ЭКОНОМИКА
И МАТЕМАТИЧЕСКИЕ МЕТОДЫ

1965
том I, вып. 4

# ОТ ПОСТАНОВКИ ДО РЕШЕНИЯ ЗАДАЧИ НА ЭВМ

Ю. В. ГЕРОНИМУС
*(МОСКВА)*

Как должен поступить человек, по профессии, скажем, экономист, столкнувшийся в первый раз в жизни с задачей, решить которую собственными силами он не может из-за чрезвычайно большого объема вычислительной работы, которую надо выполнить для получения результатов? В наше время каждый ответит на такой вопрос без промедления: «Надо воспользоваться электронной вычислительной машиной». Но что необходимо знать об этих машинах, чтобы приступить к решению своей задачи, как сделать первый шаг на пути к машине, в чем конкретно состоит процедура, ведущая от постановки задачи к ее решению? В этой статье мы постараемся сообщить некоторые сведения, связанные с перечисленным кругом вопросов.

Человека, знакомого с вычислительной техникой лишь понаслышке, перспектива решения задачи на машине обычно пугает. Его пугает необходимость изучать двоичную систему счисления, знакомиться с программированием, обучаться искусству управления машиной за пультом. На самом деле основные трудности, связанные с работой на вычислительных машинах (если стремиться к эффективному их использованию), лежат в несколько иной области. (В частности, знание двоичной системы счисления для потребителя продукции вычислительной машины практически излишне.)

Для современного общества характерны специализация и разделение труда. Автор метода решения какой-либо задачи не занят, как правило, фактическим решением этой задачи при всевозможных вариантах исходных данных. Это делают другие исполнители. Для того чтобы сделать свой метод решения понятным исполнителям, автор должен его описать. Сводка конечного числа правил, содержащих исчерпывающие указания о том, какие действия и в каком порядке надо производить, чтобы от одного или нескольких «массивов» исходных данных прийти к «массиву» чисел-результатов, называется *алгоритмом* решения задачи. Алгоритм имеет практическую ценность только в том случае, когда он применим не к какому-нибудь одному набору исходных данных, а к обширному множеству таких наборов. Например, алгоритм поиска критического пути в задаче о сетевом планировании будет полезен только в том случае, если при его помощи можно будет находить критический путь не только в данной единственной сети, но и, скажем, во всех сетях, содержащих до $n$ событий, где $n$ — число достаточно большое, чтобы охватить все сети, возникающие в некоторой отрасли производственной деятельности.

Итак, автор алгоритма должен его описать, причем описать в таких терминах, чтобы исполнитель его, во-первых, понял, а во-вторых, имел бы технические возможности выполнить указания, содержащиеся в алгоритме.

Рассмотрим для примера задачу о вычислении значений функции

$$y = f(x) = \sin\left(\frac{\pi}{4} \cdot \frac{x^2}{1+x^2}\right)$$

для $x = x_0, x_0 + h, x_0 + 2h, \ldots, x_0 + nh$; где $x_0, h, x_1$ — произвольные числа, удовлетворяющие неравенствам $h > 0$, $x_1 - x_0 > 0$. $n = E\left(\frac{x_1 - x_0}{h}\right)$*

Если обозначить $f(x_0 + ih)$ через $y_i$ $(i = 0, 1, 2 \ldots)$, то алгоритм для решения этой задачи можно записать следующим образом **.

1. Положить $x$ равным $x_0$ (кратко: $x := x_0$).
2. Положить $i$ равным 0 $(i := 0)$.
3. Положить $y_i := \sin\left(\frac{\pi}{4} \cdot \frac{x^2}{1+x^2}\right)$.
4. Дать $x$ новое значение, полученное из старого прибавлением $h$ $(x := x + h)$.
5. Если после этого стало $x > x_1$, то перейти к п. 8, в противном случае выполнить п. 6.
6. Дать $i$ новое значение, полученное из старого прибавлением единицы $(i := i + 1)$.
7. Перейти к выполнению п. 3.
8. Остановиться.

