедленному проигрышу.

— Знаешь, — сказал я папе, — хотя это и детская игра, программа получилась интересной.

— Конечно. Я думаю, что работа над ней кой-чему тебя научила. Ты познакомился с некоторыми хитростями, которые могут пригодиться и при решении более серьезных задач.

Тут подошла мама, и мы предложили ей поиграть с ПМК. Проиграв две партии и сведя четыре к ничьей, она согласилась, что ей ПМК не обыграть. Ей понравилось, что ПМК работает в диалоговом режиме — в ответ на мой ход сообщает свой.

— Погоди, — сказал я, — то ли еще будет, когда я зайдусь более серьезными играми. Конечно, такие игры с полной информацией интересны только детям. Я думаю.

что для любой из них можно за пару часов найти оптимальную стратегию и составить программу.

— Ты убежден в этом? — спросила мама.

— Ну, конечно, если игра очень сложная, то может не хватить памяти.

— А для ЭВМ с самой большой памятью и быстродействием ты оптимальную стратегию найдешь?

— Думаю, что рано или поздно справлюсь с этим.

— Тогда ты обессмертишь нашу фамилию. Разработай оптимальную стратегию для шахмат.

— Как? ... — опешил я.

А папа только улыбался.

— Ты забыл, — сказал он, — что, перечисляя игры с полной информацией, я называл и шашки, и шахматы. Существуют оптимальные стратегии для играющих как белыми, так и черными. Но, к счастью, они пока никому не известны.

— Почему «к счастью»?

— Конечно, если бы они стали известны, то интерес к этим играм пропал бы так же, как и к крестикам-ноликам. Сейчас даже неизвестно, кто должен выиграть, белые или черные, если оба игрока будут пользоваться своими оптимальными стратегиями. Скорее всего всегда будет ничья. Но строго это не доказано. Пока известны только более или менее хорошие стратегии для начальной части партии — дебюта. Для концов партии (эндшпилей) при тех или иных расположениях фигур во многих случаях известны оптимальные стратегии партнеров. Поэтому гроссмейстеры, дойдя до таких позиций, либо соглашаются на ничью, либо один из них сдается.

Ну, сегодня мы уже наигрались. Завтра я тебе расскажу о других играх.

25

УГАДАЙ-КА!

Я с нетерпением ждал родителей. Они пришли с работы одновременно, и я напомнил им о вчерашнем обещании.

— Да погоди, — сказал папа, — дай хотя бы отдохнуться. Потом хитро улыбнулся и вытащил из кармана пальто руку, сжатую в кулак.

- Угадай, сколько у меня монет. Если угадаешь, будем сейчас играть, а если нет, после ужина.
- Четыре! — сказал я наугад.
- А вот и не угадал. Их всего три. Значит, сначала поужинаем.

После ужина опять возник вопрос, будем ли мы заниматься играми. Папа снова протянул мне сжатый кулак.

- Сколько теперь монет?
- Четыре, — сказал я.
- Да, на этот раз ты угадал. Как тебе это удалось?
- Немного интуиции и капелька психологии. Я сообразил: ты подумаешь, что я ни в коем случае не повторю то же число, которое назвал в первый раз, и поэтому возьмешь четыре монетки.

— Да, на этот раз ты меня перехитрил. Что же, займемся играми, в которых игроки пытаются перехитрить друг друга. Это, как сам понимаешь, игры с неполной информацией. Простейшая из них, пожалуй, игра «чет-нечет». Один из игроков (*A*) зажимает в кулаке некоторое количество монет, а другой (*B*) пытается отгадать, является ли это число монет четным. Здесь игроки находятся в равных условиях — игрок *B*, делая свой ход (отгадывая), не знает, сколько монет взял *A*. В свою очередь, игрок *A*, делая свой ход (беря монеты), не знает, что скажет игрок *B* (чет или нечет).

При каждом ходе игрок *A* имеет две возможные простые (так называемые чистые) стратегии — он может взять четное или нечетное число монет. Игрок *B* также имеет две стратегии — сказать чет или нечет. Но в отличие от игр с полной информацией здесь не существует оптимальной чистой стратегии. Представь себе, что играют два микрокалькулятора, один по определенной программе «выбирает» четное или нечетное число монет (для удобства будем считать 0 или 1), а другой по своей программе отгадывает его. Предположим, что я — калькулятор *A* — решил, что оптимальной стратегией будет всегда выбор нуля, а ты — калькулятор *B* — всегда давать отгадку единица. Тогда, сколько бы игра не длилась, все время выигрывает *A*. Однако если сменить стратегию микрокалькулятора *B* на противоположную — давать «отгадку» нуль, то, наоборот, все время будет выигрывать *B*.

Теперь предположим, что играют между собой не машины, слепо выполняющие заложенные в них программы, а соображающие люди. Ясно, что, когда игрок *B* после нескольких ходов убедится, что игрок *A* всегда выбирает

нуль, он начнет отгадывать это безошибочно. Но игрок *A*, в свою очередь, заметив, что *B* все время отгадывает 0, быстро сменит свою стратегию — станет выбирать 1.

Может игрок выбрать такую последовательность ходов, при которой в случае длительной игры он не будет оставаться в проигрыше, что бы ни делал другой игрок? В таком смысле оптимальная стратегия существует, только она будет не «чистой», т. е. не предписывает выбирать всегда 0 или 1, а «смешанной». При смешанной стратегии определяется, с какими вероятностями игрок должен выбирать 0 или 1 в каждом ходе. Такая ситуация подробно изучена теорией игр.

— Я надеюсь, — продолжал папа, — что ты уже догадался, в чем состоит оптимальная смешанная стратегия каждого игрока. Он должен при каждом ходе с равной вероятностью выбирать 0 или 1, независимо от того, какие ходы делал он и его противник раньше. Можно, например, подбрасывать монетку и выбирать 0, если выпадет герб, и 1 в противном случае. Если оба игрока в точности придерживаются такой стратегии, то каждый ход с равной вероятностью принесет выигрыш игроку *A* или игроку *B*. Поэтому при достаточно долгой игре она должна закончиться примерно в ничью. Такая «ничья в среднем» обеспечивается игроку, использующему оптимальную смешанную стратегию и тогда, когда его партнер не придерживается оптимальной стратегии, какие бы ходы он ни делал, даже, если он будет повторять все время 1. Действительно, если один игрок применяет оптимальную смешанную стратегию, то что бы ни делал другой, всегда с равной вероятностью их ходы будут либо совпадать, либо различаться, а это и есть «ничья в среднем».

— А можно такую стратегию заложить в программу?

— Конечно. Вот пример программы, предложенной в книге [4], а перед этим опубликованной в журнале. Эта программа содержит всего три шага: XY XY B/O. Перед началом вводится 1 в РХ, а в РY остается 0, и нажимаются клавиши B/O С/П. Тогда в РХ происходит непрерывная смена 0 и 1. Если в наугад выбранный момент остановить ПМК (нажав клавишу С/П), то с равной вероятностью вы светится 0 или 1. Только не нужно очень часто останавливать ПМК, а то нули и единицы будут чередоваться регулярно.

Между прочим, в журнале программу исказили, напечатав С/П вместо B/O, что делает программу неработоспособной. Кроме того, там написано, что при такой про-

грамм ПМК всегда выигрывает. В действительности эта программа гарантирует только ничью в среднем, но не выигрыш. Правда, автор программы в этих искажениях не виноват, они возникли при редактировании.

Я проделал все, как сказал папа, и получил такую последовательность:

```
0 0 0 0 0 1 1 1 0 1 0 0 1 1 0 1 0 1 1 0 0 1 1 1 1 1 1 1 0 0 1  
0 0 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1 0 1 0 1 0 1 0 1 1 0 0 0 0 0
```

Я очень удивился, что с самого начала возникла последовательность из пяти нулей. Я думал, что при случайному выборе нулей и единиц они должны чередоваться чаще. Но папа со мной не согласился. Он заявил, что если я много раз буду записывать такие случайные «потоки» нулей и единиц, то в среднем один раз за 16 попыток я буду получать в начале пять одинаковых цифр. Так что ничего удивительного здесь нет. В общем, из 60 цифр получилось 32 нуля и 28 единиц.

— Почему же неравное число нулей и единиц?

— А ты что хотел, чтобы их было совершенно поровну? Все-таки здесь ты имеешь дело со случаем. Конечно, в среднем здесь должно быть одинаковое количество нулей и единиц. Но отклонения в ту или иную сторону, конечно, возможны. И я уже подчеркивал, что, применяя оптимальную смешанную стратегию и не зная, что будет делать его соперник, игрок обеспечиваетничейный исход только в среднем. Это значит, что если ты будешь играть с ПМК целый день, иногда ты будешь в выигрыше, а иногда ПМК. Но в среднем вы в конце концов останетесь «при своих».

Тут я заметил, что мама очень укоризненно смотрит на папу. Наконец, она не выдержала:

— Что ты морочишь голову ребенку? Зачем нужны эти доморощенные методы получения случайных чисел, когда в вычислительной технике давно используется конгруэнтный метод, не требующий никакого участия пальца?

— Не волнуйся, — сказал папа. — Я ему и про конгруэнтный метод расскажу. Но согласись, что метод с пальцем, если не соблюдать никакого постоянного ритма, действительно дает случайную последовательность нулей и единиц, а твой конгруэнтный метод позволяет получить только псевдослучайную последовательность.

— Но есть проверенные уже программы, при которых получаемая псевдослучайная последовательность удовлет-

ворят всем разумным тестам, и ими с успехом пользуются все вычислители во всем мире.

— Так-то оно так. Но все эти проверенные программы рассчитаны на машины со значительно большим числом разрядов, чем наш маленький Б3-34. Правда, в различных книгах опубликованы программы для получения псевдослучайных чисел на Б3-34, но я пробовал с ними работать и не очень был ими доволен. У меня есть своя программа. Вот ее я и хотел сейчас дать Мише.

Тут вмешался я:

— Сначала объясните мне, что это такое псевдослучайные числа.

— Псевдо — это значит ложный, — ответил папа. — Эти числа не случайные, а вполне закономерные.

— Детерминированные, — вставила мама.

— Но их вполне можно использовать вместо настоящих случайных чисел в различных задачах моделирования. Дело в том, что хотя всякая последовательность псевдослучайных чисел является периодической, но период N у нее очень большой, в простейших программах около 10 000 чисел, а в более сложных — миллиард и даже более. Ясно, что если ты будешь использовать эти числа меньше чем N раз, то обнаружить периодичность не сможешь. А в остальном они ведут себя почти как случайные независимые числа.

Теория конгруэнтного метода получения псевдослучайных чисел очень сложна, да, по-моему, до конца не разработана. Поэтому различные новые алгоритмы приходится проверять, чтобы удостовериться, что выдаваемые числа псевдослучайные.

Основной алгоритм сводится к следующей формуле:

$$X_n = aX_{n-1} + b \pmod{T}, \quad (25.1)$$

где X_n — очередное псевдослучайное число; X_{n-1} — предыдущее число; T — большое число; a , b и T — взаимно простые числа. Надеюсь, ты не забыл, что значит $(\text{mod } T)$?

— Нет, конечно. Это остаток от деления на T . Мы же этим занимались, когда составляли календарь.

— Верно. Эта формула позволяет получить длинную последовательность почти независимых и равновероятных целых чисел от 0 до T . Но чаще приходится пользоваться псевдослучайными дробными числами V_n на интервале от

О до 1. Их легко получить из той же формулы, если разделить X_n на T :

$$V_n = X_n / T. \quad (25.2)$$

Так как нам не нужны исключительно большой период и высокое «качество» псевдослучайной последовательности (ПСП), можно положить $b = 0$ и тогда приходим к очень простой и удовлетворительно работающей формуле

$$V_n = F \{aV_{n-1}\}, \quad (25.3)$$

где $F \{\cdot\}$, как ты помнишь, дробная часть. Конечно, далеко не при всех значениях a и V_0 мы получим удовлетворительную ПСП. Например, при $a = 23$ и $V_0 = 0,672459$ результаты получаются неплохие. Я проверил этот алгоритм на периодичность и убедился, что период его больше 10 000. Думаю, что это тебя устроит: ты же не будешь играть несколько суток подряд. Другие параметры — равномерность распределения и отсутствие взаимной зависимости, тоже удовлетворительные. Отсюда вытекает такая программа.

Программа 50. Получение псевдослучайных чисел, равномерно распределенных на интервале $(0,1)$

ИПА ИПВ × П9 1 + П0 КИП0 ИП9 ИП0
— ПВ С/П БП 00

Инструкция. ($23 = PA$; $0,672459 = PB$ В/О)
С/П РХ = V_n . На вычисление каждого псевдослучайного числа уходит около 5 с.

Я попробовал эту программу и получил такие числа: 0,466557; 0,730811; 0,808653; 0,599019; 0,777437; 0,881051; 0,264173; 0,075979; 0,747517; 0,192891 и т. д.

— Но нам ведь нужно иметь не дробные числа на интервале $(0,1)$, а просто равновероятные цифры 0 и 1, — сказал я.

— Их легко получить из последовательности V_n . Например, преврати все числа, меньшие 0,5, в нули, а остальные — в единицы. Для этого нужно к нашей программе добавить несколько шагов. Устроим конкурс. Пусть каждый из нас придумает вариант этих нескольких шагов начиная с шага 12 (после ПВ).

Мама начала отказываться, уверяя, что не знает микрокалькуляторов, но папа ее пристыдил. Она взялась за дело и предложила такую программу:

0, 5 — x ≥ 0 22 1 С/П 0 С/П

Я эту программу раскритиковал. Во-первых, она неудобна: для перехода к вычислению следующего числа придется нажимать обе клавиши В/О С/П или нужно вводить в программу после каждого С/П еще два шага БП 00. Кроме того, невыгодно вводить половину в виде 0, 5. Можно сэкономить один шаг, если сделать так: 2 1/x.

Я решил сделать программу хитрее:

1, 5 + П1 КИП1 ИП1 С/П

Она получилась короче маминой. Кроме того, я не перехожу в отрицательную область. Вместо того чтобы преобразовывать интервал (1, 0) в интервал (-0,5; 0,5), как это сделала мама, я переношу свои числа в интервал (1,5; 2,5), а затем записываю в Р1 и с помощью операции КИП1 отбрасываю в содержимом Р1 дробную часть и уменьшаю целую часть на единицу.

Папа составил программу, похожую на мою:

2 × 1 + П1 КИП1 ИП1 С/П

В этой программе интервал (0; 0,5) преобразуется в интервал (1, 2), а (0,5; 1) — в (2, 3). Остальное, как в моей программе.

