ЭВМ

В ЭКСПЕРИМЕНТАЛЬНОЙ ФИЗИКЕ

1968 г.

ОБЪЕДИНЕННЫЙ ИНСТИТУТ ЯДЕРНЫХ ИССЛЕДОВАНИЙ

ЛАБОРАТОРИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И АВТОМАТИЗАЦИИ

10 - 4221

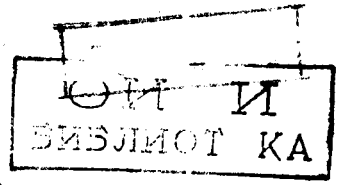
С.С.Лавров [x/]

СОСТОЯНИЕ И ПЕРСПЕКТИВЫ РАЗВИТИЯ МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ ЭВМ

Лекция, прочитанная в Школе ОИЯИ по применению электронных вычислительных машин в задачах экспериментальной физики

г. Алушта, Крым, СССР, 5-19 мая 1968г.

[x/] ВЦ АН СССР.



Дубна 1968

Широкое внедрение электронных вычислительных машин в различные звенья физического эксперимента за последние годы вызвало определенный интерес со стороны физиков-экспериментаторов к вопросам вычислительной техники и программированию. Персонал физических лабораторий при подготовке экспериментов или обработке экспериментальных данных вынужден (в большинстве случаев самостоятельно) осваивать технику ЭВМ и методы работы на вычислительных машинах.

При всем многообразии материала как по самим вычислительным машинам, так и по вопросам программирования, в процессе такой работы возникают естественные трудности, связанные, главным образом, с ограниченностью литературы, рассчитанной на физика-экспериментатора или на лиц, занимающихся развитием методических вопросов экспериментальной физики. Если учесть при этом, что методика использования ЭВМ в экспериментальной физике быстро совершенствуется, то будет понятен интерес со стороны физических институтов к летней школе Объединенного института ядерных исследований - "Применение ЭВМ в задачах экспериментальной физики".

Школа проводилась Лабораторией вычислительной техники и автоматизации ОИЯИ (директор - член-корреспондент АН СССР проф. М.Г.Мещеряков) в г.Алуште (Крым) с 5 по 19 мая 1968 года.

Программа школы наряду с основополагающими вопросами включала также лекции по некоторым конкретным современным методикам. Для чтения лекций были приглашены ведущие специалисты из Объединенного института ядерных исследований, институтов стран-участниц ОИЯИ, а также коллеги из европейских исследовательских центров - ЦЕРНа (Швейцария) и Сакле (Франция).

Не имея возможности опубликовать весь материал, ректорат Школы подготовил к изданию отдельные лекции, сохранив, в основном, их в том виде, в котором они были представлены авторами.

Лиц, интересующихся лекциями в полном объеме, мы адресуем в библиотеку ОИЯИ, где находится полный сборник прочитанных в школе лекций: "Применение ЭВМ в задачах экспериментальной физики".

Ректор Школы

доктор технических наук

Г.ЗАБИЯКИН.

Отпечатано методом ксерокс-ротапринт с материалов, подготовленных ректоратом Школы.

I. Три поколения систем математического обеспечения

В истории развития электронных вычислительных машин принято различать три поколения. Соответствующие поколения можно выделить и в развитии математического обеспечения этих машин. Для каждого поколения можно также назвать типичные классы задач и характерные приемы организации работы на машинах. В таблице I приведена эта классификация. Как всякая классификация, она условна. Одно поколение зарождалось в период расцвета предыдущего, постепенно приходило ему на смену и продолжало держаться до тех пор, пока ведущие позиции захватывало следующее поколение. Если, например, вычислительные задачи указаны в таблице как типичные для первого поколения, это не значит, что они перестали решаться в эпоху второго и третьего поколений. Напротив, они решались более интенсивно и с большим размахом, в их решении был достигнут ряд успехов и возникло много новых идей, но основное внимание и программистов, и конструкторов машин, и разработчиков математического обеспечения было приковано к другим типам задач, перечисленным в соответствующей рубрике. То же можно сказать и о любой другой графе таблицы.

Таблица I.

Поколение	Типичные задачи	Организация работы на машине	Математическое обеспечение
I	Вычислительные	Личный счет	Стандартные подпрограммы Компилирующие и интерпретирующие системы Языки символического кодирования (автокоды I:I) Отладочные программы (прокрутка, контроль) Автокоды (машинно-ориентированные алгоритмические языки)
II	Обработка экономической информации Обработка символьной информации Обработка эксперимента в реальном времени	Операторский счет Пакетная обработка	Модульное программирование Проблемно-ориентированные языки Системы программирования, включающие средства отладки Автооператоры
III	Автоматическое управление Автоматическое проектирование	Многопрограммная работа Разделение ресурсов Совместная работа человека с машиной	Супервизоры Мониторы Универсальные алгоритмические языки Программы обслуживания вынесенных пультов Информационные архивы.