Предписание, содержащееся в п. 3, фактически рассчитано на исполнителя: а) знакомого с термином «синус» и значением числа $\pi$, б) располагающего тригонометрическими таблицами или иными известными ему способами нахождения синусов. Несоблюдение по крайней мере одного из условий а) или б) сделает алгоритм, содержащий требование вычислить синус, невыполнимым.

Однако при помощи небольшого изменения алгоритма мы можем ориентировать его и на менее квалифицированного исполнителя, знакомого, например, лишь с четырьмя арифметическими действиями (и, конечно, с русскими словами «если», «заменить», «перейти» и т. п.).

Новый вариант алгоритма будет выражаться так: 1. $x := x_0$. 2. $i := 0$. 3. $z := \frac{3{,}1416}{4} \cdot \frac{x^2}{1+x^2}$. 4.*** $y_i := z - \frac{z^3}{6} + \frac{z^5}{120}$. 5. $x := x + h$. 6. Если после этого стало $x > x_1$, то перейти к п. 9, в противном случае выполнить п. 7. 7. $i := i + 1$. 8. Перейти к выполнению п. 3. 9. Остановиться.

Наши примеры показывают, как можно расчленить на элементарные действия, доступные не очень квалифицированному исполнителю, решение задачи, исходная формулировка и «естественные» методы решения которой выглядят, с точки зрения такого исполнителя, сложно и непонятно.

Простые и очевидные соображения, высказанные выше, приводят к важному понятию о *языке*, которым следует пользоваться при записи алгоритмов. Выбор этого языка (его терминов и конструкции) зависят, во-первых, от круга задач, решение которых надо уметь изобразить средствами языка, а во-вторых, как уже говорилось, от уровня и возможностей исполнителя.

---

* $E(A)$ означает: целая часть $A$.

** Условимся, что после исполнения пункта с номером $k$ следует исполнять пункт с номером $k + 1$, если только в пункте $k$ не содержится явного указания об изменении этого стандартного порядка.

*** В п. 4 используется разложение функции $\sin z$ в ряд Тейлора. Так как $|z| < \pi/4$, то три члена ряда дают точность порядка $2 \cdot 10^{-5}$.

Читатель, «прокрутивший» последний из приведенных выше алгоритмов, наверное, почувствовал себя на минуту своего рода механизмом, исполняющим шаг за шагом некоторые элементарные действия и решающим задачу, общего содержания которой он может и не понимать. Если такое ощущение у читателя действительно возникло, он может считать себя вплотную подошедшим к идее об использовании электронной вычислительной машины в качестве исполнителя алгоритмов.

Стандартная электронная вычислительная машина по своим возможностям очень напоминает исполнителя, которому был адресован второй из приведенных выше алгоритмов: она «умеет» выполнять арифметические действия над переменными в заданном порядке, т. е. либо в порядке написания отдельных действий, либо в соответствии со специальными указаниями, изменяющими стандартный порядок. Такой указание иногда просто содержит номер очередного подлежащего исполнению пункта, иногда дает правило для определения номера следующего пункта, основанное на анализе неравенства (как это делалось, например, в п. 6 последнего алгоритма) или на анализе других логических условий.

Алгоритм, написанный в расчете на его исполнение в электронной вычислительной машине, называется программой. Машина имеет специальное техническое устройство — «память», в которое можно ввести как самою программу, так и исходные числовые данные. В памяти помещаются промежуточные величины, возникающие при счете, а также окончательные результаты. Машина имеет устройство, позволяющее переписать в память программу и исходную информацию с внешних носителей (перфокарт или перфолент); кроме того, машина «умеет» по специальным указаниям программы печатать на бумажной ленте (или выдавать на перфокарты или перфоленту) информацию, возникающую в заданном участке памяти машины.

Те «единицы действия», которые мы выделяли как отдельные пункты в записи алгоритмов, в программе называются командами. Каждая команда выполняет обычно операцию над парой чисел, так что алгоритм, записанный на языке машины, как правило, очень «размельчен» и поэтому оказывается более длинным, чем алгоритм решения той же задачи, адресованный исполнителю-человеку, ибо человек умеет, естественно, самостоятельно разбираться в структуре сколь угодно сложных арифметических выражений и выполнять предписанные в нем действия. Каждая машинная команда (обычно она размещается в одной ячейке памяти машины) несет в себе информацию о типе операции, которую она исполняет («сложить», «умножить», «изменить порядок выполнения команды» и т. п.) и о расположении в памяти машины аргументов, над которыми операция должна быть выполнена (слагаемые, множители, номера команд и т. п.). Информация эта обычно закодирована при помощи цифр; изучение программирования частично и состоит в овладении этим языком кодов.