— Все три программы хороши, но моя будет считать чуть-чуть быстрее, — сказал папа, — поэтому возьмем ее. В результате получилась

Программа 51. Получение псевдослучайной последовательности цифр 0,1

ИПА ИПВ × П9 1 + П0 КИП0 ИП9 ИП0

— ПВ 2 × 1 + П1 КИП1 ИП1 С/П

БП 00

Инструкция. 23 = РА; 0,672459 (или любое другое число из последовательности V_n) = РВ (В/О) С/П РХ := C_n (C_n — псевдослучайные числа 0 или 1).

Я набрал программу, ввел 23 в РА, а в РВ последнее из найденных мной чисел $V = 0,192891$. Запустив ПМК, я стал через каждые 10 с получать цифры, которые дали такую последовательность: 0; 0; 1; 1; 1; 1; 0; 1; 0; 1; 0; 1; 0; 0; 1; 1 и т. д.

Я продолжал играть с ПМК, в который была введена программа 51. Ради простоты я загадывал все время единицу, а ПМК иногда отгадывал, а иногда ошибался, так что в среднем действительно получалась ничья. В самом начале я даже проиграл два очка, потом мой проигрыш уменьшился и общий итог колебался вокруг нуля. Но потом как-то программа выдала довольно много нулей, и я оказался в выигрыше на 4 очка. И тут мне в голову пришла мысль.

— А знаете, я берусь в этих условиях выиграть у микрокалькулятора. Предположим, что я загадываю все время единицу. Микрокалькулятор же пользуется оптимальной смешанной стратегией и выдает нули и единицы в том порядке, как я здесь записал:

```
0 0 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 1 0 0 0 1 0 1 0  
0 1 0 0 0 1 0 1 1 0 1 1 0 0 0 1 0 1 0 1 0 1 0 1 1 0 1 0 0 1  
0 1 0 0 0 1 0 0
```

За первые шесть ходов я проигрываю 2 очка. Далее игра идет с переменным успехом, а после 20 ходов я по-прежнему в проигрыше на 2 очка. Но я не унываю и продолжаю играть дальше, уповая на то, что в среднем должна быть ничья. И действительно, за следующие 4 хода я эти 2 очка отыграл, после 29-го хода я уже одно очко выигрываю, но после 40 ходов опять ничья. Далее мне больше везет и после 60-го хода я оказываюсь в выигрыше на целых 4 очка. И тут решаю прекратить игру.

Потом я могу возобновить игру и опять играть до тех пор, пока не окажусь в выигрыше хотя бы на 2—3 очка, и опять игру прекращаю. Таким образом, у меня всегда будет положительный счет выигранных партий, а если бы мы играли с ПМК на деньги, то я бы его разорил.

— Вот, оказывается, какой ты азартный игрок, — сказала мама. — К счастью, все то, что ты говоришь, может произойти только при игре с машиной, когда ты можешь в любой момент начать и закончить игру. К тому же ты сам навязываешь ПМК программу. Если же ты будешь играть не с машиной, а с человеком и начнешь каждый раз предсказывать единицу, то твой партнер быстро это заметит и обыграет тебя. К тому же он не даст тебе закончить партию, когда ты захочешь.

Тут вмешался папа:

— Смешанная равновероятная стратегия является оптимальной, только когда партнер пользуется ею же или

когда стратегия партнера неизвестна. Если же мой партнер пользуется какой-то другой стратегией и я знаю все сделанные им ранее ходы, то смогу под него подстроиться и начну обыгрывать. Для этого я должен определить на основании нескольких его последних ходов, какой ход он вероятнее всего сделает сейчас. Вот если ты составишь такую программу для микрокалькулятора, то он наверняка будет тебя обыгрывать.

— Но в этом случае я буду выбирать совершенно случайную последовательность нулей и единиц и обеспечу в среднем ничью.

— Вряд ли ты сможешь выбирать нули и единицы совершенно случайно, если только не будешь подбрасывать для этого монетку. Когда человек хочет это сделать прямо «из головы», у него всегда появляется скрытая закономерность, и ее можно установить и использовать для выигрыша.

— А знаешь ли, — продолжал папа, — такие программы, позволяющие микрокалькулятору изучить стратегию противника и использовать ее, существуют и описаны в книге [2]. Но они сделаны для БЭ-21. У меня есть другая идея и мы с тобой сейчас такую программу составим.

Пусть для определенности ты будешь игроком *A* и задумываешь свой первый ход $x_1 = 0$ или 1. Чтобы было без обмана, ты свой выбор запишешь на бумажке, но держи ее в стороне, чтобы микрокалькулятор его не просмотрел.

— Ты опять шутишь, папа.

— Ладно, больше не буду. Итак, ты нажимаешь клавишу С/П, и микрокалькулятор, играющий роль игрока *B*, записывает свою отгадку y_1 в Р9. Ты сообщаешь микрокалькулятору (вводишь в РХ) свой выбор x_1 (это нужно для того, чтобы микрокалькулятор мог учитывать твою стратегию и вносить корректизы в свою). Тут же выбираешь и записываешь свой второй ход x_2 , опять нажимаешь клавишу С/П и т. д.

Теперь обучим ПМК выбирать стратегию. Будем хранить в РА, РВ и РС последние три твоих хода x_i , x_{i-1} и x_{i-2} . Этими тремя ходами могут быть 000, 001, 010, 011 и т. д., всего, как ты можешь подсчитать, 8 вариантов. Обозначим эти варианты цифрами:

000—0 010—2 100—4 110—6

001—1 011—3 101—5 111—7

— Так это просто перевод двоичного числа в десятичное.

— Совершенно верно. Предположим, что последними тремя ходами были 1 0 0, что соответствует числу 4. Теперь ПМК (уже после своего ответа) узнает, например, что в качестве x_{i+1} ты выбрал 1. В этом случае он прибавляет 1 к содержимому Р4. Если бы ты выбрал $x_{i+0} = 0$, то он, наоборот, вычел бы 1 из содержимого Р4. Точно так же он поступает с содержимым Р0, если твоими последними ходами были 0 0 0, с содержимым Р1, если последними ходами были 0 0 1, и т. д.

Таким образом, спустя некоторое время после начала игры, в течение которого ПМК может сильно проиграться, в Р4 будет положительное число, если после ходов 1 0 0 ты чаще всего выбирал 1. Учитывая это, ПМК выдаст тоже $y = 1$ с повышенным шансом на выигрыш, если в твоей стратегии существует какая-то закономерность.

Итак, построим на этих основах программу для игры «чет-нечет», которую можно было бы назвать также «кто кого перехитрит». Для большего удобства заставим ПМК также подсчитывать и выдавать величину твоего выигрыша с начала игры.

Программа 52. Для игры «чет-нечет» или «кто кого перехитрит»

1	3	П0	Сх	КП↑	Л0	04	П0	ИПА	ИПВ
2	×	+	ИПС	4	×	+	ПД	КИПД	$x \geq 0$
24	1	БП	25	0	П9	ИПВ	ПС	ИПА	ПВ
ИП8	С/П	ПА	ИП9	С/П	—	x^2	2	×	1
—	ИП8	+	П8	ИПА	2	×	1	—	КИПД
+	КПД	БП	08						

Инструкция. (В/О) С/П РХ = 0, $x_1 = РХ$
С/П РХ = y_1 С/П, ..., РХ = y_i , $x_{i+1} = РХ$ С/П РХ =
 $= y_{i+1}$ С/П ...

Разобраться в этой программе нетрудно. Первые 8 шагов служат для того, чтобы очистить регистры Р0 — Р9, РА — РС. Затем вычисляется десятичный эквивалент $x_i + 2x_{i-1} + 4x_{i-2}$ двоичного числа (x_{i-2}, x_{i-1}, x_i), записанного в РА, РВ, РС, и вводится в РД, после чего косвенным обращением КИПД вызывается число из соответствующего регистра. Это число в первых ходах отгадывания равно нулю и ничего нам не говорит, однако,

как ты сейчас увидишь, после более или менее продолжительной игры оно равно разности $n_1 - n_0$, где n_1 — количество случаев, когда при тех же значениях последних трех задуманных цифр (x_i, x_{i-1}, x_{i-2}) задумывалась 1 (нечет), а n_0 — количество случаев, когда задумывался 0 (чет). Таким образом, если $n_1 - n_0 > 0$, то это значит, что партнер микрокалькулятора в такой ситуации чаще выбирает 1, чем 0, а если $n_1 - n_0 < 0$, то наоборот. Поэтому следующие шаги с 19 по 25 служат для того, чтобы принятые на этом основании решение (наиболее вероятную отгадку) записать в Р9. Затем осуществляется передвижка данных из РВ в РС и из РА в РВ, и на этом ПМК останавливается. При этом ты видишь свой выигрыш, хранящийся в Р8, где вначале записан 0, а предсказание микрокалькулятора не видишь, хотя оно уже готово и хранится в Р9. Это я сделал нарочно, чтобы ты не мог задним числом, зная решение ПМК, изменить выбранную и записанную на бумаге цифру. Сейчас ты ее вводишь в РХ и запускаешь ПМК, который записывает твой выбор x_{i+1} в РА, где ему и надлежит быть, и высвечивает решение y_{i+1} калькулятора. Вот здесь ты можешь увидеть, верно ли ПМК угадал твой выбор.

Шаги с 35 по 43 служат для того, чтобы подсчитать твой выигрыш и записать его в Р8. Если загаданное тобой число и предсказанное ПМК не совпадают, то вычитание на шаге 35 дает ± 1 . Возведя это число в квадрат, умножив на 2 и вычтя единицу, мы всегда получим 1. Если же ПМК угадал, то вычитание на шаге 35 дает 0, а последующие операции превращают его в -1 . Таким образом, при ошибочном решении ПМК к содержимому Р8 добавляется 1, а при правильном — вычитается 1, так что в дальнейшем на шаге 31 всегда будет высвечиваться твой выигрыш g_i на данный момент.

Остальные 8 шагов (с 44 по 51) позволяют ПМК пополнить свой архив. Для этого введенное тобой в РА значение x_{i+1} (0 или 1) преобразуется описанным выше способом в ± 1 , и этот результат прибавляется к содержимому того регистра, номер которого был записан в РД, в соответствии с содержимым РА, РВ, РС. В каждом из восьми регистров Р0 — Р7 постепенно накапливается статистика твоих привычек: какую цифру ты скорее всего загадаешь, если известны три последние загаданные тобой цифры.

— Вот это да! — сказала мама. — У тебя уже микрокалькулятор анализирует привычки человека? Мы у себя на БЦ до этого не додумались.

— Вы занимаетесь серьезными делами, а не играми, — сказал папа. — Проверим нашу программу. Посмотрим, как быстро будет ПМК раскрывать нашу стратегию. Начнем с самой простой — будем загадывать подряд единицы и под ними записывать отгадки ПМК.

Я быстро ввел программу и получил такой результат на 5 ходах:

```
1 1 1 1 1  
1 1 1 1 1
```

— Смотри-ка, ПМК сразу разгадал мою стратегию и не дал мне ни одного очка! Я уже проиграл 5 очков.

— Ладно, — сказал папа. — Смени стратегию на другую чистую и начинай загадывать нули. Первое время ПМК будет по привычке отвечать единицами, но, вероятно, довольно быстро разберется в ситуации.

Я сделал еще 10 ходов нулями и вот что получил в ответ:

```
0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 0 0 0 0 0
```

Итак, 5 очков я отыграл, но ПМК спохватился и изменил свои предсказания. Я снова стал проигрывать и почувствовал, что надо опять менять стратегию.

— Попробуй, — сказал папа, — загадывать поочередно 1 0 1 0 ... Посмотрим, что получится.

Получилось вот что:

```
1 0 1 0 1 0 1 0 1 0  
0 1 1 1 1 0 1 0 1 0
```

Здесь в течение 5 ходов ПМК был в замешательстве, но потом понял, в чем дело, и стал отгадывать безошибочно. В результате я уже проиграл 9 очков.

— Давай-ка я выберу более сложную стратегию — буду загадывать три раза 0 и два раза 1.

— Давай.

Вот результат:

```
0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1  
1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1  
0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1  
0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1
```

Здесь ПМК затратил больше времени на то, чтобы начать отгадывать безошибочно, но уже с первых ходов

он ошибался довольно редко, а мой проигрыш дошел до 33 очков.

— Если бы ты пытался выбирать нули и единицы без всякой закономерности, то, безусловно, некоторое время ПМК ошибался бы очень часто. Но постепенно он бы вскрыл некоторые закономерности, которых ты и сам не замечаешь. В результате он все же будет чаще отгадывать верно. Конечно, это верно лишь в том случае, когда партнер ПМК не знает его алгоритма (программа 52). В противном случае он может рассчитать свои ходы так, чтобы всегда выигрывать. Например, если ты будешь загадывать цифры в таком порядке: 0 1 0 0 0 0 1 1 0 0 1 0 1 0 0 ..., то ПМК, работающий по этой программе, ни разу не отгадает. Сейчас уже поздно, а потом как-нибудь поупражняйся в этом направлении.

— Конечно, поупражняюсь.

— А пока — спокойной ночи.

26

СПОРТ, СПОРТ, СПОРТ

Сегодня, пока родители были на работе, я немного поупражнялся с микрокалькулятором в игре «чет-нечет», но обыграть его хотя бы на 10 очков мне ни разу не удалось. Конечно, когда ПМК работал по программе 51, выдавая нули и единицы случайно, я иногда выигрывал несколько очков, но в последующих ходах этот выигрыш уменьшался, а затем и переходил в проигрыш, так что в среднем имела место ничья. Когда же я вводил адаптивную программу 52, то рано или поздно у ПМК появлялся стойкий выигрыш и мой проигрыш доходил с течением времени до 10, даже если я старался выбирать чет и нечет случайно. Мне это в конце концов надоело. Когда пришел пapa, я набросился на него.

— Ты обещал показать мне спортивные игры с микрокалькулятором!

— Ты имеешь в виду волейбол или хоккей?

Я понял, что пapa шутит.

— Ладно, — сказал пapa. — Самая массовая и увлекательная спортивная игра, если судить не по числу болельщиков, а по числу участников — это Спортлото. Его легко приспособить к микрокалькулятору.

— Что же, ПМК будет покупать билеты, вычеркивать номера, а затем ходить в сберкассу за выигрышем?

— Зачем же. Мы продемоделируем на ПМК тираж. Здесь он будет играть роль судьбы — будет определять случайные выигравшие номера, заменит тиражную машину. Ты, вероятно, видел ее по телевидению.

— Но как же мы будем играть в Спортлото с ПМК?