Тем не менее, каждому поколению свойствен свой особый дух, накладывающий отпечаток на все стороны использования машин.

2. Первое поколение - эпоха машинных языков

В наши дни под математическим обеспечением понимают комплекс программ, направленных на облегчение труда программиста. С этой точки зрения первые электронные машины не имели никакого математического обеспечения. Но можно взглянуть на это и по-другому. Элементы, из которых строились машины, были способны выполнять лишь простейшие операции - обнаружить одновременное поступление двух сигналов на входе элемента или поступление хотя бы одного из этих сигналов и т.п. На основе таких элементарных операций в машинах удалось реализовать выполнение четырех арифметических действий, иногда даже над числами с плавающей запятой. Если угодно, это можно считать первичным математическим обеспечением, встроенным в конструкцию машины.

Первой разновидностью математического обеспечения в современном смысле этого слова были стандартные подпрограммы. Машины были созданы для решения вычислительных задач и вначале на них решались почти исключительно такие задачи. Первые стандартные подпрограммы также носили вычислительный характер. Это были программы перевода чисел из десятичной системы счисления, привычной человеку, в двоичную, в которой работала машина, программы вычисления элементарных функций. Часто такие программы "впаивались" в постоянную память машины, расширяя набор ее операций. Это значит, что конструкторы машины хорошо понимали в то время, что математическое обеспечение -

неотъемлемая принадлежность ЭВМ. Затем появились подпрограммы ввода и вывода, без которых вскоре не мог уже обходиться ни один программист, так как любая программа составлялась для обработки каких-то исходных данных, которые вместе с самой программой нужно было вводить в машину, и для получения результатов, которые нужно было выводить, затрачивая на это как можно меньше человеческого труда и используя форму представления, наиболее удобную для человека.

Другие классы вычислительных задач, для которых составлялись разнообразные стандартные подпрограммы, - это аппроксимация функций, интерполяция, задачи линейной алгебры, решение нелинейных алгебраических и трансцендентных уравнений и систем уравнений, вычисление определенных интегралов, численное решение обыкновенных дифференциальных уравнений, вычисление специальных функций, отыскание экстремумов функций одной и многих переменных, обработка статистических данных, моделирование на машине случайных величин и процессов.

Пока число стандартных подпрограмм было невелико, они размещались каждая на своем месте в оперативной памяти. Место, не занятое стандартными подпрограммами и занятое подпрограммами, не нужными в данной задаче, оставалось в распоряжении программиста. Но скоро подпрограмм стало так много, что уже нельзя было для каждой из них выбрать свое постоянное место, да и разбросанность их по всей оперативной памяти причиняла неудобства. Надо было научиться двигать подпрограммы с места на место.

Если программа написана в расчете на то, что ее первая (по месту, не обязательно по порядку выполнения) команда помещается в ячейке с адресом α, длина подпрограммы равна ℓ,

то для перемещения ее на другое место с начальным адресом b необходимо все адреса в командах подпрограммы из диапазона $a \div a+l-1$ изменить на величину $b-a$. Такое преобразование подпрограммы при ее перемещении называется настройкой по месту. Эту настройку нетрудно сделать вручную, но тогда преобразованную подпрограмму надо вновь перфорировать и вводить в машину. Чтобы механизировать эту работу, были созданы компилирующие и интерпретирующие системы использования стандартных подпрограмм, хранящихся во внешней памяти машины или даже - на перфокартах.

В случае компиляции подпрограммы однократно вводились в оперативную память и настраивались на то место, куда они попадали.

В режиме интерпретации подпрограмма вызывалась и настраивалась по месту каждый раз, когда основная программа должна была к ней обратиться (если только эта подпрограмма не сохранялась в пригодном для использования виде от предыдущего обращения к ней). Интерпретация - более общий режим, позволяющий более экономно использовать место в памяти. Можно ограничиться участком оперативной памяти, достаточным для размещения наибольшей цепочки подпрограмм, обращающихся одна к другой (работающих параллельно). Если подпрограммы работают последовательно, они в режиме интерпретации могут располагаться на одном и том же месте. Конечно, экономия места достигается, как это типично для программирования, ценой затрат времени - в данном случае, времени на повторный вызов и настройку подпрограмм.

Компилирующие и интерпретирующие системы позволяли довольно легко включать в программу не только стандартные подпрограммы, но и любые куски программы, написанные любым программистом в стандартной для данной системы форме.