Хотя в конечном итоге алгоритм, предназначенный для исполнения в машине, должен быть расписан на языке команд этой машины, работу по постановке задачи на машинный счет целесообразно начать с записи на чистом русском языке *проекта* этого алгоритма (с добавлением, если это необходимо, соответствующей математической или иной технической символики). Проект — это документ, в котором автор алгоритма указывает назначение алгоритма; здесь подробно перечисляется, что́ именно должен делать алгоритм, и лишь приблизительно намечается, *как* перечисленные функции будут в алгоритме реализованы. В проекте следует описать форму вводимых в машину исходных данных и форму, в которой должны печататься результаты. При составлении проекта необходимо тщательно продумать, что́ именно следует объявить *вариантом* задачи; надо ли вво-

дить исходные данные для каждого варианта вручную с перфокарт или перфолент или лучше ввести в машину сразу всю числовую информацию, относящуюся к большой группе вариантов, а сортировку и выбор исходных данных, соответствующих отдельным вариантам, поручить делать машине автоматически (для чего придется включить в программу соответствующий командный материал). Проект описывает, таким образом, эксплуатационные особенности будущего алгоритма, важные при его конкретных применениях. Детализацию методов и схем, намеченных в проекте, можно осуществлять на более поздних стадиях подготовки задачи к счету. В проекте же иногда достаточно сослаться на название метода, который предлагается использовать в конкретной реализации (если, например, в рамках алгоритма возникает необходимость минимизировать некоторую выпуклую функцию, то в проекте можно ограничиться указанием на то, что для минимизации можно применить, скажем, квадратичную экстраполяцию или метод наискорейшего спуска). Детали могут повлиять на точность и скорость исполнения алгоритма, но пути решения задачи и форма контакта человека с машиной и машины с человеком при эксплуатации программы должны определяться на стадии составления проекта.

Проект, не перегруженный излишними подробностями, помогает автору обозреть и осознать свой собственный замысел, увидеть места, подлежащие корректировке, и дает возможность обсудить будущий алгоритм в его принципиальных аспектах со специалистами и коллегами. При создании проекта будущего алгоритма очень важно найти разумный компромисс между желанием сделать алгоритм весьма общим (таким, чтобы он включал в себя, как частные случаи, сразу несколько типов интересующих автора задач) и желанием сделать алгоритм простым и быстро исполняемым на вычислительной машине.

Для того чтобы написать проект алгоритма, не надо быть знакомым с программированием. Достаточно, вообще говоря, иметь представление об основных возможностях машины, о структуре ее памяти и о возможных формах входной и выходной информации.

Автор алгоритма, прежде чем думать о программировании своей задачи на языке команд доступной ему машины, должен выяснить, нет ли в архивах вычислительного центра, с которым ему предстоит иметь дело (или в архивах других организаций), уже готовых программ, реализующих нужный ему алгоритм. Нередко автор новой задачи обнаруживает, что, хотя готовой программы для решения в точности его задачи нет, имеется все же программа для решения другой задачи, достаточно близкой к его собственной, или позволяющей (иногда ценой известной изобретательности) интерпретировать интересующую автора задачу как некоторый частный случай запрограммированной задачи.

Просмотр чужих программ следует вести, конечно, по их описаниям, сделанным на «человеческом» языке и являющимся фактически проектами этих программ, откорректированными после их реализации. Понять назначение чужой, сколько-нибудь сложной, программы из текста самой этой программы (т. е. из последовательности машинных команд) практически невозможно, и даже самые опытные программисты обычно уклоняются от того, чтобы разбираться в чужих программах.