— Очень просто. Возьмем, например, вариант «5 из 36». Ты берешь бумажку и записываешь на ней 5 выбранных тобой номеров. Затем запускаешь 5 раз ПМК с программой, которую мы сейчас составим, и он выдаст тебе свои 5 номеров. Если хотя бы три из выбранных тобой номеров совпадут с выпавшими в тираже, то считай, что ты выиграл 3 руб. А если совпадут 4 номера, то ты уже можешь выиграть больше. Если же все 5 номеров будут тобой угаданы, то получишь максимальный выигрыш.

— А как же получать выигрыш? Ведь сберкасса его не оплатит.

— Да, конечно, это некоторый недостаток нашей модели. Но зато тебе не придется ждать неделю сообщения о результате тиража. И к тому же не нужно платить по 30 коп. за участие в тираже.

— Это, конечно, не так интересно.

— Чего же ты хочешь, модель есть модель. Немного погодя, я тебе покажу модель управления парусной яхтой. Там все, как в настоящей регате, но ни свежего морского ветра, ни ощущения скорости нет. Зато ты не рискуешь принять неожиданную морскую ванну.

— Ну ладно, как же действуют твои модели?

— Начнем со Спортлото (5 из 36). ПМК должен выбрать 5 случайных натуральных чисел из множества от 1 до 36 с равными вероятностями. Воспользуемся программой 50, которая дает псевдослучайные числа между 0 и 1. Умножим их на 36 и получим также псевдослучайные дроби между 0 и 36. Прибавим к ним 1, т. е. сдвинем ниже множество на интервал (1, 37). Если теперь взять целую часть каждого полученного числа, то с равной вероятностью получим любое целое число от 1 до 36. Итак, получается

Программа 53. Модель Спортлото (5 из 36)

ИПА ИПВ × П9 1 + П0 КИП0 ИП9 ИП0

— ПВ 3 6 × 1 + П8 КИП8 ИП8

С/П БП

Инструкция. (23 = РА, 0,672459 = РВ В/О)
С/П РХ = выигрышный номер С/П... Повторить 5 раз.
Весь тираж (выдача пяти номеров) занимает около 1 мин.

Я ввел программу и предложил папе сыграть. Мы оба взяли по листу бумаги и записали каждый по 5 номеров. Я записал 8, 16, 18, 26 и 33, а папа 1, 2, 3, 4 и 5.

— Разве так выбирают номера? — воскликнул я. — Ведь это же совершенно невероятно, чтобы выпали пять номеров подряд начиная с единицы. Такого еще никогда не было

— А разве более вероятно, что выпадут твои номера? — возразил папа. — Я почти уверен, что никогда еще не выпадали номера 8, 16, 18, 26 и 33.

— Почему?

— Очень просто. Спортлото существует у нас немногим более 10 лет. Может быть, я ошибаюсь. Поэтому возьмем для верности 30 лет. Тиражи бывают один раз в неделю. Значит, всего за 30 лет могло быть $30 \cdot 52 = 1560$ тиражей. А число сочетаний из 36 по 5: $C_{36}^5 = 376\,992$. Чтобы каждое из них могло выпасть, нужно было бы проводить еженедельные тиражи более 7000 лет. А поэтому на каждый выпавший набор номеров приходится по крайней мере 240, ни разу не выпадавших.

Я запустил ПМК, и он выдал мне такие 5 номеров: 17, 27, 30, 22, 28. Все эти номера близки к тем, которые я записал, но ни одного попадания у меня не получилось. А папины номера даже не приближаются к выпавшим.

— Вот видишь, — сказал папа. — Нам с тобой не повезло. Давай сохраним наши номера и проведем еще один тираж.

Я еще раз провел серию из пяти запусков и получил такие номера: 32, 10, 3, 27, 7. Здесь один номер совпал с папиным, а мне опять не повезло. Впрочем, за один отгаданный номер в Спортлото ничего не платят, так что можно считать, что папе повезло не больше, чем мне.

Мы продолжили игру и провели 10 тиражей. При третьем тираже у нас с папой совпало по одному номеру, затем в двух тиражах подряд у нас не было ни одного «попадания» и, наконец, в 6-м тираже у меня совпало два номера 8 и 18. Более высоких результатов не достиг ни один из нас. На 10-м тираже произошел казус — выпали номера 23, 19, 2, 32, 32. Два номера оказались одинаковыми.

— Но ведь этого в Спортлото не может быть, — закричал я. — Если какой-либо номер выпал, то он в этом тираже уже больше не участвует.

— Ты совершенно прав. Наша модель не вполне точна. Она соответствует так называемой схеме возвращенных шаров. При Спортлото действует схема невозвращенных шаров — выпавший шар в данном тираже больше не участвует. Почему я пошел на неточность и смоделировал не ту схему, которая имеет место в действительности? Во-первых, потому, что разница между ними ничтожна. Ты видел ведь, что 9 тиражей прошли так же, как если бы наша модель была точной — все 5 номеров, выпавших в каждом тираже, были различны. Во-вторых, точная модель, построенная по схеме невозвращенных шаров, была бы очень сложной. Конечно, при желании можно уложить такую программу в память БЗ-34. Но считать она будет медленно, тираж удлиняется в несколько раз и играть будет совсем скучно. Поэтому я предпочел взять не вполне точную модель. Нужно только в инструкции дописать: «В случае повторения в данном тираже двух выданных чисел одно из них отбрасывается и микрокалькулятор запускается еще раз». Вот и сейчас, в 10-м тираже, мы одно из двух чисел «32» отбросили и нажали лишний раз клавишу С/П. В результате получили число 9, так что результатом 10-го тиража оказались номера 23, 19, 2, 32 и 9, без каких-либо повторений. После этого все опять идет своим чередом.

— У меня еще один вопрос. Каждый раз, когда я начинаю играть в Спортлото, ПМК выдает в первом тираже числа 17, 27, 30, 22, 28, во втором тираже — 32, 10, 3, 27, 7 и т. д., т. е. все будет повторяться. Зная эту псевдослучайную последовательность, я смогу безошибочно выигрывать. Значит, опять твоя модель не соответствует реальному Спортлото, в котором номера выпадают действительно случайно, а не псевдослучайно?

— Конечно, но это поправимо. Просто нужно внести действительно элемент случайности. Для этого в начале вводи в РВ не фиксированное число $V_0 = 0,672459$, которое я указал только потому, что для него я точно проверил отсутствие периода короче 10 000 и равновероятность всех выдаваемых псевдослучайных чисел. Но можно практически с таким же успехом использовать и любую другую десятичную дробь, не менее чем с шестью десятичными разрядами. При этом ты получишь другую ПСП, ее период, вообще говоря, может быть меньше чем 10 000, но на игру тебе его наверняка хватит, если только в твоем V_0 последняя цифра не будет 0 или 5, а то период может оказаться очень коротким. Когда мне на работе пришлось моделировать одну систему телефонной связи и я пользо-

вался таким генератором ПСП, то поступал следующим образом. Я езжу на работу автобусом и всякий раз сохранял свой билет. Придя на работу, вводил в машину в качестве V_0 шестизначный номер своего билета, разделенный на 10^6 . В тех редких случаях, когда он оканчивался цифрой 5 или нулем, я заменял эту последнюю цифру любой другой.

— Вот, оказывается, на какие хитрости идут научные сотрудники. А я-то думал, что у вас все строго по науке без всяких лотерей ...

— Ничего не поделаешь, в современной науке приходится нередко моделировать случайность, и без таких хитростей не обойдешься.

Мы еще немного поиграли в Спортлото, не только в варианте «5 из 36», но и в другом — «6 из 45». Это делается очень просто. В программе 53 вместо цифр 3 и 6 на шагах 12 и 13 вводятся 4 и 5, а каждый тираж состоит из шести пусков программы. При этом варианте после долгих попыток и мне и папе удалось несколько раз угадать три номера, но большего мы не добились.

Наконец, нам это надоело, и я вспомнил, что папа обещал мне показать еще управление парусной яхтой.

— Такая программа есть в книге [2]. Там она сделана для микрокалькулятора Б3-21, но ее легко перевести на язык Б3-34, чем мы и займемся. Так как в Б3-34 возможностей больше, то мы ее немного дополним — в исходной программе скорость ветра фиксирована, а мы предусмотрим возможность ее изменять, кроме того, учтем, что скорость яхты меняется не мгновенно.

Задача заключается в следующем. Данна карта некоторой акватории с берегами, островами, рифами, и на ней указаны два пункта *A* и *B*. Нужно за минимальное время (число ходов) пройти на яхте из точки *A* в точку *B* и вернуться обратно, не наткнувшись на берег и не сев на мель. Точность прихода в точку *B* и возвращения в точку *A* должна быть не хуже ± 2 м по каждой координате. Тебе не приходилось водить парусную яхту. Поэтому расскажу, как это делается.

— А разве ты, папа, умеешь?

— Конечно, не умею, но кое-что об этом знаю. Так вот, ты управляешь рулем и парусом. Рулем держишь курс яхты — угол Φ направления от кормы к носу по отношению к меридиану (рис. 26.1, *a*). В навигации принято отсчитывать Φ относительно направления на север, причем по часовой стрелке, а не против нее, как привыкли счи-

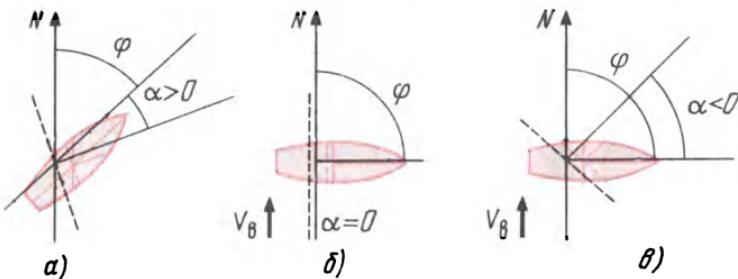


Рис. 26.1

тать математики. Будем придавать φ только положительные значения от нуля до 360° : направлению на восток соответствует $\varphi = 90^\circ$, на юг — $\varphi = 180^\circ$, на запад — $\varphi = -270^\circ$. Ты должен регулировать также положение паруса. Мы будем его отсчитывать по углу α , который образует перпендикуляр к плоскости паруса с осью яхты. По направлению этого перпендикуляра, как ты, вероятно, уже сообразил, действует сила ветра F . Но сила эта действует на яхту по-разному в зависимости от того, как она приложена. Если $\alpha = 0$, то сила направлена от кормы к носу, а так как яхта обтекаемой формы, то легче всего будет ее разгонять. Если же $\alpha = \pm 90^\circ$, то сила будет направлена поперек яхты и почти не сдвинет ее с места. При всяком другом α разложим силу F на две составляющие — направленную по курсу $F_k = F \cos \alpha$ и направленную «к бортам» $F_b = F \sin \alpha$, и будем в нашей модели считать, что только составляющая F_k приводит яхту в движение.

— Но тогда, папа, лучше всего устанавливать всегда $\alpha = 0^\circ$ при этом $F_k = F$ и используется вся сила ветра.

— Это верно только в одном случае, когда ты хочешь плыть всегда по ветру. Если же ты захочешь поплыть по направлению, перпендикулярному направлению ветра (скажем, прямо на восток, т. е. $\varphi = 90^\circ$, при южном ветре), и установишь $\alpha = 0$, то плоскость твоего паруса (показанная штриховой линией) будет лежать в направлении ветра (рис. 26.1, б), который будет скользить мимо паруса, и паруса повиснут, а сила F будет равна нулю. А вот если ты возьмешь, скажем, $\alpha = -45^\circ$ (рис. 26.1, в), то при $\varphi = 90^\circ$ сила $F > 0$ и $F_k = F \cos(-45^\circ) \approx 0,7F$. Маневрируя выбором φ и α , мы можем заставить яхту идти даже

против ветра, точнее, сможем лавировать, двигаясь с отклонениями то вправо, то влево. Для этого и нужно выбирать наиболее удачные сочетания значений ϕ и α , что поможет проделать необходимый путь в кратчайшее время.

Чтобы построить нужную нам математическую модель, необходимо обратиться к теории, что мы сейчас и сделаем. Для простоты предположим, что дует достаточно сильный южный ветер. В программе, приведенной в книге [2], ветер северный. Но южный ветер, безусловно, приятнее, даже в теплую погоду, поэтому введем это небольшое изменение. В дальнейшем мы сможем усложнить нашу программу и выбирать произвольное направление и силу ветра, благо в БЭ-34 для этого хватит регистров памяти и длины программы.

Итак, переходим к теории, из которой известно, что сила, приложенная к парусу, пропорциональна квадрату скорости ветра v_B , точнее, квадрату той составляющей скорости ветра, которая направлена перпендикулярно к плоскости паруса. В данном случае, когда ветер южный, это значит, что $F_0 = k_1 [v_B \cos(\phi + \alpha)]^2$. Здесь k_1 — коэффициент пропорциональности, зависящий главным образом от площади паруса, а также от плотности воздуха. Индекс 0 при F означает, что мы вычислили силу, действующую на неподвижную яхту. Но ведь на самом деле яхта движется со скоростью v под воздействием этой силы. Если бы ветра не было, но яхта двигалась бы по своему курсу (предположим, что у нас яхта с мотором), то на парус действовала бы тормозящая сила, определяемая его движением относительно неподвижного воздуха. Легко сообразить, что эта тормозящая сила соответствует тому значению F_0 , которое имело бы место, если бы яхта была неподвижна, а на парус дул ветер со скоростью v в направлении — ϕ . Поэтому для результирующей силы ветра относительно движущегося паруса нужно взять величину $v_B \cos(\phi + \alpha) - v \cos \alpha$, что дает для силы F выражение

$$F = k_1 [v_B \cos(\phi + \alpha) - v \cos \alpha]^2 \quad (26.1)$$

или

$$F_k = k_1 [v_B \cos(\phi + \alpha) - v \cos \alpha]^2 \cos \alpha. \quad (26.2)$$

Эта сила вызывает движение яхты по курсу, так что скорость v быстро нарастает. Одновременно с увеличением v возрастает сила сопротивления F_c воды и воздуха движе-

нию яхты. Относительно этой силы известно, что она пропорциональна квадрату скорости яхты:

$$F_c = k_2 v^2, \quad (26.3)$$

где k_2 — коэффициент пропорциональности, зависящий от формы яхты и ее материала, определяющих обтекаемость. Таким образом, пока скорость v мала и $F_k - F_c > 0$, яхта набирает скорость, при этом F_k уменьшается, а F_c возрастает. Поэтому довольно быстро (практически через несколько секунд) наступает равенство этих сил $F_k - F_c = 0$. При достижении этого равенства ускорение яхты становится равным нулю, т. е. устанавливается некоторая неизменяющаяся скорость v . Найдем ее, приравняв F_k и F_c :

$$k_1 [v_B \cos(\varphi + \alpha) - v \cos \alpha]^2 \cos \alpha = k_2 v^2,$$

откуда

$$v = k v_B \cos(\varphi + \alpha) \sqrt{\cos \alpha} / (1 + k \sqrt{\cos^3 \alpha}), \quad (26.4)$$

где $k = \sqrt{k_1/k_2}$.