Кроме настройки по месту, системы использования стандартных подпрограмм позволяли в некоторых случаях осуществлять настройку по параметрам. Для примера рассмотрим стандартную подпрограмму решения системы линейных алгебраических уравнений. Некоторые адреса этой подпрограммы зависят от порядка системы и от начального адреса матрицы коэффициентов (а также, возможно, от начального адреса массива свободных членов и массива - группы ячеек,- отведенного для размещения решения системы после его отыскания). Перечисленные величины называются параметрами данной подпрограммы, а вычисление и подстановка на нужное место адресов, зависящих от этих параметров, носят название настройки подпрограммы по параметрам. Часто настройка по параметрам осуществляется внутри самой подпрограммы, но некоторые общие приемы такой настройки могут быть реализованы и при обращении к подпрограмме, то есть во время компиляции или интерпретации, тогда они становятся принадлежностью соответствующей системы.

Стандартные подпрограммы и системы их использования, родившиеся на первом этапе развития математического обеспечения, не утратили своего значения до сих пор, и было бы большой ошибкой при создании любой современной системы программирования не учитывать необходимости использования библиотеки стандартных подпрограмм.

Составление стандартных подпрограмм, как и все программирование, велось вначале исключительно в машинном языке. Программирование было новым и довольно увлекательным занятием, своего рода искусством. Программистов на первых порах требовалось не очень много, и ими обычно становились довольно квалифицированные математики. Овладеть тонкостями программирования

в машинном языке для них не представляло большого труда. Составление как можно более коротких и быстро работающих программ вызывало спортивный азарт. Единственной досадной вещью было то, что машинные команды требовалось писать в том виде, в каком они должны были попадать на регистр команд, например,

001 1706 3425 7730 . (1)

(Даже эта запись является сокращением истинного вида команды, представляющей собой такую последовательность нулей и единиц:

000 000 001 001 111 000 110 011 100
010 101 111 111 011 000).

Написанная команда является машинным эквивалентом формулы, которая в обычных математических обозначениях имеет, допустим, вид:

$$x = a + b \qquad (2)$$

если, распределяя память, программист отвел для величины a ячейку с адресом 1706, для b - ячейку 3425 и для x - ячейку 7730. Естественно, что программисту хотелось бы команду (1) писать в привычном виде (2), или, на худой конец, в виде

$$+\ a\ b\ x\ , \qquad (2')$$

сохраняя принятую в машине последовательность записи знака операции (+), операндов (a и b) и результата операции (x).

Подавляющее большинство так и стало поступать. Машинные команды они записывали в виде (2) или (2'), затем составляли таблицу распределения памяти, которая выглядела так:

a	1706
b	3425
x	7730
	....

(3)

После этого перекодировка программы к виду (I) становилась чисто механической работой и могла быть поручена лаборанту. Наиболее сообразительные программисты поняли, что эту работу быстрее и аккуратнее лаборанта может выполнить сама машина. Так родились языки символического кодирования – набор правил, точно регламентировавших запись машинных команд в виде (2) или (2'), и "автокоды I:I" – программы, осуществляющие перевод с языка символического кодирования в машинный язык, то есть в форму (I), с помощью таблицы (3). Языки символического кодирования оставались машинными языками, но с использованием буквенных обозначений адресов вместо цифровых. Символ I:I в названии автокода подчеркивал взаимооднозначное соответствие между командами в языке символического кодирования и в машинном языке.

Далее было замечено, что подавляющее большинство программ составляется с использованием небольшого числа правил, гарантирующих для этих програм приемлемое качество программирования. Так, например, для вычислений по формуле

$$y = a \times x + c/z \qquad (4)$$

любой программист напишет (для трехадресной машины) три команды

$$\begin{aligned} r &= a \times x \\ r_1 &= c/z \\ y &= r + r_1 \end{aligned} \qquad (5)$$

Но те же три команды может составить и автокод, если его дополнить некоторыми новыми разделами, устанавливающими порядок действий в формулах типа (4) и выписывающими последовательность машинных команд, реализующих эти действия. Другие дополнительные разделы автокода могли включать в программу команды, обеспечивающие в нужных случаях разветвление вычис-

лительного процесса, циклическое повторение участков программы и другие вещи, вытекающие уже не из особенностей машины, а из структуры задачи. Разумеется, в язык, предназначенный для описания задач, должны быть при этом включены средства, указывающие на необходимость разветвления или зацикливания процесса, на все то, что должно, в конечном счете, войти в машинную программу.

Так постепенно автокоды утратили приставку "I:I", а их входные языки стали не чисто машинными, а машинно-ориентированными. Машинная ориентированность означает, что в основе этих языков продолжала лежать система команд какой-либо конкретной вычислительной машины.

Таким образом, первое поколение систем математического обеспечения было тесно связано с программированием на языке машины. В силу особенностей машинных языков, далеких от традиционной символики, и в связи с трудоемкостью самого процесса программирования редкук программу удавалось написать без ошибки. Ошибки появлялись и в результате простых описок и неправильной перфорации программ, и потому, что программист писал не те команды, которые он хотел написать, то есть не соответствующие выбранному им алгоритму - методу решения задачи, и потому, что сам выбранный метод не соответствовал поставленной задаче, не давал ее решения. Все ошибки надо было найти и исправить. Эта работа называлась отладкой программ. Часто на отладку уходило больше времени, чем на решение задачи по отлаженной программе.