К сожалению, информация об имеющихся готовых программах очень скупа, и мы не можем указать какую-нибудь стандартную процедуру, которая позволила бы автору новой задачи установить, должен ли он сам заботиться о программировании своей задачи или может воспользоваться готовой программой.

Между тем вероятность того, что нужный алгоритм уже запрограмми-

рован, совсем не мала, ибо алгоритмы, решающие важные задачи в какой-либо отрасли, рождаются не очень часто, и за несколько лет активного применения вычислительных машин в библиотеках эксплуатирующих их организаций накапливается обычно обширный и ценный программный материал.

Может оказаться, что подходящая готовая программа предназначена не для той машины, которой первоначально собирался воспользоваться потребитель, а для машины другого типа. Иногда есть возможность и смысл изменить первоначальные намерения и провести счет на машине, для которой готовая программа имеется, а не программировать заново алгоритм для «своей» машины.

В настоящее время некоторое распространение получили программы-переводчики, автоматически переводящие программы, написанные на языке команд одной машины, в программы, использующие систему команд другой машины. Однако основное направление современного программирования состоит в разработке методов записи программ на специальных языках, не зависящих от типов машин, с последующим их переводом (трансляцией) на конкретные машинные языки для исполнения. Ниже об этом будет сказано подробнее.

Если автору задачи повезло, и обнаружилась программа, которую он готов использовать, то он может приступить к практически самостоятельной эксплуатации программы, даже если не умеет программировать. Для этого он должен получить текст программы, написанный на бланках, и отдать его в перфораторную для изготовления комплекта перфолент или перфокарт, содержащих программу. Еще практичнее получить не текст программы (в нем все равно трудно разобраться), а готовый комплект перфокарт или перфолент. Далее надо изучить инструкцию к программе. Инструкция — это документ, содержащий указания о форме записи исходной информации на стандартных бланках и о форме и порядке печати результатов, а также некоторые указания оператору для работы за пультом. Обычно «исходная информация» — это массивы чисел, записанные в нормальной десятичной форме. (Например, число — 253, 381 представляется в виде $-0{,}253381 \cdot 10^3$; на бланке пишется: — +03 253381000. Число 0,0000161 представляется в виде $0{,}161 \cdot 10^4$; на бланке будет написано + —04 161 000000; детали и отклонения от приведенных форм можно найти в описаниях конкретных машин.) Инструкция должна включать в себя пояснение смысла наименований переменных, используемых для обозначения исходных данных и результатов, а также указывать порядок, в котором числовые значения этих переменных вносятся соответственно на бланки и печатаются на бумажной ленте.

Числа сами по себе могут не охватывать информацию, которую надо включать в исходные данные или получать в качестве результатов. Поэтому часто приходится прибегать к числовой кодировке нечисловой информации. Можно, например, договориться, что группа данных, представляющих собой объемы производств, будет начинаться с двух строк, состоящих из всех семерок, а заключаться одной такой строкой; группу данных, представляющих собой цены, можно выделить, скажем, строками шестерок и т. д. Ясно, что такой способ имеет весьма ограниченные изобразительные возможности и, главное, неудобен, так как требует много труда на зашифровку и расшифровку признаков такого рода. Поэтому (за границей уже давно, а у нас в последнее время) вычислительные машины стали снабжаться алфавитно-цифровыми устройствами ввода и вывода информации. Эти устройства дают возможность писать на бланках и печатать на бумаге не только числа, но и буквы, знаки, символы и целые слова человеческого языка. Написанный на бланке текст печатается перфора-

торщицей на упомянутом вводном алфавитно-цифровом клавишном устройстве и автоматически превращается в систему пробивок на перфокартах и перфолентах, а после ввода перфокарт или перфолент в память машины — в систему нулей и единиц, которые легко расшифровываются программой.

При выдаче результатов системы нулей и единиц, подготовленные по специальным правилам программой, превращаются при печати в цифры, буквы и другие символы, позволяющие читать результаты счета непосредственно на обычном человеческом языке и понимать их без предварительного запоминания шифров, форм и порядка выдачи (как это приходится делать при наличии одной лишь цифровой печати).