Представим процесс управления яхтой таким образом. Вначале яхта неподвижна в точке A с координатами x_0, y_0 . В момент старта мы устанавливаем начальные значения $\varphi = \varphi_0$ и $\alpha = \alpha_0$ и отдаем себя на волю ветра. За некоторое время Δt установится скорость v_1 , которую вычислим по (26.4). В этот момент мы можем изменить φ и α либо, если найдем нужным, сохранить прежние значения. Так же поступаем и дальше. Таким образом, в начале i -го интервала времени Δt мы имеем скорость v_{i-1} , а в конце — скорость v_i . Составляющие этих скоростей по осям координат x и y , очевидно, будут $v_{x,i-1} = v_{i-1} \sin \Phi_{i-1}$, $v_{y,i-1} = v_{i-1} \cos \Phi_{i-1}$, $v_{x,i} = v_i \sin \Phi_i$ и $v_{y,i} = v_i \cos \Phi_i$. В первом приближении можно считать, что на этом интервале яхта двигалась по оси x со скоростью $0,5(v_{x,i-1} + v_{x,i})$, а по оси y со скоростью $0,5(v_{y,i-1} + v_{y,i})$ и координаты ее изменились на $\Delta x = \Delta t (v_{x,i-1} + v_{x,i})/2$ и $\Delta y = \Delta t (v_{y,i-1} + v_{y,i})/2$. Это было бы совершенно верно, если бы действительная скорость изменилась линейно от v_{i-1} до v_i . Но при достаточно малом интервале Δt это будет очень близко к действительности. Таким образом мы можем определить изменения координат яхты на каждом интервале Δt и построить по точкам ее траекторию на нашей карте.

Коэффициент k для простоты будем считать равным 1. Между прочим значения его не очень сильно влияют на результат: k входит и в числитель и в знаменатель (26.4). Величину Δt примем равной 10 с, площадь акватории примерно 300×300 м.

Скорость ветра выберем близкой к 10 м/с. При скорости $v_b < 5$ м/с наша яхта будет двигаться как черепаха, а при $v_b > 25$ м/с таких неумелых яхтсменов, как мы с тобой, просто нельзя подпускать к воде. При выбранных данных наше путешествие будет длиться 5...8 мин. На микрокалькуляторе оно займет больше времени.

Определим роли некоторых регистров памяти. Координаты яхты будем записывать в Р2 и Р3, скорость ветра v_b (м/с) (который пока считаем южным) в Р0, $\varphi = P4$, $\alpha = P5$.

Программа 54. Игра «Парусная регата»

```

P4 0    ПС  ПД ИП5 С/П П5 cos ↑   √
P6 ×    1    +    П7 ИП4 ИП5 +    cos ИП6
      × ИП0 × ИП7 ÷ 5    ×    П8 ИП4 sin
      × ПА ИПС + ИП2 +    П2 С/П ИПА ПС
ИП8 ИП4 cos × ПВ ИПД + ИП3 +    П3
С/П ИПВ ПД ИП4 С/П П4 БП 04

```

Инструкция. ($x_0 = P2$, $y_0 = P3$, $v_b = P0$)
 $\varphi = RX(B/O)$ С/П, $\alpha = RX$ С/П $RX = x_i$ С/П, $RX = y_i$ С/П, $RX = \varphi_{i-1}$, $\varphi_i = RX$ С/П $RX = \alpha_{i-1}$, $\alpha_i = RX$ С/П ... Переключатель Р — Г лучше поставить в положение Г и вводить углы в градусах.

Когда папа записал программу, он предложил мне ее проанализировать. Теперь я с этим справляюсь хорошо, да и здесь все очень просто. При первом запуске программы угол φ записывается в Р4 и очищаются регистры РС и РД. Затем вводится значение угла α , которое записывается в Р5, затем вычисляется $\sqrt{\cos \alpha}$ (записывается в Р6) и $1 + \sqrt{\cos^3 \alpha}$ (записывается в Р7). С помощью этих величин, полагая $k = 1$, вычисляется v_i . Но почему-то оно умножается на 5 и только потом записывается в Р8. Откуда взялась эта пятерка?

— Я не стану тебе подсказывать. Сейчас сам догадаешься. Посмотри, что было бы без этой пятерки.

— Без этой пятерки мы бы записали в Р8 величину v_i , а потом вычислили ее проекцию на оси x и y и записали их в РА и РВ. Затем нужно сложить их с проекциями v_{i-1} на те же оси. Очевидно, они хранятся в РС и РД. Согласно нашим формулам нужно было бы эти суммы разделить пополам и умножить на Δt , чтобы получить приращения координат. Теперь все понятно. У нас $\Delta t = 10$, так что нужно наши суммы умножить на 5. А ты в своей программе сразу умножил v_i на 5, так что суммы оказались уже умноженными и равны приращениям координат Δx и Δy .

— Ну конечно, — сказал папа. — Вместо того чтобы умножать на 5 два раза, я сделал это в один прием, поэтому сократил программу на 2 шага и немного ускорил вычисления.

— Дальше все понято. Мы прибавляем Δx к содержимому Р2 и Δy к содержимому Р3 и получаем новые координаты яхты, которые поочередно высвечиваются на индикаторе. А значения $5v_{x,i}$ и $5v_{y,i}$ из РА и РВ переписываются в РС и РД, где они на следующем этапе играют роль $5v_{x,i-1}$ и $5v_{y,i-1}$. Затем высвечивается старое значение φ , которое можно сохранить или изменить. Оно записывается в Р4. Последняя двухшаговая команда БП04 служит для того, чтобы обойти команды очистки РС и РД, нужные только в самом начале. Потому высвечивается старое значение α , которое при необходимости изменяется и вновь записывается в Р5, и дальше в том же духе.

— Теперь возьмем лист клетчатой бумаги и нарисуем карту (рис. 26.2). Начальную точку A папа выбрал с координатами $x_0 = 0$ и $y_0 = 0$. И мы вместе с папой стали пытаться доплыть из точки A в точку B с координатами $x = 238$, $y = 160$ и обратно. Ветер, как я уже говорил, южный, $v_w = 10 \text{ м/с}$ (в Р0). Вначале у нас ничего не получалось. Мы иногда плыли совсем не в том направлении, куда хотели, и очень часто оказывались на мели. Но постепенно мы оба приобретали опыт и действовали более успешно. Прежде всего, мы убедились, что α нужно всегда выбирать в пределах от -90° до $+90^\circ$. Причем если мы плывем на восток (при южном ветре), то нужно выбирать $\alpha < 0$, а если на запад, $\alpha > 0$. В конце концов, я удовлетворительно справился с заданием. Вот какие значения φ и α я выбирал и какие координаты получил (с округлением до 0,1 м):

i	φ_i	α_i	x_i	y_i	i	φ_i	α_i	x_i	y_i
0	—	—	0	0	20	260	50	100,9	149,8
1	120	-60	11,3	-6,5	21	260	50	67,4	143,8
2	120	-60	33,9	-19,6	22	260	50	33,9	137,9
3	150	-70	47,4	-29,8	23	220	70	11,8	128,6
4	150	-70	51,6	-37,1	24	120	-60	17,8	115,7
5	150	-70	55,8	-44,4	25	120	-60	40,4	102,6
6	100	-50	74,7	-51,1	26	220	60	48,8	92,6
7	80	-50	114,0	-50,0	27	220	60	42,9	85,6
8	20	-10	145,0	-22,8	28	220	60	37,1	78,7
9	20	-10	161,9	23,6	29	220	60	31,3	71,8
10	20	-10	178,8	70,1	30	220	70	23,0	61,9
11	20	-10	195,6	116,5	31	220	70	12,3	49,2
12	60	-20	221,0	149,4	32	220	70	1,6	36,4
13	100	-10	237,8	159,2	33	150	-60	-3,7	30,0
14	300	30	218,5	170,3	34	150	-60	-3,6	29,0
15	300	30	179,9	192,6	35	120	-60	7,5	23,4
16	220	70	155,2	197,4	36	120	-60	30,1	10,4
17	220	70	144,5	184,6	37	260	40	28,6	1,6
18	220	70	133,7	171,9	38	270	40	-1,1	-0,7
19	220	70	123,0	159,1					

Весь путь занял 38 интервалов Δt , т. е. 380 с или 6 мин 20 с. Из них на первую половину пути, от A до B , ушло 2 мин 10 с — почти всю дорогу ветер был попутный, а на путь от B до A при встречном ветре пришлось затратить 4 мин 10 с. Эти минуты относятся, конечно, к условному «игровому» времени, в течение которого плывет яхта. Что же касается микрокалькулятора, то он для вычисления каждой точки затрачивает не 10 с, а почти в 7 раз больше (папа сказал, что он работает не в реальном масштабе времени). Поэтому я потратил на составление этой таблицы около часа. Да еще примерно 3 ч у нас с папой ушло на первые неудачные попытки.

Папа убежден, что при более продуманном управлении можно пройти по этому маршруту значительно быстрее. Но у меня уже не хватило сил на новые эксперименты. Я думаю найдутся желающие испытать свои силы на этой задаче. Я сказал папе, что из всех показанных им игр эта — самая увлекательная.

— Конечно. Особенno, если ее разнообразить, составляя всякий раз новую карту. Хорошо было бы также выбирать различное направление ветра. Для этого нужно в начале игры ввести в свободный регистр, например Р1, направление ветра, например 180° для южного, 45° для северо-восточного и т.д. Программу при этом придется чуть-чуть удлинить. Как именно, подумай сам.

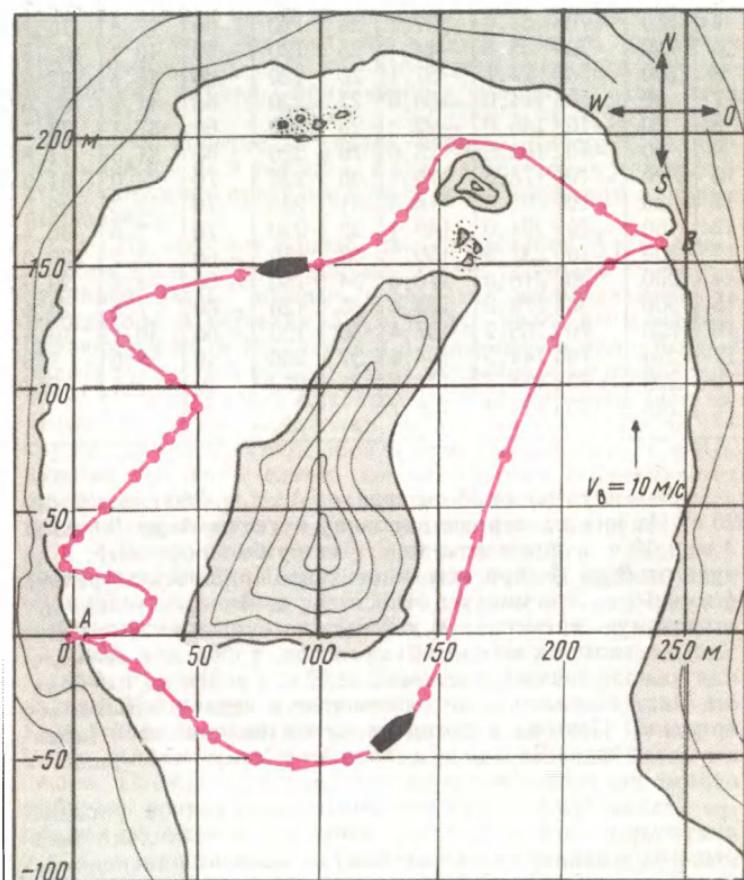


Рис. 26.2

Когда-нибудь я переделаю программу 54, введу в нее свободный выбор направления ветра. А может быть это сделает кто-нибудь другой?

Тут я вспомнил, что тетя Марина называла такие игры, как «морской бой» и «космический полет», и стал приставать к папе, чтобы он мне их показал.

— Почему ты решил, что я знаю все игры на свете? Да если бы и знал, то не стал бы терять время на твое обучение. Ведь я взялся учить тебя программированию для ПМК, а вовсе не убивать время. Если уж тебе так хочется поиграть, то читай популярные журналы, например «Технику молодежи» (№ 6—9 за 1985 г.). Там приведены программы для космических полетов. А еще лучше сам придумай игру и составь программу для нее.

Я твердо решил, что займусь этим. Но только после того, как овладею программированием.

27

ИНТЕГРАЛ? НЕТ НИЧЕГО ПРОЩЕ

Сегодня суббота. Я с утра зашел к папе и увидел, что он сидит перед своим микрокалькулятором и ждет решения.

— Ты что считаешь? — спросил я.

— Да вот, мне нужно вычислить несколько довольно сложных интегралов. Обычно я отдаю такие интегралы на ВЦ. Но вчера не успел, а ждать до понедельника не хочется. От значений этих интегралов зависит направление моей работы, и поэтому я хотел бы их знать поскорее.

— А что такое интеграл? —

— Разве ты не знаешь? Хотя да, у вас в 8-м классе еще об интегралах не говорили. Постараюсь тебе объяснить, может быть не очень строго.

Пусть имеется некоторая функция $y = f(x)$. Каждому значению x соответствует некоторое определенное значение y . Для начала предположим $y \geq 0$. Можно построить график этой функции (рис. 27.1, а). Рассмотрим отрезок оси x от a до b . Так вот, площадь фигуры, которую я заштриховал и которая ограничена снизу осью x , сверху кри-

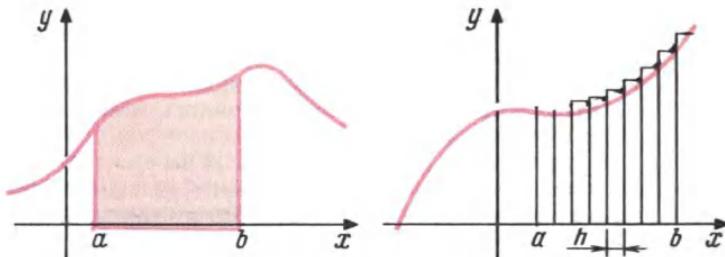


Рис. 27.1

вой $f(x)$, а по бокам ординатами a и b , называется **интегралом функции $f(x)$ в пределах от a до b** .

— А для чего нужно знать эту площадь?

— Это очень часто бывает нужно. Вот простейший пример. Пусть ось x представляет собой желоб, в который насыпан неравномерно песок, так что при каждом значении x высота слоя песка равна y . Тогда наш интеграл означает общее количество песка в желобе между точками a и b . Немного погодя, я приведу тебе другие примеры.