Отладку обычно вел сам программист за пультом машины. Применялись следующие основные приемы отладки. Выполнялась часть программы до того места, где подозревалось появление ошибки. Просматривалось содержимое некоторых ячеек памяти. Выпол-

нялась часть программы в однотактном режиме, при этом надо было следить за содержимым регистров машины(арифметических, регистра и счетчика команд)после каждой выполненной команды. Включался останов по записи в ячейку, содержимое которой в процессе работы программы отличалось от того,что ожидал программист. Отпечатывалось содержимое группы ячеек (массива чисел или части программы).

Последний прием был наиболее эффективным, так как он позволял проследить состояние всей программы или наиболее существенной ее части в критический момент ее работы. Но и он требовал больших затрат машинного времени, так как предварительно надо было нащупать этот момент, заставить программу проработать до этого состояния, может быть,с целым рядом промежуточных остановов, подготовить печать (ввести или вызвать программу печати, набрать на пульте параметры печати - диапазон адресов и указание — что печатать - числа или команды). Другие методы отладки были еще более расточительными. За время раздумий и манипуляций программиста за пультом машина, даже не очень быстродействующая, успела бы выполнить сотни тысяч, если не миллионы команд. Ясно, что эту работу необходимо механизировать и ускорить.

Были созданы различные программы отладки. Эти программы использовали преимущественно режим прокрутки, при котором команды извлекались из программы в том порядке, в каком они должны были выполняться, исследовалось содержимое операндов (ячеек, указанных в адресах команды) и результат выполнения команды, который засылался в требуемую ячейку, после чего начинался такой же анализ очередной команды. Вся эта информация - адрес выполняемой команды и сама команда, значения операндов и результата - печаталась по ходу прокрутки. В более экономных режимах печаталась лишь часть информации;

например, адреса переходов, встречавшихся в программе. Другие варианты отладочных программ предусматривали организацию в отлаживаемой программе контрольных точек. От точки до точки программа выполнялась в обычном режиме, то есть без потерь времени на контроль. В контрольной точке могла быть отпечатана информация о состоянии программы, заданная в плане контроля, выполнена какая-либо группа команд, не предусмотренных в основной программе, или даже включен режим прокрутки до следующей контрольной точки. Расположение контрольных точек и план отладки задает программист. Работой отладочных программ можно было управлять с пульта, включая или отключая выполнение операций контроля.

Наиболее совершенные отладочные программы позволяли значительно сократить потери времени на отладку. Однако контроль сопровождался, как правило, печатью обширной избыточной информации, так как программисту трудно было предусмотреть, где и что именно потребуется отпечатать для выявления ошибки, даже когда он примерно знал, в чем она проявляется. Многие программисты отказывались от применения отладочных программ, настаивая на предоставлении им возможности самим посидеть за пультом, наблюдая за работой своей программы.

Их можно легко понять. За пультом программист ведет отладку активно, выясняя то, что его действительно интересует в данный момент, планируя отладку на основе получаемых сведений. Более того, сам процесс общения со своей программой, наблюдение за ее жизнью доставляет массу впечатлений, не всегда приятных ("Вот шляпа-то!" - типичная реакция на обнаруженную ошибку), но всегда волнующих. Программиста охватывает азарт, он исправляет одну ошибку (делая при этом новую), другую, пытаясь заставить программу наконец-то работать правильно.

Отладка за пультом - это одно из самых эмоционально насыщенных занятий, требующее от программиста максимальной внимательности, сообразительности, быстрой реакции. Далеко не каждый программист обладает требуемыми качествами, чтобы быстро добиваться успеха в отладке.

В работу на машинах вовлекалось все больше лиц, далеких от математики. Машины находили все более разнообразное применение, спрос на них обгонял рост их выпуска. Машинное время стало цениться дороже. Большинство машин использовалось круглосуточно, а заниматься отладкой по ночам соглашались лишь немногие энтузиасты. Надо было находить другие приемы программирования, облегчающие этот процесс, уменьшающие вероятность ошибок в программах, и другие методы организации работы на машинах. Создание таких приемов и методов означало приход второго поколения систем математического обеспечения.

3. Второе поколение - эпоха алгоритмических языков

Это поколение проникнуто духом коллективной работы на машинах. Математики, без которых на первых порах трудно было бы обойтись, начали решать задачи совместно с экономистами, биологами, лингвистами и представителями других специальностей, прежде далеких от использования математических методов. Сами задачи стали в среднем сложнее, детально знать программу решения задачи одному человеку стало зачастую трудно. Если раньше обмен программами, совместная работа над ними были редки, вовлекали небольшое число близких сотрудников, то теперь стали появляться большие программы, пользоваться которыми должны были сотни и тысячи человек. Разрабатывались такие программы коллективами, объединявшими десятки членов.