После того как исходная информация написана на бланках, ее передают в перфораторную для пробивки. Комплекс перфокарт (или перфолент), содержащий готовую программу и пробитую исходную информацию, следует теперь передать операторам, которые в выделенное для потребителя время поставят задачу на счет в машине. Потребителю останется лишь получить, прочесть и, если надо, истолковать результаты. К сожалению, нарисованная схема может оказаться несколько идеализированной на фоне реальных условий. Если инструкция к программе написана недостаточно тщательно (к сожалению, это бывает), потребителю придется брать консультации у автора программы или у лиц, уже успешно ею пользовавшихся. Если предоставленная потребителю вычислительная машина не имеет специального штата операторов, принимающих отперфорированные программы к счету, то потребителю придется овладеть некоторым техминимумом для работы за пультом машины, подобно тому, как он в силу необходимости овладел искусством управлять своим телевизором. Но это, конечно, отклонение от нормы. Современная тенденция состоит в том, чтобы ограничить круг лиц, работающих за пультом, и по возможности автоматизировать процесс счета переданных машине программ.

Но что же делать, если подходящая готовая программа не найдена? Кто должен ее написать, отправляясь от проекта алгоритма?

Можно представить себе два ответа на этот вопрос, и оба они соответствуют двум сложившимся вариантам практического программирования задач.

Вариант первый. Автор передает свой проект программисту, т. е. специалисту, владеющему языком, доступным машине. Многие вычислительные центры располагают штатом программистов, назначение которого состоит в переводе чужих алгоритмов с языка проекта на язык программ. Программист, помимо знакомства с машинным языком, должен обладать некоторой квалификацией для «разворачивания» методов решения задачи, иногда лишь намеченных в проекте, в подробные машинные алгоритмы. Надо сказать, что в первые годы после появления электронных вычислительных машин специальность программиста считалась очень трудной и даже таинственной. Вначале непосредственным программированием занимались крупные математики, которые создали основополагающие приемы в этой области. Впоследствии, однако, применение этих приемов стало делом довольно стандартным, и в среде программистов произошла дифференциация между программистами-переводчиками чужих готовых (или почти готовых) алгоритмов и программистами, занятыми общими вопросами теории программирования и разработкой специальных программ, увеличивающих эффективность вычислительных машин (комплекты таких программ называют «математическим обеспечением машины»).

Услуги программиста-переводчика избавляют автора алгоритма от необходимости «влезать» в «машинную кухню», но такой вариант про-

граммирования делает процесс прохождения задачи от постановки до решения довольно длинным и создает предпосылки для появления ошибок, основанных на том, что программист может незаметно для себя и для автора алгоритма неверно понять отдельные места проекта. Выявлять такие ошибки очень трудно.

В настоящее время права гражданства завоевывает второй вариант подхода к программированию: программирует автор алгоритма. Эта схема особенно жизненна для учреждений, заводов, лабораторий, институтов, располагающих собственной вычислительной машиной. Сам факт существования машины привлекает к постановке задач широкий круг специалистов, и централизованная группа программистов-переводчиков быстро становится неспособной оперативно обслуживать своих заказчиков. Итак, если кому-либо предстоит часто пользоваться электронной вычислительной машиной, то ему надо научиться программировать. Человек, имеющий некоторый опыт составления алгоритмов на обычном языке, овладевает программированием сравнительно быстро.

Процесс составления программ упрощается, если в комплект «математического обеспечения» машины входит библиотека стандартных подпрограмм. Так называют программы, которые находятся постоянно во внешней памяти машины — на магнитной ленте или магнитном барабане. Наличие стандартных подпрограмм дает возможность автору программы при помощи простых приемов включать в машинные алгоритмы такие действия, как перевод чисел из одной системы счисления в другую, вычисление тригонометрических и других элементарных функций, приближенное решение дифференциальных уравнений, решение систем линейных алгебраических уравнений и т. п., не выражая их каждый раз при помощи соответствующей последовательности элементарных команд (как это нам пришлось сделать выше с синусом при переходе от первого алгоритма ко второму).