Попробуем научиться вычислять интеграл. Разделим наш отрезок ab на n равных частей. Число n выберем достаточно большим так, чтобы на каждой из этих частей (полосок) значение y очень мало изменялось (рис. 27.1, б). Ширину каждой полоски обозначим dx .

— А почему двумя буквами?

— Буква d означает дифференциал. Но это совершенно не важно. Заметим только, что для каждого значения x имеется полоска шириной dx , а значение y на всей этой полоске примерно одинаково и равно $f(x)$. Тогда наша полоска представляет собой почти прямоугольник и площадь ее равна $f(x) dx$. Это будет тем точнее, чем больше n , и, значит, чем меньше dx . Дальше тебе уже, вероятно, все ясно. Сложим площади всех наших полосок и получим интеграл.

Сформулируем это немного строже. Чтобы получить точный интеграл, необходимо перейти к пределу, устремив

n к бесконечности, а dx — к нулю. Это можно записать так:

$$\lim_{\substack{n \rightarrow \infty \\ dx \rightarrow 0}} \sum_{x=a}^b f(x) dx.$$

С обозначением суммы Σ ты уже знаком. Но в старину, когда зародилось понятие интеграла, это обозначение не было общепринятым. Лейбниц, который одновременно с Ньютоном и независимо от него разрабатывал дифференциальное и интегральное исчисление, обозначал сумму латинской буквой S . Ее он и применил для обозначения интеграла. А для того, чтобы не путать интеграл с исходной буквой S и не записывать при этом предельный переход, он несколько деформировал ее, вытянув так, что получилось нечто вроде змеи:

$$\int_a^b f(x) dx = \lim_{\substack{n \rightarrow \infty \\ h \rightarrow 0}} \sum_{x=a}^b f(x) h. \quad (27.1)$$

Я здесь несколько изменил обозначения, а именно я обозначил ширину полоски через $h = (b - a) / n$, оставив dx как символ этой ширины при предельном переходе, когда мы уже устремили n к бесконечности. Вот, в сущности, и все об интеграле с конечными пределами. Впрочем, нужно учесть еще одну вещь. Я ведь рисовал тебе графики, в которых y был положительным. Но это вовсе не обязательно. Посмотри, например, на такую функцию (рис. 27.2, а). Здесь, как видишь, площадь и над осью x и под ней. Если ты посмотришь на формулу (27.1), то поймешь, что первая площадь войдет со знаком плюс, а вторая — со знаком минус.

— Значит, могут быть и отрицательные интегралы?

— Конечно. Могут быть и нулевые. Вот, например, функция $y = f(x) = \sin x$ (рис. 27.2, б), и я беру ее интеграл от $a = 0$ до $b = 2\pi$. Как видишь, этот интеграл равен нулю, как разность между двумя одинаковыми площадями.

— А какой смысл имеет отрицательный интеграл? Ведь отрицательного количества песка не бывает.

— Что-то у тебя на моем примере с песком свет клином сошелся. Интегрирование применяется во многих слу-

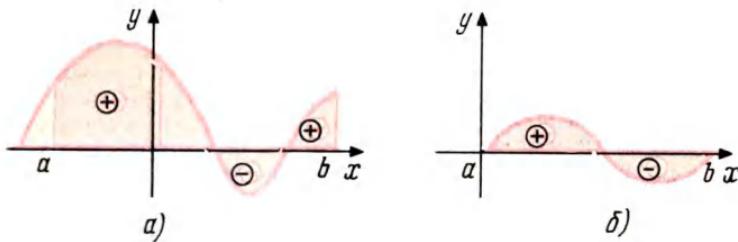


Рис. 27.2

чаях, когда отрицательный интеграл имеет смысл. Пусть, например, x означает время, а $y = f(x)$ — скорость, с которой перемещается вдоль прямой некоторое тело. Для ясности перенесем обозначения x на t и y на v , так что нашей функцией будет $v = f(t)$. Вычислим пройденный путь S за отрезок времени t от 0 до T . Если ты немного подумаешь, то поймешь, что $S = \int_0^T v(t) dt$. Если скорость изменяется и по значению, и по направлению, то и пройденный путь может иметь разные знаки. В частности, при равноускоренном движении, когда $v = v_0 + gt$, имеем $S = \int_0^T (v_0 + gt) dt$. По правилам интегрирования, которых ты еще не знаешь, этот интеграл легко вычисляется и дает следующий результат:

$$S = v_0 T + gT^2/2. \quad (27.2)$$

Мы получили хорошо известную тебе формулу, которой ты пользовался, составляя свои первые программы. Как помнишь, если v_0 и g имеют разные знаки, то и S может принимать отрицательные значения.

— А как вообще вычислять интегралы?
 — Для многих функций можно вычислить интегралы по различным формулам. Так, если $f(x) = x^k$, то $\int_a^b x^k dx = (b^{k+1} - a^{k+1})/(k+1)$, если $f(x) = 1/x$, то $\int_a^b (1/x) dx =$

$= \ln b - \ln a$, если $f(x) = \sin kx$, то $\int_a^b \sin kx dx =$
 $= -(\cos ka - \cos kb)/k$, и т. д. Есть специальные таблицы значений тех или иных интегралов. Но очень часто интеграл не выражается в элементарных функциях и отсутствует в таблицах, тогда приходится его определять численно. Вот здесь и пригодится ЭВМ или даже ПМК. Это, в сущности, нетрудно.

— Да, я уже понимаю. Нужно запрограммировать вычисление функции $y = f(x)$. Это будет вспомогательным алгоритмом или подпрограммой. Затем нужно выбрать достаточно большое значение n и вычислить $h = (b - a)/n$. После этого по порядку вычислять значения функции $f(x)$ в середине каждой полоски и умножать его на h и все эти произведения сложить. Вот и будет интеграл.

— Можно и так. Это будет самый простой способ численного интегрирования. Его называют интегрированием по формуле прямоугольников. Но это далеко не самый точный и быстрый метод. Все же начнем с него. Попробуй записать алгоритм и программу.

Я сел в сторонку и стал писать и зачеркивать, пока не получил такой алгоритм:

```

алг ИНТЕГРАЛ ПРЯМОУГОЛЬНИКОМ (вещ a, b, I)
    арг a, b,
    рез I
    нач цел n, вещ h, x,
        h := (b-a)/n; x := a + h/2; I := 0
    пока x < b
        цикл ФУНКЦИЯ (x, y)
            I := I + y; x := x + h
    кон
    I := I * h
кон

```

Вспомогательный алгоритм ФУНКЦИЯ вычисляет $y = f(x)$.

По этому алгоритму я составил программу, в которой пределы интегрирования a и b записываются в РА и РВ, n — в Р0, h — в Р1 и I — в РС.

Программа 55. Численное интегрирование по формуле прямоугольников

```

П0 0 ПС С/П ПА С/П ПВ XY — ИП0
÷ П1 2 ÷ ИПА + П2 ПП 34 ИПС
+ ПС ИП2 ИП1 + П2 ИПВ — x≥0 17
ИПС ИП1 × С/П

```

Инструкция. $n = \text{РХ В/О С/П}$, $a = \text{РХ С/П}$, $b = \text{РХ С/П РХ} = I$. Подпрограмма вычисления $f(x)$ вводится с шага 34, причем x помещено в Р2, а $y = f(x)$ должно находиться в РХ.

Эту программу я показал папе.

— Ну что ж, — сказал он. — Это программа вполне приемлема. Конечно, как и всякую программу, ее можно было бы подсократить на пару шагов. Вместо того чтобы сравнивать полученное значение $x = \text{Р2}$ с пределом интегрирования $b = \text{РВ}$, можно было бы просто отсчитать n циклов с помощью счетчика L0, поскольку число n у тебя записано в Р0. Можно сделать программу еще короче, если не выносить вычисление $f(x)$ в подпрограмму, а непосредственно записать его, начиная с шага 17 и сдвинув соответственно ту часть программы, которая у тебя идет с шага 19. Это позволит сократить общую программу еще на три-четыре шага. Впрочем, не будем сейчас этим заниматься. Дело в том, что вычислять интегралы на ПМК по формуле прямоугольников — занятие малоэффективное. Для получения практически приемлемой точности приходится обычно брать очень большое n , а поэтому считать приходится очень долго. Между прочим учти, что когда считаешь по таким длительным программам, лучше включать ПМК в сеть, а то аккумуляторы могут разрядиться до окончания вычисления и все твои труды пропадут впустую.

Попробуем все-таки вычислить по твоей программе хотя бы такой простенький интеграл:

$$I = \int_0^1 \sqrt{1-x^2} dx. \quad (27.3)$$

— Это очень просто, — ответил я. Подпрограмма будет такая: 1 ИП2 x^2 — $\sqrt{-}$ В/О

— Правильно, — сказал папа. — Только не забудь про капризы твоего ПМК. После операции $\sqrt{}$ (а также некоторых других операций) он не хочет возвращаться из подпрограммы в нужное место. Поэтому вставь перед В/О еще команду \uparrow или НОП.

Я ввел программу и подпрограмму и начал считать, взвя $n = 10$. Примерно через 1,5 мин я получил ответ $I = 0,78810286$, который с гордостью показал папе.

— Как ты думаешь, сколько здесь верных десятичных знаков? — спросил папа.

— Не знаю. Но это можно проверить. Я еще раз вычислю этот интеграл, но теперь возьму $n = 20$ и посмотрю, на сколько новый результат будет отличаться от старого.

Сказано — сделано. Я снова запустил ПМК при $n = 20$. На этот раз он считал 3 мин и насчитал $I = 0,78635765$. Я был очень разочарован. Здесь только первые два десятичных знака совпали с полученными раньше. Значит, за третий знак никак поручиться нельзя. Мне это не понравилось, и я со злости пересчитал еще при $n = 30$ и при $n = 50$, получив соответственно $I_{30} = 0,78592112$ и $I_{50} = 0,78564144$. При $n = 50$ ПМК считал целых 7,5 мин. Зато теперь я мог уверенно определить значения интеграла с точностью до третьего знака $I = 0,785 \dots$

— А как ты думаешь, какой должен быть истинный результат?

— Откуда мне знать. Я ведь не знаю формул, по которым ты интегрируешь такие функции, да и таблиц у меня нет.

— А голова у тебя есть? Этого, пожалуй, достаточно. Построй график твоей подынтегральной функции и подумай.

Я взял у папы клошок миллиметровки и нарисовал график (рис. 27.3).

— Да ведь это четверть окружности радиуса 1! — закричал я. — Значит, наш ин-

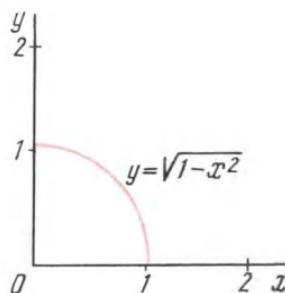


Рис. 27.3

теграл — это просто площадь четверти круга единичного радиуса, т. е. $\pi/4$.

Я быстро разделил на микрокалькуляторе π на 4 и нашел $I = 0,78539815$. Значит, первые три десятичных знака интеграла я определил верно. Но чтобы улучшить точность еще раз в 10, нужно взять очень большое n и очень долго ждать результата. Папа сказал мне, не вдаваясь в подробности, что в данном случае точность до четвертого знака получится только при $n \geq 200$.

— Для грубой прикидки значения интеграла, когда его нужно знать хотя бы с точностью до 1 %, можно пользоваться и формулой прямоугольников. Но обычно пользуются другими формулами, позволяющими ускорить эти вычисления. Это так называемые формула трапеций и формула парабол, которую называют также формулой Симпсона. Они также основаны на делении отрезка на n полосок, но учитываются значения $f(x)$ не только в середине, но и на краях полоски. Поэтому с увеличением n точность вычисления по этим формулам растет значительно быстрее, чем по формуле прямоугольников. На формуле трапеций мы останавливаться не будем. Скажу только, что мы ее уже использовали, когда моделировали парусную ре-гату.

— Как? Разве там мы что-то интегрировали?

— Конечно. Ведь по положению руля и паруса мы вычисляли скорость, а затем по составляющим скорости вычисляли составляющие пройденного пути. А это ведь и есть интегрирование. Но мы сейчас перейдем к наиболее эффективной из простых формул численного интегрирования — формуле Симпсона. Здесь интервал интегрирования от a до b делится также на n полосок шириной h , но значения функции вычисляются не только в середине каждой полоски, но и на границах полосок.

— Так это же будет еще дольше, чем по формуле прямоугольников. Там ведь функция вычисляется только в середине полосок.

— Погоди немного. Зато здесь для получения хорошей точности нужно меньшее n , а поэтому вычисление происходит быстрее. Ты скоро в этом убедишься. Так вот, формула такая:

$$\int_a^b f(x) dx \approx \frac{h}{6} \left[f(a) + 4f\left(a + \frac{h}{2}\right) + 2f(a+h) + \dots \right]$$

$$+4f\left(a+\frac{3h}{2}\right)+2f(a+2h)+\dots+2f(b-h)+ \\ +4f\left(b-\frac{h}{2}\right)+f(b)\Big]. \quad (27.4)$$

Разберись в ней и составь программу.

Я долго рассматривал эту формулу и понял, что в сумме в квадратных скобках все значения $f(x)$, взятые в середине каждой полоски, входят с множителем 4, а на границах полосок — с множителем 2. Точек в середине полосок n , а на границах между полосками $n - 1$. Кроме того, туда входят значения $f(x)$ на краях интервала с множителем 1. Это вовсе не так сложно. Вот что получилось.

Программа 56. Численное интегрирование по формуле Симпсона

```

П0 П9 С/П ПА С/П ПВ ПП 62 ПС ИПВ
ИПА — ИП0 ÷ П2 2 ÷ П1 ИПА ПП
62 ИПС + ПС 0 ПД ИПА ИП1 ПП 44
ИП9 П0 КИП0 ИПА ИП2 ПП 44 ИПС + ИП1
× 3 ÷ С/П + П3 ПП 62 ИПД +
ПД ИП3 ИП2 + П3 Л0 46 ИПД 2 ×
ПД В/О (подпрограмма для  $f(x)$ ) В/О

```

Инструкция. $n = RX$ В/О С/П, $a = RX$ С/П, $b = RX$ С/П $RX = I$. Подпрограмма вычисления $f(x)$ вводится с шага 62, причем x находится в RX и туда же записывается вычисленное значение $f(x)$.