Прежде чем перейти к основному достижению этого поколения - алгоритмическим языкам - несколько слов о так называемом мо-

дульном программировании. Оно развивало идею стандартных подпрограмм. Модули представляли собой более специализированные подпрограммы, относящиеся к какой-либо одной отрасли науки или техники. Такие модули позволяли строить из них программы решения задач этой отрасли почти целиком, как здания из крупных блоков, используя лишь небольшое количество связующего материала в виде команд, написанных программистом. Подобно стандартным подпрограммам, модули требовали наличия системы компиляции, реже - интерпретации, но и эти системы становились специализированными, отражая специфику отрасли.

Хорошо продуманная и отработанная система модулей позволяла быстро составлять весьма эффективные программы. Но если задача существенно выходила за рамки, предусмотренные системой, то возведение здания программы требовало множества надстроек, осложнявших строительство и уродующих сооружение.

Коллективный дух поколения сказался и на организации счета. Программистов перестали пускать за пульт. От них стали требовать, чтобы они писали инструкции к своим программам, описывая последовательность действий по запуску программы, отладочные операции, объем и характер материала, выводимого на печать, признаки нормальной или, наоборот, неправильной работы программы, ожидаемое время счета и т.п. За пультом стали работать лаборанты-операторы, от которых требовалось лишь точно выполнять требования инструкции, не вникая в смысл программы. Дисциплина программирования (и поведения в машинном зале) укрепилась, но радость живого общения со своей программой была утрачена.

Разумеется, возможность такой организации работы была обеспечена развитием тех средств математического обеспечения, о которых говорилось выше, в первую очередь, программ отладки,

позволявших обойтись без присутствия автора программы около машины во время работы программы.

Символом второго поколения можно считать проблемно-ориентированные языки программирования (алгоритмические языки) - прямые потомки входных языков автокодов. Развитие языков автокодов определялось все в большей степени спецификой задач, а не особенностями машин. На первый план выступило то общее, что было в различных задачах, а это сближало разные языки, так как они развивались еще в эпоху господства вычислительных задач.

Появление языков, целиком ориентированных на специфику задач, и не зависящих от конкретной машины, стало тем более неизбежным, что машины разных марок быстро сменяли одна другую или использовались совместно. Обучать армию программистов каждому новому машинному языку стало трудно и нерентабельно.

Одним из первых и наиболее удачных языков такого рода стал ФОРТРАН - детище фирмы ИБМ (США). Он не только просуществовал до наших дней, но и уверенно удерживает первое место в мире по распространенности, особенно в западных странах. Он прошел через ряд ступеней развития, причем программы предыдущей ступени оставались правильными программами и для следующей версии языка. Трансляторы (так стали называться программы перевода с машинно-независимых языков на машинные языки, а потом и вообще программы перевода с одного искусственного языка на другой) для ФОРТРАНа обладали довольно простой структурой. С их помощью получались машинные программы - результат трансляции - хорошего качества, лишь немного уступающие программам, составленным вручную для решения такой же задачи.

Через несколько лет после ФОРТРАНа появился язык АЛГОЛ 60, созданный на основе широкого международного сотрудничества. Ему не удалось превзойти ФОРТРАН по совокупности своих качеств, но он повсеместно признан как весьма удобное средство для публикации алгоритмов и для обучения основам программирования. АЛГОЛ 60 оказал большое влияние на разработчиков последующих алгоритмических языков как богатством идей, в нем содержавшихся, так и попыткой дать точное формализованное описание языка, исключающее кривотолки (эта цель не была достигнута и едва ли достижима вообще, как показывает опыт формализации основ математики и любой другой науки).

Как и ФОРТРАН, АЛГОЛ 60 оказался удобен для описания прежде всего вычислительных задач, хотя авторы и стремились сделать его универсальным. Поэтому примерно в то же время появились алгоритмические языки с другой ориентацией, отвечавшие нуждам тех новых направлений, которые стали интенсивно развиваться в эти годы.

Экономические задачи - это, главным образом, задачи учета материальных ценностей, выпущенной продукции, выполненных работ, личного состава и, в конечном счете, финансов предприятия, фирмы или отрасли. К ним примыкают задачи управления производством и планирования. Для большинства этих задач характерен большой объем исходной и результирующей информации и относительно очень небольшое количество вычислений. Типичный режим работы машины при решении таких задач - это ввод информации с внешнего запоминающего устройства в оперативную память (обычно, отдельными порциями), несложная ее обработка (простые вычисления, сортировка по какому-нибудь признаку) и вывод снова на внешний носитель (магнитную ленту, диски, реже-на перфокарты). Для описания действий такого рода

той же фирмой ИБМ был предложен язык КОБОЛ, также имевший и сохранивший заслуженный успех среди пользователей. Существует много других языков для той же цели.