Процесс обучения программированию и, главное, процесс составления программ можно ускорить за счет использования алгоритмических языков для автоматизации программирования. Применение таких языков дает возможность лицу, программирующему свою задачу, не расчленять ее на отдельные команды, приспосабливаясь к схемным и конструктивным особенностям конкретной машины, а записывать ход ее решения в терминах и «единицах действия», рассчитанных скорее на исполнителя-человека, а не на исполнителя-машину. Специальная, раз навсегда составленная программа («транслятор»), постоянно находящаяся в памяти машины, автоматически переводит программу, написанную на алгоритмическом языке, в последовательность команд данной машины, которая вслед за тем и исполняется. Автору программы, написанной на алгоритмическом языке, как правило, нет нужны разбираться в структуре машинной программы, изготовленной транслятором. Так как алгоритмические языки включают в себя (в отличие от языка машинных команд) не только цифры, но и буквы, знаки препинания, математические символы, то их практическое применение возможно лишь при наличии у машин алфавитно-цифровых вводных устройств.

В настоящее время для записи вычислительных алгоритмов широко применяется международный язык АЛГОЛ-60.

Для примера приведем без комментариев алгоритм со стр. 598, написанный на языке АЛГОЛ-60 *.

---

* Читатели, знакомые с АЛГОЛом, заметят, что в примере можно было бы воспользоваться оператором цикла; это не сделано для сохранения наглядной связи с текстом алгоритма на стр. 598.

```
начало вещественные x0, h, x1; ввод (x0, h, x1);
        начало вещественный x; целый i;
                массив y [0 : (x1 — x0)/h];
                        x := x0; i := 0;
                счет таблицы: y [i] := sin (0.7854×x ↑ 2/(1 + x ↑ 2));
                        x : x + h; если x > x1, то на окончание
                        иначе i := i + 1; на счет таблицы;
                окончание: вывод (y); стоп
        конец
конец
```

Приведенную выше программу можно непосредственно решать на машине, имеющей в памяти транслятор *.

Для программирования задач с экономическим содержанием в настоящее время создается специальный алгоритмический язык (он называется АЛГЭК), базирующийся на АЛГОЛе, но содержащий дополнительные возможности для простой записи операций над сложными (со многими входами) таблицами данных и над текстами. Проект АЛГЭКа будет обсуждаться в сентябре 1965 г. на совещании представителей европейских социалистических стран, цель которого — выработать стандартный язык для программирования экономических задач в социалистических странах.

Вернемся теперь к описанию процесса решения задачи на машине. Независимо от того, как и на каком языке задача запрограммирована, полученная программа должна быть отлажена, т. е. в ней должны быть обнаружены ошибки, связанные с неверной реализацией курсов алгоритмов, в частности все случайные описки.

Читать снова и снова свою программу для выискивания в ней ошибок практически невозможно: всех ошибок при таком способе все равно не обнаружить, а времени такой процесс занимает много.

Обычно при отладке программу исполняют для нескольких «эталонных» наборов исходных данных с заранее сосчитанными эталонными результатами. Фактические результаты сравнивают с эталонными. Близость фактических и эталонных результатов повышает уверенность в правильности программы. Их расхождение заставляет искать ошибку в программе или в эталонных результатах. К сожалению, до сих пор не разработана теория, позволяющая назначать в каждом конкретном случае некоторую испытательную процедуру, которая, во-первых, была бы не слишком сложной для выполнения, а во-вторых, приводила бы к одному из выводов: «программа верна» или «программа не верна» с малой вероятностью того, что соответствующий вывод неверен. Требование простоты испытательной процедуры связано с тем, что получение как эталонных, так и фактических результатов сопряжено с затратой человеческого труда и машинного времени. Таким образом, высказанное требование можно перефразировать как требование ограниченности некоторой «цены» за отладку программы.

На практике конец отладки устанавливается весьма субъективно, и поэтому не исключены случаи, когда в отлаженной, казалось бы, программе, успешно эксплуатировавшейся в течение длительного времени, неожиданно, при счете с каким-нибудь очередным вариантом исходных данных, обнаруживается ошибка. Эта ошибка, например, может содержаться

---

** Не исключена лишь необходимость в некоторой редакции операторов «ввод», «вывод», «стоп» в связи с особенностями конкретных трансляторов.