Обратите внимание на использование РД. Для проверки этой программы я ввел подпрограмму для подынтегральной функции (27.3), поскольку верное значение ее интеграла я знаю: $x^2 \int_1^X Y - \sqrt{HOP}$ В/О. Вводя различные n , я получил следующие результаты:

n	Время вычисления	Результат
10	2 мин 50 с	0,78411173
20	5 20	0,78494386
30	8 30	0,78515096
50	13 50	0,7852834

Итак, при одинаковых значениях n вычисление по программе 56 занимает примерно на 75 % больше времени, чем по программе 55, но зато погрешность оказывается примерно в 2,5 раза меньше. Если сравнивать точность вычисления не при одинаковых n , а при одинаковой затрате времени, то небольшое преимущество все же остается за формулой Симпсона, ее погрешность примерно в 1,5 ... 2 раза меньше, чем по формуле прямоугольников.

Эти выводы я показал папе. Он с ними в основном согласился, только заметил, что с увеличением n погрешность вычисления по Симпсону убывает быстрее, чем по формуле прямоугольников. Так, если увеличить n (т. е. уменьшить h) в α раз, то погрешность вычислений по формуле прямоугольников уменьшится примерно в α^2 раз, а по формуле Симпсона — в α^5 раз. Поэтому если нужно знать значение интеграла с большой точностью, то, потратив значительно больше времени, чем в моих упражнениях, например около часу или больше, я добьюсь значительно лучших результатов, вычисляя по формуле Симпсона, а не по формуле прямоугольников.

— Теперь попробуй вычислить по программе 56 интеграл, который мне сейчас нужен. Это так называемый интеграл Френеля, точнее один из интегралов Френеля:

$$S(z) = \int_0^z \sin(\pi x^2/2) dx \text{ при } z=0,5. \quad (27.5)$$

Мне желательно знать этот интеграл с точностью по крайней мере до 7 знаков. К сожалению, в лучших таблицах, которые у меня есть, приведено только 6 знаков.

Программа у меня уже была набрана, так что осталось только ввести подпрограмму для $f(x) = \sin(\pi x^2/2)$. Это оказалось совсем просто: $x^2/2 \div \pi \times \sin \text{ НОП В/О}$, и заняло всего 8 шагов из имеющихся в моем распоряжении 36.

— Я возьму $n = 200$. Тогда наверняка буду уверен, что все вычисленные знаки верные.

— Ты глубоко заблуждаешься, — сказал папа. — Увеличение n не всегда приводит к повышению точности. Конечно, ошибки, связанные с тем, что мы плохо учитывали изменение функции в пределах полоски (методические погрешности) с увеличением n уменьшаются. Но есть еще погрешности самого вычисления функции на микрокалькуляторе и округления частных результатов. Эти ошибки,

разумеется, тем больше, чем больше слагаемых мы суммируем; каждое из них вносит свою долю в эту погрешность. Поэтому чрезмерное увеличение n может ухудшить точность. Лучше всего взять умеренное значение n , а потом повторить вычисление, взяв n вдвое больше. Те знаки, которые будут совпадать в обоих результатах, можно полагать скорее всего верными. Поэтому начни, скажем, с $n = 20$.

Я так и сделал. При $n = 20$ я получил результат $S(0,5) = 0,064732426$ (продолжительность вычисления — около 7 мин), а при $n = 40$ $S(0,5) = 0,064732430$ (продолжительность вычисления — 13,5 мин). Я еще раз удвоил n и получил за 25 мин $S(0,5) = 0,064732433$. Таким образом, можно считать, что с точностью до 7 значащих цифр $S(0,5) = 6,473243 \cdot 10^{-2}$.

— Если ты хочешь получить достаточно точный результат, затратив не очень много времени, выбирать число полосок n (или ширину полоски h) надо с умом. Впрочем, в книге [3] есть программа вычисления интегралов по формуле Симпсона с заданной точностью. При этом ПМК постепенно увеличивает n , пока не будет обеспечена заданная точность. Нужно только иметь в виду, что в этой программе учитывается лишь методическая погрешность, а не погрешности вычисления и округления. Поэтому если задавать очень большую точность, например до 10^{-6} , то она может ввести в заблуждение. Несколько интересных программ численного интегрирования приведены и в журнале «Наука и жизнь» (№ 5 за 1986 г.).

— Кстати, папа, по какой программе ты считал свои интегралы?

— Мне пришлось считать их по другим программам. У меня интегралы с бесконечным верхним пределом. Математики называют такой интеграл несобственным.

— А это что такое?

— Ты много хочешь знать сразу. Ладно, попробую объяснить. Пусть верхний предел b интеграла $\int_a^b f(x) dx$ —

переменное число. Тогда наш интеграл будет функцией от b , которую обозначим $I(b)$. Теперь будем неограниченно увеличивать b . Может случиться, что $I(b)$ будет стремиться к некоторому конечному пределу. Этот предел и

является значением интеграла $\int_a^\infty f(x) dx$. При этом гово-

рят, что наш несобственный интеграл сходится. Вот пример:

$$I = \int_0^{\infty} e^{-x^2} dx \quad (27.6)$$

сходится. Это значит, что площадь фигуры, которую я нарисовал (рис. 27.4), является конечной, хотя справа она ничем не ограничена: наша кривая очень быстро приближается к оси абсцисс и практически при больших x сливается с ней.

— Но ведь могут быть функции, для которых такого предела нет.

— Конечно. Простой пример $f(x) = 1$. Тогда $I(b) = \int_0^b 1 dx = b$ и с неограниченным возрастанием b тоже неограниченно возрастает. В этих случаях говорят, что интеграл расходится. Но меня сейчас интересуют сходящиеся интегралы, для них я и составил программу.

Так вот, моя программа считает по формуле Симпсона до тех пор, пока накапливаемая сумма перестанет изменяться. Тебе будет нетрудно понять идеи этой программы. Замечу только, что на шаге 34 при выходе из подпрограммы мы имеем в РС и РХ новое значение текущей суммы, старое ее значение сохранилось в «архивном регистре РХ1. Поэтому мы можем на шаге 20 вызвать это старое значение и сравнить его с новым. Когда их разность становится машинным нулем, вычисление заканчивается.

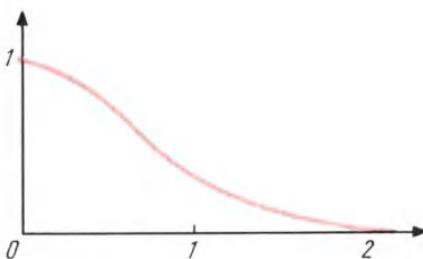


Рис. 27.4

Программа 57. Численное интегрирование $I = \int_a^{\infty} f(x) dx$ по формуле Симпсона

```

П0 0      ПС С/П П1    ИП0 ПП 39 ПП 31
ПП 35      4      ПП 30   ПП 35  2   ПП 30
Вх — x=0 10 ИПС ИП1 × 3 ÷ С/П
× ИПС + ПС В/О ИП0 ИП1 + П0 ...
... (программа для  $f(x)$ ) ... В/О

```

Инструкция. $a = RX$ В/О С/П, $h/2 = RX$ С/П
 $RX = I$. Программу для $f(x)$ ввести начиная с шага 39.

Я довольно легко разобрался в этой программе. Разные обращения к подпрограммам позволили сэкономить много шагов.

— Попробуй вычислить по этой программе интеграл (27.6). Не будем пока гоняться за большой точностью и возьмем не очень маленькое h , например $h/2 = 0,2$. Кстати, запишем это в качестве тест-примера.

Я быстро составил программу для $f(x)$ в виде: $x^2 e^{-x}$ 1/x В/О (конечно, не $x^2 / -/ e^x$ В/О, так как помню о капризах ПМК). При $a = 0$ и $h/2 = 0,2$ получил результат $I = 0,88622693 \dots$ через 3 мин.

— Теперь проверим, насколько твой ответ. Дело в том, что точное значение твоего интеграла известно из теории, а именно: $I = \sqrt{\pi}/2$.

Вычислив это на ПМК, я получил 0,8862269, что до последнего знака совпадает с моим результатом

Я подсчитал еще некоторые интегралы, которые мне посоветовал папа, в частности

$$\int_0^{\infty} e^{-x} dx = 1; \int_1^{\infty} e^{-x} dx = 1/e \approx 0,367879;$$

$$Q(A) = \frac{1}{\sqrt{2\pi}} \int_A^{\infty} e^{-x^2/2} dx =$$

$$= \begin{cases} 2,867906 \cdot 10^{-7} & \text{при } A = 5, \\ 7,6199 \cdot 10^{-24} & \text{при } A = 10. \end{cases}$$

Вообще говоря, все эти интегралы вычисляются на ПМК медленно, по несколько минут каждый. Если для ускорения увеличить h , то сильно страдает точность.

Чтобы не так скучно было ждать, пока ПМК досчитает до машинного нуля, я немного переделал программу 57 по примеру программы 40 для решения уравнений, взяв на себя часть функций ПМК. В данном случае в программе 57 я изменил шаги 20—22: вместо $Bx - x = 0$ я вставил С/П НОП БП и инструкцию изменил так: $a = RX$ С/П, $h/2 = RX$ С/П $RX = C_1$ С/П, $RX = C_2$ С/П, ..., пока значение C_n не будет совпадать в нужном числе знаков с C_{n-1} . После этого $\overrightarrow{ШГ} \overrightarrow{ШГ} С/П RX = 1$. Здесь «пустую» операцию НОП я ввел только для того, чтобы не изменять адреса остальных операций. При этом вычисления не стали быстрее, но считать стало интереснее: видно, как мы приближаемся к пределу. Впрочем, папа в данном случае предпочитает свой вариант программы, а пока ПМК вычисляет интеграл, занимается другой работой.

28

ЕЩЕ РАЗ О СУММИРОВАНИИ

Сегодня выходной день, но папа с утра сидит за своим микрокалькулятором и что-то упорно считает. Я знаю, что это бывает с папой тогда, когда к нему в голову приходит блестящая идея и он хочет как можно скорее в ней разобраться. Иногда после этого он получает авторское свидетельство и начинает его упорно «внедрять». Что это значит, я толком не знаю, но знаю, что при этом папа ходит сумрачный и к нему не подступиться. Но чаще папины блестящие идеи кончаются благополучно — уже на следующий день папа смеется: «Надо же было придумать такой БСК». БСК — значит «бред сивой кобылы».

Сегодня я долго не решался подойти к папе, потом подумал, что, может быть, я ему смогу чем-то помочь. К моему удивлению, папа согласился.

— Ты сегодня опять интегрируешь? — спросил я.

— Нет, сегодня я считаю разные суммы. Кстати, тебе будет очень полезно научиться программировать их вычисление.

— Чему же тут учиться? — удивился я. — Слава боту, складывать я умею и на калькуляторе, и даже в уме.

— Речь идет о функциональных суммах и рядах, когда нужно вычислить каждое слагаемое, а уж потом прибавлять его к остальным. Кстати, сейчас у меня очень простая задача, и ты с ней легко справишься. Скажи-ка, тебе никогда не попадалась такая формула:

$$q^n + C_n^1 q^{n-1} p + C_n^2 q^{n-2} p^2 + C_n^3 q^{n-3} p^3 + \dots + \\ + C_n^{n-1} q p^{n-1} + p^n = \sum_{i=0}^n C_n^i q^{n-i} p^i ? \quad (28.1)$$

Я не мог припомнить, где ее встречал.

— Это ведь формула бинома Ньютона. Когда-то ее в школе учили, а теперь облегчили вам жизнь. Бином — значит двучлен, в данном случае $q + p$. Если возвести его в степень n , то $(q + p)^n$ можно разложить в сумму (28.1). Поэтому величины C_n^i называются биномиальными коэффициентами.

Тебе, конечно, ясно, что если мы хотим найти всю сумму (28.1) при заданных p , q и n , то проще всего ее вычислить как $(q + p)^n$. Но очень часто требуется знать только часть суммы, например

$$P = \sum_{i=0}^k C_n^i q^{n-i} p^i, \quad k < n, \quad (28.2)$$

причем чаще всего $q = 1 - p$. При этом условии выражение (28.2) представляет собой вероятность того, что при n независимых испытаниях событие, имеющее вероятность p , произойдет не больше чем k раз. Впрочем, ты ведь о вероятностях, к сожалению, ничего не знаешь.

— Кое-что знаю, читал.

— Это очень хорошо, — сказал папа, — но сейчас совершенно не важно. Наша цель — составить программу для вычисления суммы (28.2). Интересно, как бы ты это сделал?

— Очень просто. У меня ведь есть коротенькая программа 19 для вычисления биномиальных коэффициентов. Я ее использую в качестве подпрограммы и буду вычислять один член суммы за другим и складывать их.

— К сожалению, это не так уж просто. Такой алгоритм будет работать очень медленно. И к тому же при та-

ком большом количестве операций у тебя будут накапливаться ошибки. Я тебе сейчас покажу общий принцип построения алгоритма для почти всех задач вычисления функциональных сумм. Этот принцип основан на рекуррентном вычислении каждого члена по предыдущему. Проще всего это можно пояснить так. Перепишем (28.2):

$$P = \sum_{i=0}^k A_i, \quad (28.3)$$

где A_i — i -й член суммы. Я мог бы в явном виде выписать член A_i , да фактически это сделано в (28.2). Но подойдем к этому иначе. Запишем нулевой член (при $i = 0$), учитывая, что $C_n^0 = 1$ и $p^0 = 1$:

$$A_0 = q^n. \quad (28.4)$$

Далее рассмотрим значение A_i и сравним его с A_{i-1} , учитывая выражения для биномиальных коэффициентов (11.4):

$$A_i = \frac{n(n-1)(n-2)\dots(n-i+1)}{i(i-1)(i-2)\dots 1} q^{n-i} p^i,$$

$$A_{i-1} = \frac{n(n-1)(n-2)\dots(n-i+2)}{(i-1)(i-2)\dots 1} q^{n-i+1} p^{i-1},$$

а это позволяет записать рекуррентную формулу

$$A_i = A_{i-1} \frac{n-i+1}{i} \frac{p}{q}. \quad (28.5)$$

Теперь алгоритм тебе ясен. Сначала вычисляем A_0 , а затем последовательно все A_i по (28.5) и тут же складываем их. В программе значения A_i будем записывать в РА, а текущие суммы — в РС. Индекс i запишем в Р4 и будем его увеличивать каждый раз на единицу командой КИП4. Здесь, пожалуй, удобно хранить $n - i + 1$ в Р3, команда КИП3 позволит уменьшить его содержимое за один шаг, а не за три, как было бы при прямом вычислении. Кроме того, используем счетчик на регистре Р0 для того, чтобы закончить вычисление при $i = k$. Вот и получится такая программа.