Задачи обработки символьной информации возникают преимущественно в области научных исследований. Это, например, преобразование формул, решение уравнений, не численное, а в аналитическом виде, анализ и синтез текстов на искусственном или естественном языке (в частности, автоматическое программирование и машинный перевод) и т.п. Упомянем два языка, предназначенных для описания подобных задач. В языке ЛИСП вся находящаяся в обработке информация, в том числе и сама программа, организуется в так называемые списки - последовательности элементов. Элемент может быть первичным (буквенным обозначением или числом) или, в свою очередь, списком. Так могут возникать сколь угодно сложные структуры. Примером может служить алгебраическое выражение, в котором всегда можно выделить главный знак операции - той операции, которая должна выполняться последней в этом выражении. Этот знак и два соответствующих операнда образуют список из трех элементов. Операнды, если они не являются первичными элементами, могут быть подвергнуты такому же анализу и т.д. С выражением, в котором таким способом выделена его структура, легко производить различные действия. Язык ЛИСП сейчас очень популярен, преимущественно среди представителей наук физико-математического цикла, так как его применение требует известной математической культуры.

Язык СНОБОЛ, наоборот, пользуется успехом у гуманитариев. В нем основным понятием является строка - произвольная последовательность букв, цифр и других знаков. Главная операция - поиск в строке части строки, построенной по заданному образцу,

и замена этой части другой строкой. Как образец, так и замена составляются из отдельных элементов простого вида. Исход поиска определяет последовательность дальнейших действий. Язык крайне прост для изучения.

Основное достоинство проблемно-ориентированных, машинно-независимых алгоритмических языков в том, что они были построены с максимальным учетом представлений человека, если не о существе, то о форме решаемой задачи, с максимальным приближением к той форме, в которой человек привык описывать эти задачи, и с учетом тех логических связей, которые он привык выделять в исследуемых явлениях.

Для АЛГОЛа, например, характерно приближение к привычной математической символике (но, конечно, и лучшее, чем у его предшественников, отражение структуры вычислительной задачи). Для ЛИСПа - использование аппарата рекурсивных описаний, широко применяемого в математической логике и в исследованиях по основаниям математики, да и вообще родственного по духу научному описанию явлений.

Благодаря этим свойствам проблемно-ориентированных языков овладеть ими и правильно программировать на них стало гораздо легче, чем это было с машинными языками. Число ошибок в программах резко сократилось, соответственно упростилась и ускорилась отладка. Программы, которые правильно работали с первого же запуска, перестали быть редкостью. И хотя использование алгоритмических языков связано с потерями времени на трансляцию, да и транслированные программы работают медленнее, чем составленные вручную, общие затраты машинного времени на решение задачи вряд ли заметно изменились. А время (календарное) решения задачи - от начала программирования до получения результатов - сильно сократилось, благодаря упрощению программирования и отладки.

Алгоритмических языков было изобретено великое множество - несколько сот или даже тысяч. Большинство из них не получили никакого распространения, и лишь десяток-другой завоевали широкое признание. Сейчас стало очевидным, что создавать новые языки разумно на основе широкого, лучше всего, международного сотрудничества и в случае крайней необходимости. Может быть, исключение стоит сделать для узкоспециализированных языков, но, вероятно, здесь полезнее программные модули, о которых шла речь выше.

На базе алгоритмических языков стали создаваться системы программирования. Помимо входного языка и транслятора с него, хорошая система программирования содержит библиотеку стандартных подпрограмм и аппарат их использования, возможность включать в программу куски, написанные на машинном языке (в символических обозначениях, тождественных обозначениям входного языка), средства компоновки программ из независимо написанных частей, и средства отладки. Последние должны быть не менее гибкими, чем описанные выше отладочные программы для ручного программирования. При обнаружении ошибки, при невозможности выполнять программу дальше, или по указанию оператора они должны выдавать сведения о ситуации, в которой прерваны вычисления, в обозначениях входного языка. Должен печататься участок программы, на котором произошел останов, и значения, а также обозначения переменных, участвующих в вычислениях на этом участке или указанных программистом. Внедрение систем программирования еще больше упростило и ускорило подготовку задач к решению и отладку программ.