в некоторой ветви программы, не обследованной по недосмотру при отладке и случайно не затрагивавшейся при счете со всеми предыдущими вариантами исходных данных.

Требования к надежности отладки повышаются с увеличением объемов решаемых на машине задач и с повышением роли, которую результаты машинного счета играют при принятии ответственных решений. Если программа, предназначенная для определения параметров детали какой-либо машины, содержит ошибку, не выявленную при отладке, это может привести к неверной рекомендации на конструкцию детали. Можно утешать себя тем, что испытания опытных образцов этой детали (предшествующие обычно запуску детали в серийное производство) выявят дефект расчета. Однако последствия ошибок в программе, дающей параметры для составления плановых заданий целой отрасли промышленности могут привести к весьма неприятным последствиям. Поэтому для контроля особенно сложных и ответственных («управляющих») программ целесообразно моделировать на машине целиком систему, управляемую подлежащей отладке программой. Часто оценка работы системы в целом оказывается «естественнее» и проще, чем оценка ее части — отлаживаемой программы. Задавая такой системе различные случайные условия ее существования, по статистическим результатам работы системы можно судить и о качестве исследуемой программы. Вообще, помимо формальных методов отладки программ, полезны и «содержательные методы», состоящие в контроле смысла полученных результатов. Неправдоподобные результаты свидетельствуют обычно об ошибке. Использование алгоритмических языков уменьшает вероятность появления ошибок (хотя и не исключает их полностью). Значительная часть ошибок автоматически вылавливается специальными блоками транслятора.

Но ошибки могут возникать даже при счете по идеально отлаженной программе. Машина может давать случайные сбои. Ошибки такого рода обычно легко обнаруживаются двойными просчетами с автоматическим сравнением результатов. Гораздо сложнее обнаруживать «небольшие» систематические ошибки в работе машины, которые, накапливаясь, могут совершенно искажать результаты. В последнее время вычислительные машины все чаще и чаще снабжаются специальными электронными схемами, осуществляющими так называемый аппаратный контроль за правильностью выполнения каждой команды. Такой контроль и система специальных программ-тестов, периодически исполняемых на машине, делает счет очень надежным.

Другой источник ошибок — это неверные исходные данные. Ошибки при переносе информации с бланков на перфокарты можно практически исключить работой перфораторщиц в «две руки» с последующим автоматическим сравнением комплектов перфокарт. Ошибки на первичных бланках искать труднее. Полезно программировать перед решением задачи выдачу на печать только что введенных в машину исходных данных, но при большом объеме исходной информации просмотр обширных массивов чисел может оказаться для поиска ошибки совершенно неэффективным. Для контроля ошибок такого рода можно и здесь рекомендовать «содержательный» контроль по легко обозримым и «понятным» результатам.

Мы затронули вопрос об объеме информации, выдаваемой на печать. Какие величины выдавать на печать, в каком объеме их печатать, какую форму следует придавать информации — к продумыванию этих вещей стоит приложить необходимые усилия, причем это полезно сделать на стадии составления проекта. Распространенная среди лиц, не имеющих опыта общения с машиной, тенденция печатать «на всякий случай» побольше данных, как правило, вредна, ибо приводит не только к излишним расходам

машинного времени, но и к неоправданной загрузке людей разбором и анализом незначащей информации. Человек сам представляет собой информационно-вычислительную систему с ограниченной пропускной способностью, и надо в каждом отдельном случае разумно определить характер и объем информации, которую он может эффективно использовать.

Можно ожидать, что контакты человека с машиной будут совершенствоваться за счет сближения их языков — как на входе машины (способы записи алгоритмов и исходных данных), так и на выходе машины (объем и форма выдачи результатов).

## РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Е. А. Жоголев, Н. П. Трифонов. Курс программирования. М., «Наука», 1964.
2. Д. Д. Мак-Кракен. Программирование на АЛГОЛЕ. М., «Мир», 1964.
3. С. С. Лавров. Универсальный язык программирования. М., «Наука», 1964.
4. Т. Боттенбрух. Структура АЛГОЛ-60 и его использование. М., Изд-во иностр. лит., 1963.

Поступила в редакцию
3 V 1965