Программа 58. Вычисление биномиальной суммы
(28.2)

```
↑ ↑ 1 П4 XY С/П П7 ÷ П6 ИП8
П0 ИП9 П3 ИП7 XY ПА ПС ИПА ИП3 ×
ИП4 ÷ ИП6 × ПА ИПС + ПС КИП3 КИП4
Л0 17 ИПС С/П
```

Инструкция. ($n = P9$, $k = P8$) $p = RX$ В/О С/П, $q = RX$ С/П $RX = P$. Если $q = 1 - p$, то вместо операции ввода ($q = RX$) можно просто нажать клавишу — и далее продолжать С/П.

Примеры.

1. $n = 5$, $k = 2$, $p = 3$, $q = 4$. Результат $P = 10624,002$ (истинное значение $P = 10624$). Время вычисления — 20 с.

2. $n = 23$, $k = 3$, $p = 0,01$, $q = 1 - p$. Результат $P = 0,99992264$. Более точное значение $P = 0,99992395$. Продолжительность — около 25 с. Она приблизительно пропорциональна k .

— Ну как, — спросил папа, — идея программы тебе понятна?

— Конечно. Я вижу, что величину p ты в память не записываешь, а запоминаешь только q , которое нужно для вычисления A_0 , и отношение p/q . Но вот что непонятно. Ты сначала ввел n в Р9 и k в Р8, а затем переписал их в Р3 и Р0 и больше к Р9 и Р8 не возвращался. Можно было сэкономить шагов 6, если сразу записать n в Р3 и k — в Р0.

— Это было бы так, если бы тебе нужно было только один раз вычислить сумму (28.2). Но обычно приходится рассчитывать разные варианты при тех же n и k , но при других p и q . Если принять твое предложение, то придется заново вводить n и k перед каждым новым вычислением. А так они сохраняются в памяти и вводить приходится только p и q .

— Понятно. У меня еще один вопрос. Ты ведь предупреждал, что операцию X^Y лучше не применять, а сам ее использовал.

— Это верно. Но от этого принципа иногда можно и отступить. В данном случае нет условий для возникновения «капризов» ПМК при этой операции. Конечно, ПМК вычисляет X^Y с заметной погрешностью. Это видно и по ре-

зультатам, в которых шестой или восьмой знаки оказались неверными. Иногда могут возникнуть ошибки даже в пятом знаке. Но обычно такие погрешности на практике допустимы. Если же нужен более точный результат, то придется немного удлинить программу, заменив операцию X^Y вычисления степени циклическим умножением, как я тебе уже однажды показывал (см. стр. 64). В данном случае вместо шагов 14 и 15 нужно применить такой фрагмент: П2 1 ИП7 \times Л2 16, соответственно скорректировать адрес перехода в конце программы (22 вместо 18). Результат будет более точным, но продолжительность вычислений возрастет пропорционально n . Какие у тебя еще претензии к программе 58?

— Больше, пожалуй, никаких.

— Тогда я тебе укажу на один подводный камень; о нем надо помнить при пользовании этой программой. Если величина q очень мала ($q \ll 1$), а n велика (а с этим встречаешься довольно часто), то вычисление q^n по этой программе на шаге 15 даст машинный нуль. Беда эта сохраняется, даже если заменить команду X^Y приведенным выше фрагментом.

— Что же делать в таком случае?

— Нужно менять алгоритм. Для этого имеются различные способы. Чаще всего используют тот факт, что формула (28.2) симметрична относительно p и q . Легко показать, что имеет место равенство

$$\sum_{i=0}^k C_n^i p^i q^{n-i} = (p+q)^n - \sum_{i=0}^{n-k-1} C_n^i q^i p^{n-i}. \quad (28.6)$$

Поэтому можно в инструкции к программе 58 заменить p на q , а k на $n - k - 1$, тогда программа вычислит дополнение к P до $(p+q)^n$ (т. е. при $p = 1 - q$ до единицы).

Вот другая задача такого же рода. Недавно мне пришлось рассчитывать одну систему связи и для этого нужно было вычислять выражение

$$P = \sum_{k=0}^{n-1} C_{n+k-1}^k p^n (1-p)^k \quad (28.7)$$

при различных положительных значениях p примерно от 10^{-5} до 0,5 и $n = 2, 3, 4$ и 5. Кстати, догадайся, почему мне не пришлось считать по этой формуле для $n = 1$?

— Откуда мне знать? Я ведь даже не знаю, что такое n .

— Да этого сейчас и не нужно знать. Ты посмотри на формулу. Что получится, если $n = 1$?

Я посмотрел и подумал. При $n = 1$ есть только одно значение $k = 0$. А $C_0^0 = \frac{0!}{0! 0!} = 1$. Таким образом, при $n = 1$, $P = p$, так что и считать здесь нечего.

— Теперь составим программу. Запишем $P = \sum_{k=0}^{n-1} A_k$,

где

$$A_0 = p^n, \quad (28.8)$$

а при $k > 0$

$$A_k = A_{k-1} \frac{n+k-1}{k} (1-p). \quad (28.9)$$

Значит, здесь тоже очень удобно вычислять все A_k по рекуррентной формуле (28.9). Будем записывать A в РА, текущие суммы в РС, $1 - p$ в Р6, k в Р4 и $n + k - 1$ в Р5. Это позволит легко переходить от k к $k+1$ с помощью команд КИП4 и КИП5. Счетчик на Р0 используем для отсчета $n - 1$ сложений. Попробуй записать такую программу.

Я, конечно, не сразу записал программу. Но, воспользовавшись как образцом программой 58, довольно быстро ее осилил.

Программа 59. Вычисление P по формуле (28.7) ($n > 1$)

П7	1	П4	ХУ	—	П6	ИП9	П0	П2	П5
1	ИП7	×	L2	11	ПА	ПС	КИП0	ИПА	ИП5
×	ИП4	÷	ИП6	×	ПА	ИПС	+	ПС	КИП4
КИП5	L0	18	ИПС	С/П					

Инструкция. $n = \text{Р9}$, $p = \text{РХ}$ В/О С/П РХ =
= Р.

Примеры. При $n = 2$, $p = 0,001$ результат $P = 2,998 \cdot 10^{-6}$, при $n = 4$, $p = 0,01$ результат $P = 3,4167 \cdot 10^{-7}$, при $n = 4$, $p = 0,1$ результат $P = 2,728 \times 10^{-3}$. Время вычисления 30 ... 50 с.

— Хорошо, — сказал папа, — с этим ты справился. А теперь попробуй записать алгоритм для вычисления такой функциональной суммы (28.3).

— Зачем же записывать алгоритм, если у нас уже есть программа?

— А затем, что это будет универсальный алгоритм, который позволит вычислять почти любую функциональную сумму. Различие между задачами здесь будет только в том, как вычисляется A_0 и по какой рекуррентной формуле следует переходить от A_{k-1} к A_k .

Итак, предположим, что для этого мы составили два вспомогательных алгоритма, которые назовем А НУЛЬ и РЕКУРРЕНТА. В большинстве случаев это очень простые алгоритмы, как ты мог убедиться по формулам (28.4) и (28.5) для биномиальной суммы или (28.8) и (28.9) для суммы (28.7). Теперь составим общий алгоритм для решения задачи

$$P = \sum_{k=0}^n A_k. \quad (28.10)$$

алг СУММА (вещ P , a, b, \dots, x , цел n, m, \dots)

арг $x, n, a, b, \dots, m, l, \dots$

рез P

нач цел k вещ A_0, A_k, C

А НУЛЬ (... A_0)

$C := A_0; k := 1$

нц пока $k \leq n$

РЕКУРРЕНТА (A_{k-1}, A_k)

$C := C + A_k;$

$k := k + 1$

кц

$P := C$

кон

Вот это и есть универсальный алгоритм для функциональных сумм. Здесь a, b, m, l — некоторые параметры, от которых могут зависеть A_k . Иногда их может вовсе не быть. Возможны, конечно, и некоторые вариации. Так, в задаче (28.7) индекс последнего члена обозначен не n , а $n - 1$. Значит, нужно ввести поправку в алгоритм и писать «пока $k \leq n - 1$ ». Но с этим ты легко разберешься

в каждом конкретном случае. А в общем, для любой такой задачи достаточно составить алгоритмы (или формулы) для A_0 и A_k и смело подставлять их в наш универсальный алгоритм.

Сейчас ограничимся этим. Завтра я расскажу тебе о рядах и там мы опять столкнемся с функциональными суммами и рекуррентными формулами. А сейчас мне нужно решить свои проблемы.

29 РЯДЫ, РЯДЫ...

— Что это такое ряды, о которых ты мне хотел сегодня рассказать, — спросил я у папы, когда он вернулся с работы.

— Это те же суммы, но с бесконечным числом слагаемых. Вот, когда мы с тобой находили предел суммы геометрической прогрессии $1 + q + q^2 + q^3 + \dots$, мы фактически суммировали ряд. Если эта сумма стремится к конечному пределу, мы говорим, что ряд сходится, если же предел не существует или бесконечен, то ряд расходится. Сегодня я хотел поговорить с тобой о функциональных рядах, в которых каждый член является определенной функцией некоторой переменной величины. С помощью таких функциональных рядов вычисляются многие так называемые специальные функции, которые в математике и технике иногда играют не меньшую роль, чем синусы и косинусы.

Мы с тобой вычисляли позавчера интеграл

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-z^2/2} dz \quad (29.1)$$

и потратили на это довольно много сил. На практике обычно функцию $Q(x)$ вычисляют не прямым интегрированием, а с помощью ряда

$$Q(x) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{n! 2^n (2n+1)}. \quad (29.2)$$

Вот для этого ряда и составь программу.

Я сразу окунулся с головой в работу. Представив входящий в эту формулу ряд в виде

$$\sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{n! 2^n (2n+1)} = \sum_{n=0}^{\infty} A_n,$$

я быстро определил

$$A_0 = x; \quad A_n = -A_{n-1} \frac{x^2}{2n} \frac{2n-1}{2n+1}. \quad (29.3)$$

Остальное было делом техники. Программа получилась такой.

Программа 60. Вычисление функции $Q(x)$ (29.2)

ПА ПС x^2 П2 1 П4 ИПА /-/ ИП2 ×
ИП4 × КИП4 → ИП4 ÷ КИП4 → ИП4 ÷
ПА ИПС + ПС Вх — $x=0$ 06 ИПС 2
л × √ ÷ 2 1/x XY — С/П

Инструкция x РХ В/О С/П РХ — $Q(x)$.

В этой программе я использовал регистр Р4 для значений $2n - 1$, $2n$ и $2n + 1$. Вначале я ввел 1 в Р4 и на шаге 10, пока $n = 1$, использовал Р4 как $2n - 1$. Затем командой КИП4 я увеличил Р4 до 2 и использовал как $2n$ (шаг 14). Применив эту команду еще раз, увеличил содержимое Р4 еще на 1 и использовал как $2n + 1$ (шаг 18). При втором прохождении цикла, охватывающего шаги 06—27, $n = 2$ и в Р4 содержится $2n - 1 = 3$, которое затем преобразуется в $2n - 4$ и в $2n + 1 = 5$, и так при каждом повторении цикла. Для выхода из цикла служат шаги 24—27, с помощью которых я сравниваю текущее значение суммы с предыдущим и прекращаю дальнейшее повторение, если разность этих сумм стала машинным нулем.

Примеры. $Q(0.4) = 0.34457826$ (0.344457826) —
около 1 мин,
 $Q(2) = 0.02275009$ (0.02275005) —
2,3 мин,
 $Q(3) = 1,34987 \cdot 10^{-3}$ ($1,349899 \cdot 10^{-3}$) —
3,5 мин,

$$Q(4) = 3,001 \cdot 10^{-5} (3,1671 \cdot 10^{-5}) — \text{около}$$

5 мин.

В скобках приведены значения, которые папа выписал из таблиц.

— Первый результат совпадает с табличным, — сказал он. $Q(2)$ ты вычислил с шестью верными цифрами, $Q(3)$ — с пятью, а $Q(4)$ — совсем плохо, здесь относительная погрешность больше 5 %.

— А почему это так? — спросил я.

Очень просто. В отличие от предыдущих задач, у тебя здесь сумма знакопеременная. При четном k члены A_k положительны, а при нечетном k отрицательны. Если они по абсолютному значению близки друг к другу, то ты имеешь уже знакомую тебе неприятность — вычитание двух близких друг к другу величин. А при этом, как ты знаешь, возникают большие относительные погрешности. А так как такое вычитание при больших x происходит в данном случае много раз, то неудивительно, что $Q(x)$ вычисляется с огромной погрешностью. Поэтому к вычислению знакопеременных сумм необходимо подходить с большой осторожностью.

— Ясно. Ряд знакопеременный, и с увеличением аргумента увеличивается накопление ошибок округления. Как с этим бороться?

— К счастью, в данном случае существует другой ряд, который хорошо работает как раз при больших x . Этот ряд не совсем такой, как предыдущие. Прежде всего, это ряд расходящийся. Да, представь себе, что и по расходящемуся ряду можно кое-что подсчитать. Дело в том, что этот ряд асимптотический. Я не буду тебе объяснять, что это такое. Скажу только, что если оборвать этот ряд в том месте, где его члены начинают увеличиваться, то он дает приближенное значение $Q(x)$ тем точнее, чем больше x . Абсолютная погрешность меньше первого отброшенного члена. Вот этот асимптотический ряд для $Q(x)$:

$$Q(x) \approx \frac{e^{-x^2/2}}{x \sqrt{2\pi}} \times$$

$$\times \left[1 + \sum_{n=1}^N (-1)^n \frac{1 \cdot 3 \cdot 5 \dots (2n-1)}{x^{2n}} \right]. \quad (29.4)$$

Выражение в квадратных скобках запишем так:

$$\sum_{n=0}^N A_n, \text{ где } A_0 = 1, A_n = -A_{n-1}(2n-1)/x^2. \quad (29.5)$$

В справедливости этих формул ты легко убедишься. А теперь составь программу. Только не забудь, что суммирование нужно закончить, как только значения $|A_n|$ перестанут уменьшаться. Взгляни на формулу (29.5) и ты убедишься, что это произойдет, когда $2n - 1$ достигнет значения x^2 .

Я разобрался во всем этом и быстро составил программу.

Программа 61. Вычисление $Q(x)$ (29.4) при $x > 3,5$

x² П2 1 ПА ПС П4 ИПА /-/ ИП4 ×
ИП2 ÷ ПА ИПС + ПС КИП4 КИП4 ИП4 ИП2
— x ≥ 0 06 ИПС ИП2 2 × π × ИП2
e^x × √ ÷ С/П

И н с т р у к ц и я. Такая же, как в программе 60.

При составлении этой программы я преобразовал множитель перед квадратными скобками в (29.4):

$$e^{-x^2/2}/(x\sqrt{2\pi}) = 1/\sqrt{x^2 2\pi e^{x^2}}. \quad (29.6)$$

Это позволило сэкономить несколько шагов и не хранить в памяти значение x в первой степени.