Рост производительности машин привел к усложнению работы операторов за пультом. Встал вопрос о механизации их труда. Это потребовало внесения некоторых изменений в конст-

рукцию машин, в результате которых каждая аварийная ситуация, а также программный признак конца задачи стали приводить не к останову машины, как раньше, а к так называемому прерыванию решения задачи. После останова требовалось вмешательство человека, чтобы машина возобновила работу. При прерывании начинает работать специальная программа-автооператор. Она всегда находится в памяти машины и отвечает на прерывание тем, что разбирается в причине прерывания и в зависимости от обнаруженной ситуации либо предпринимает стандартные действия, например, при делении на нуль производит аварийную выдачу (печать отладочной информации - см. выше), по окончании задачи - очищает память и вводит следующую, может быть, печатая протокольные сведения о закончившейся задаче , либо выполняет действия, указанные автооператору в специальной программе-инструкции, составленной программистом. Лишь в крайнем случае, например, при неправдоподобных для исправной машины явлениях, автооператор останавливает машину и дает сигнал оператору-человеку. Правда, предварительно автооператор может вызвать тест - программу, проверяющую работу сомнительного устройства машины.

Во всех других случаях решение задач осуществляется непрерывным потоком. Такой режим получил название режима пакетной обработки задач. При исправной машине роль человека за пультом свелась лишь к смене бумаги на печатающем устройстве, установке новых магнитных лент в случае необходимости по сигналу от автооператора, подкладыванию на читающее устройство новых пакетов программ и т.п. Человек (оператор) почти перестал тормозить работу машины и служить источником ее неправильной работы.

рукцию машин, в результате которых каждая аварийная ситуация, а также программный признак конца задачи стали приводить не к останову машины, как раньше, а к так называемому прерыванию решения задачи. После останова требовалось вмешательство человека, чтобы машина возобновила работу. При прерывании начинает работать специальная программа-автооператор. Она всегда находится в памяти машины и отвечает на прерывание тем, что разбирается в причине прерывания и в зависимости от обнаруженной ситуации либо предпринимает стандартные действия, например, при делении на нуль производит аварийную выдачу (печать отладочной информации - см. выше), по окончании задачи - очищает память и вводит следующую, может быть, печатая протокольные сведения о закончившейся задаче , либо выполняет действия, указанные автооператору в специальной программе-инструкции, составленной программистом. Лишь в крайнем случае, например, при неправдоподобных для исправной машины явлениях, автооператор останавливает машину и дает сигнал оператору-человеку. Правда, предварительно автооператор может вызвать тест - программу, проверяющую работу сомнительного устройства машины.

Во всех других случаях решение задач осуществляется непрерывным потоком. Такой режим получил название режима пакетной обработки задач. При исправной машине роль человека за пультом свелась лишь к смене бумаги на печатающем устройстве, установке новых магнитных лент в случае необходимости по сигналу от автооператора, подкладыванию на читающее устройство новых пакетов программ и т.п. Человек (оператор) почти перестал тормозить работу машины и служить источником ее неправильной работы.

4. Третье поколение - эпоха мультипрограммирования

Если второе поколение проходило под знаком коренных изменений в технике программирования, то третье поколение связано со столь же радикальным переворотом в организации работы на машинах, вызванным появлением многопрограммных (мультипрограммных) машин.

Предпосылки для многопрограммной работы создала замена остановов прерываниями. Скажем коротко, что это позволило заставить все основные устройства машины работать параллельно. В работе арифметического устройства и устройства управления, быстродействие которых (в сравнимых показателях) намного обогнало быстродействие других устройств, возникали окна. Теперь эти окна стало возможным заполнить работой программ, улучшающих использование всех устройств машины в целом.

При этом оказалось даже необходимым для равномерной загрузки всех устройств пропускать на машине несколько программ параллельно. Пока, к примеру, одна задача ждет окончания считывания очередной порции данных с магнитной ленты, для другой задачи можно подготовить вывод результатов на печать, а пока идет печать, точнее, пока арифметическое устройство свободно от подготовки очередной строки для печати, его можно загрузить вычислениями для третьей задачи. Как только закончится чтение с ленты или печать строки, эти вычисления будут прерваны и продолжено решение первой задачи или, соответственно, подготовка новой печатной строки для второй задачи и т.д. Такой режим работы и назван многопрограммным режимом или мультипрограммированием.

Организация многопрограммной работы потребовала создания сложных управляющих программ. Та из этих программ, ко-

торая реагирует на сигналы прерывания, идущие от различных устройств самой машины, и координирует работу этих устройств, может быть названа супервизором. Другая программа - монитор, планирует и организует прохождение нескольких задач, учитывая их приоритет (срочность), ожидаемую продолжительность работы, характер загрузки ими внешних устройств и требования задач на выделение в их распоряжение тех или иных ресурсов машины (участков оперативной памяти, устройств ввода-вывода и внешних запоминающих устройств). Кроме того, в функции управляющей программы входит все то, что делал автооператор (предшественник и прообраз современных программ-диспетчеров) - реакция на сигналы прерываний, связанные с обнаруженными сбоями или идущие от выполняемых программ. Число тех и других сигналов заметно возросло. В машинах стал применяться более широкий контроль правильности выполнения операций и передачи данных. Любая обнаруженная неисправность вызывает прерывание и требует разбирательства для принятия мер. С другой стороны, многие операции, которые в однопрограммной машине выполнялись по командам самой программы, теперь стали требовать обращения к диспетчеру (т.е. прерывания), так как их выполнение должно быть скоординировано с выполнением аналогичных запросов других программ. К таким операциям относятся обращения к внешним устройствам, обращения к стандартным подпрограммам и др.