Примеры. $Q(3) \approx 1,3610997 \cdot 10^{-3}$ (меньше 1 мин),
 $Q(4) \approx 3,1679147 \cdot 10^{-5}$ (около 1 мин),
 $Q(5) \approx 2,8665234 \cdot 10^{-7}$ (около 2 мин).

Папа проверил эти результаты по таблицам. Оказалось, что $Q(3)$ вычислено неудовлетворительно — здесь только две значащие цифры верны, зато $Q(4)$ вычислено с хорошей точностью, относительная погрешность порядка 0,01 %, а значение $Q(5)$ совпадает с полученным из таблиц.

— Вот видишь, асимптотический ряд считает тем точнее, чем больше x . Попробуй на досуге составить универсальную программу для вычисления $Q(x)$, объединив программы 60 и 61 так, чтобы по величине x сам ПМК выбирал, какой из двух рядов следует использовать. Ну, на сегодня хватит.

ВОТ И МЕСЯЦ ПРОШЕЛ

Сегодня папа пришел в приподнятом настроении и, когда мы все сели ужинать, он обратился к маме.

— Заметь, сегодня ровно месяц, как Миша занялся программированием. Можешь убедиться, что он уже вполне грамотно и самостоятельно составляет программы, если суть задачи ему понятна. А если суть непонятна, то он все равно может составить программу — была бы формула. Так что я оказался прав. За месяц можно научиться программированию на ПМК.

— Да, с Б3-34 я вполне освоился, так же, как с его братьями МК-54 и МК-56. Но говорят, что появились новые микрокалькуляторы и появятся еще другие, так что мне придется скоро переучиваться.

— По этому поводу ты особенно не волнуйся, — сказал папа. — Сейчас конструкторы стали более благородными, чем несколько лет тому назад, когда от Б3-21 перешли к выпуску Б3-34, в котором использован совсем другой язык. Теперь все новые ПМК будут базироваться на том же языке Б3-34, точнее, будут содержать все его команды, к которым добавляются новые возможности, расширяющие область применения ПМК и делающие общение с ним более удобным.

— Какие же это новые возможности?

— Ну вот, например, два новых ПМК — «Электроника МК-61» и «Электроника МК-52». Первый в таком же корпусе, как МК-54. Второй несколько больше по размерам, но зато имеет полупостоянное запоминающее устройство (ППЗУ), в котором длительное время могут храниться несколько программ (до 512 шагов в сумме), даже при отключенном питании. В любой момент любую из этих программ можно сразу переписать в программную память оперативного запоминающего устройства (ОЗУ).

Программная память в ПМК обоих типов увеличена с 98 до 105 шагов, а также добавлен один регистр памяти (РЕ). Все программы, составленные для Б3-34, пригодны для МК-52 и МК-61, кроме тех, в которых используются команда КП↑ или КИП↑. К сожалению, «черный ход» здесь не сохранился. Но самое главное, добавлено еще несколько операций: выделение целой и дробной частей

числа, нахождение абсолютного значения числа, определение максимума из двух чисел, перевод долей градуса в минуты и секунды и наоборот. Конечно, ты умеешь выполнять все эти операции и на Б3-34, но на каждую из них уходит от 2 до 8 шагов, а здесь каждая занимает один шаг. Кроме того, введены логические операции над двоичными числами. Все эти новшества существенно расширяют возможности ПМК и облегчают программирование. Разрабатываются еще несколько новых типов ПМК. Например, «Электроника МК-72» может подключаться к бытовому магнитофону и телевизору. Магнитофон может хранить библиотеку программ и автоматически вводить их в ОЗУ, а телевизор играть роль дисплея. В журнале «Наука и жизнь» № 4, за 1987 сообщается, что скоро будет выпущен карманный микрокомпьютер «Электроника МК-85» с быстродействием, в 10—15 раз большим, чем ПМК «Электроника Б3-34», позволяющий вводить длинные программы на языке Бейсик и сохраняющий их даже при выключенном питании. Таким образом, постепенно будет стираться грань между ПМК и персональным компьютером.

— А когда у меня будет персональный компьютер? — спросил я.

— Вероятно, довольно скоро. Вот закончишь учебу, приобретешь специальность, начнешь работать и будешь с каждой получки откладывать деньги на компьютер. К тому времени они, вероятно, будут не очень дорогими.

Я подумал про себя, что у меня с папой различные масштабы времени — то, что он считает «довольно скоро», в моем представлении значит в отдаленном будущем.

— Только имей в виду, — сказала мама, — что для персонального компьютера тебе придется составлять программы на Фокале или на Бейсике. Так что берись за изучение языков программирования.

Я ответил, что к тому времени, когда я накоплю деньги на компьютер, я успею выучить все языки программирования и заодно наладить свои отношения с русским языком.

Тут вдруг мама вспомнила о моих записках.

— Ты ведешь дневник о своем микрокалькуляторе?

— Конечно, — сказал я и вытащил из своего стола три толстые тетради, из которых две были исписаны полностью. Родители за них ухватились и стали читать.

— А знаешь, — сказала мама, — совсем неплохо написано.

— Да,— ответил папа, — если немного подредактировать, то может получиться полезное пособие для ребят, начинающих учиться программированию на ПМК. Давай сделаем вот что. Ты ведь знаешь, Миша, Валентину Ивановну, которая работает со мной в лаборатории?

— Конечно, знаю, — ответил я. — У нее еще есть дочка Катя, которая учится в Мореходном училище. Валентина Ивановна печатает все твои отчеты и статьи.

— Вот именно. Так давай попросим Валентину Ивановну перепечатать твои записи, а я покажу их в изда-тельстве. Чем черт не шутит, может быть, согласятся опубликовать их для твоих сверстников.

Так мои родители и сделали. И вот что получилось (см. стр. 1 и далее до самого конца).

Конец

ПРИЛОЖЕНИЕ

Перечень программ, рекомендуемых для использования

Номер программы	Назначение	Стр.
4	Вычисление дальности прямой видимости	44
5	Вычисление радиуса прямой видимости	44
12	Вычисление факториала $n!$ при целых $n \geq 0$	71
13	Вычисление $n!!$ для целых $n \geq 0$	72
14	Вычисление $\lg(n!)$ для любых целых $n \geq 0$	72
15	Вычисление $\lg(n!)$ по формуле Стирлинга при $n \gg 1$	76
16	Вычисление величины, обратной гамма-функции $1/\Gamma(x)$	82
19	Вычисление биномиальных коэффициентов C_n^m	93
23	Вычисление многочлена 4-й степени	113
24*	Универсальная программа для решения треугольников	115
25**	Вычисление многочлена 12-й степени	119
27*	Нахождение общего наибольшего делителя чисел n и m ($n > m$)	130
29*	Разложение числа $N < 10^8$ на простые множители	137
32*	Вычисление вычета a натурального числа n по модулю m	143
34*	Вечный универсальный календарь (юлианский и григорианский)	151
36	Решение квадратного уравнения с повышенной точностью	156

Продолжение приложения

Номер программы	Назначение	Стр.
38	Отделение корней уравнения $F(x)=0$ и уточнение их методом дихотомии	162
39	Решение уравнения $x=\varphi(x)$ методом простой итерации	170
41	Нахождение положительных корней уравнения $x=a\tg(2\pi x)$ методом простой итерации	177
42**	Решение алгебраических уравнений $F_n(x)=0$ степени $n \leq 11$ методом Ньютона	184
43	Решение произвольного уравнения $F(x)=0$ методом Ньютона	186
44	Решение алгебраических уравнений $F_n(x)$ при $n \leq 9$ методом Ньютона с понижением степени	188
45**	Решение системы линейных уравнений с двумя неизвестными	195
46	Решение системы линейных уравнений с тремя неизвестными	198
47	Вычисление среднего выборочного значения и стандартного отклонения по выборке	206
50*	Получение псевдослучайных чисел, равномерно распределенных на интервале $(0, 1)$	222
51*	Получение псевдослучайной последовательности цифр 0, 1	223
52*	Игра «чет-нечет»	226
54	Игра «Парусная регата»	237
56	Численное интегрирование по формуле Симпсона (в конечных пределах)	249
57	Численное интегрирование (с бесконечным верхним пределом) по формуле Симпсона	253

Продолж. приложения

Номер программы	Назначение	Стр.
58	Вычисление биномиальной суммы	257
60	Вычисление функции $Q(x)$ (дополнения к функции Лапласа) при $x < 3,5$ с помощью сходящегося ряда	262
61	Вычисление функции $Q(x)$ при $x > 3,5$ с помощью асимптотического ряда	264

П р и м е ч а н и е. Все программы пригодны для микрокалькуляторов «Электроника» Б3-34, МК-54 и МК-56, а также, кроме отмеченных двумя звездочками, для МК-52 и МК-61. Программы, отмеченные одной звездочкой, пригодны для всех названных микрокалькуляторов, однако при использовании МК-52 или МК-61 они могут быть упрощены или улучшены.

СПИСОК ЛИТЕРАТУРЫ

1. Микрокалькулятор «Электроника Б3/34». Руководство по эксплуатации. — Светлогорск, 1980. — 156 с.
2. Трохименко Я. К., Любич Ф. Д. Микрокалькулятор. Ваш ход. — М.: Радио и связь, 1985. — 224 с.
3. Цветков А. Н., Епанечников В. А. Прикладные программы для микро-ЭВМ «Электроника Б3-34», «Электроника МК-56», «Электроника МК-54». — М.: Финансы и статистика, 1984. — 175 с.
4. Трохименко Я. К. Игры с микро-ЭВМ. — Киев: Техника, 1986. — 120 с.
5. Трохименко Я. К. Программирование микрокалькуляторов «Электроника МК-61» и «Электроника МК-52». — Киев: Техника, 1987.

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ	3
1. КАК ЭТО НАЧАЛОСЬ	4
2. ЧТО МНЕ УДАЛОСЬ УЗНАТЬ СРАЗУ	6
3. Я ВЫПОЛНЯЮ ПЕРВОЕ ЗАДАНИЕ	14
4. Я СОСТАВЛЯЮ И ВВОЖУ СВОЮ ПЕРВУЮ ПРОГРАММУ	19
5. НАСЛЕДСТВО АЛЬ-ХОРЕЗМИ	25
6. КАК ЗАПИСЫВАТЬ ПРОГРАММУ, РЕДАКТИРОВАТЬ И ОТЛАЖИВАТЬ	39
7. ВИТЯЗИ НА РАСПУТЬЕ	47
8. ЧТО ТАКОЕ ХОРОШО И ЧТО ТАКОЕ ПЛОХО?	55
9. ДЛЯ ЧЕГО В ПМК СЧЕТЧИКИ?	61
10. ВАРИАЦИИ НА ТЕМУ О ФАКТОРИАЛЕ, ГАММА, НЕ ИМЕЮЩАЯ ОТНОШЕНИЯ К МУЗЫКЕ	73
11. ПРОГРАММА В ПРОГРАММЕ	84
12. ЕЩЕ О ТЕЛЕВИДЕНИИ	94
13. ТРЕУГОЛЬНАЯ ФАНТАЗИЯ	99
14. СЕКРЕТЫ КЛАВИШИ К	107
15. ЕЩЕ О КЛАВИШЕ К	111
16. ДОШЛИ ДО ПРЕДЕЛА	120
17. МСЬЕ АНТЬЕ И МАДАМ ФРАКСЬОН	128
18. В КАКОЙ ДЕНЬ БЫЛА КУЛИКОВСКАЯ БИТВА?	140
19. СМОТРИ В КОРЕНЬ!	153
20. ПО СТУПЕНЬКАМ И СПИРАЛЯМ	165
21. ПО КАСАТЕЛЬНОЙ	178
22. СО МНОГИМИ НЕИЗВЕСТНЫМИ	194
23. НЕМНОГО СТАТИСТИКИ	202
24. ПОИГРАЕМ?	208
25. УГАДАЙ-КА!	217
26. СПОРТ, СПОРТ, СПОРТ	229
27. ИНТЕГРАЛ? НЕТ НИЧЕГО ПРОЩЕ	241
28. ЕЩЕ РАЗ О СУММИРОВАНИИ	254
29. РЯДЫ, РЯДЫ...	261
30. ВОТ И МЕСЯЦ ПРОШЕЛ	265
ПРИЛОЖЕНИЕ. ПЕРЕЧЕНЬ ПРОГРАММ, РЕКОМЕНДУЕМЫХ ДЛЯ ИСПОЛЬЗОВАНИЯ	268
СПИСОК ЛИТЕРАТУРЫ	270

Библиотечная серия

Научно-популярное издание

ФИНК ЛЕВ МАТВЕЕВИЧ

ПАПА, МАМА, Я И МИКРОКАЛЬКУЛЯТОР

Заведующая редакцией Г. И. Козырева
Редактор Т. М. Толмачева

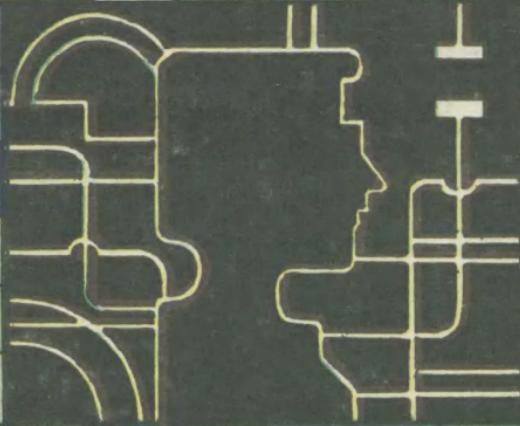
Художественный редактор Н. С. Шеин
Обложка художника Т. Л. Доброхотовой
Технический редактор И. Л. Ткаченко
Корректор Т. Л. Кускова

ИБ № 1519

Сдано в набор 20.07.87. Подписано в печать 04.04.88. Т-08683.
Формат 70×100¹/₃₂. Бумага офсетная № 2. Гарнитура литературная.
Печать офсетная. Усл. печ. л. 11,05. Усл. кр.-отт. 19,18.
Уч. изд. л. 13,91. Тираж 120.000 экз. (2-й зав. 80.001—120.000 экз.).
Изд. № 21762. Зак. № 575 Цена 60 к.

Издательство «Радио и связь», 101000. Москва, Почтамт, а/я 693

**Московская типография № 4 Союзполиграфпрома
при Государственном комитете СССР
по делам издательств, полиграфии и книжной торговли
Москва, 129041, Б. Переяславская, 46**



$+ 104 \times 108 - 12$
 $451 +$
 $\frac{2}{3} +$
 b/x
 $229y$
 ab
 $10a$



Радио и связь.