Программы-диспетчеры, некоторые функции которых были очень коротко перечислены, стали важнейшим звеном в системе математического обеспечения любой многопрограммной машины. К диспетчеру должна быть привязана любая система программирования, цель которой - подготовить и облегчить решение отдельной задачи. Диспетчеры, в свою очередь, должны разрабатываться с учетом нужд одной или нескольких систем програм-

мирования, существующих или проектируемых для данной машины. Все это, вместе взятое, называется операционной системой - синоним системы математического обеспечения. Следует оговориться, что терминология в этой области далеко еще не установилась.

При решении сразу нескольких задач на многопрограммной машине ресурсы машины, как было сказано, распределяются между этими задачами, некоторые - временно, другие - постоянно.

Самым ценным из этих ресурсов является центральное устройство машины, время которого делится между задачами. Однако режимом работы с разделением времени называется несколько иная вещь. На многопрограммных машинах появились новые виды внешних устройств - выносные пульты. Такой пульт оборудован телетайпом, клавиатура которого позволяет одновременно с печатью вводить информацию в машину. На то же печатающее устройство может выводиться информация из машины. Кроме телетайпа,на пульте могут быть читающие и перфорирующие устройства, обычно для бумажной перфоленты, а также экран с лучевой трубкой, на котором по сигналам от машины может высвечиваться разнообразная информация: текст, графики и другие изображения.

С помощью таких пультов одновременно несколько человек могут быть подключены к машине. Сигналы с пультов поступают по меркам современных машин крайне медленно. На прием и передачу данных с нескольких десятков, а иногда и сотен пультов, машина расходует незначительнук часть своего времени. Но поступающая с пультов информация - программа решения некоторой задачи - должна обрабатываться. Теоретически для обработки этой информации могут потребоваться все или почти все ресурсы машины. Практически все требуемые ресурсы, в первую очередь, оперативная память и центральное устройство, выделяются для выполнения программ, вводимых с пультов, но на очень короткое

время - небольшими порциями (квантами). Отсюда и название - система разделения времени. Человек, работающий за пультом, как бы получает в свое распоряжение однопрограммную машину с большой памятью, разнообразными внешними устройствами и неплохой скоростью работы. То, что эта скорость много меньше истинного быстродействия машины, обслуживающей пульты, человек не ощущает. О возможностях, которые достигаются благодаря разделению времени, речь пойдет в следующем разделе. Рассказ о третьем поколении завершим на средствах математического обеспечения, непосредственно не связанных с мультипрограммированием.

Обилие алгоритмических языков, появившихся в период второго поколения, во многом объясняется модой. Но имелись и более существенные причины для создания новых языков и вариантов старых. Ни один из предложенных языков не позволял удобно описывать все возникавшие задачи. АЛГОЛ 60, например, несмотря на все богатство средств, предоставляемых программисту, был совершенно не приспособлен, даже с позднейшими добавлениями, для описания всего многообразия процедур ввода-вывода. Аппарат рекурсивных процедур был сложен в реализации и все же недостаточен и неудобен для описания процессов обработки символьной информации.

Третье поколение поставило на повестку дня создание действительно универсального алгоритмического языка. Одной из попыток такого рода был язык ПЛ/I, предложенный опять-таки фирмой ИБМ. В его основе лежали языки ФОРТРАН и КОБОЛ, ряд изобразительных средств и понятий был почерпнут из АЛГОЛа и других языков, в частности, языков для обработки символьной информации. Однако язык ПЛ/I, особенно первые

его варианты (он проделал в течение нескольких лет заметную эволюцию), не представлял собой органического целого. Различные вложенные в него возможности выпирали во все стороны острыми углами. В последних версиях все это лучше утряслось и подогналось одно к другому, углы закруглились, но язык продолжает оставлять впечатление чего-то очень громоздкого. Из числа достоинств его хочется отметить аппарат процедур ввода-вывода, достаточно богатый, гибкий и стройный.

Другое направление на пути создания универсального алгоритмического языка связано с деятельностью Рабочей группы 2.I Международной федерации по обработке информации (ИФИП), взявшей на себя ответственность за дальнейшее развитие АЛГОЛа. Члены группы пошли по линии обобщения и углубления основных понятий в области обработки информации. Они стремились свести число этих понятий к разумному минимуму и добиться высокой изобразительной силы языка, обеспечив свободу сочетания и взаимодействия этих понятий между собой. Эта цель в значительной степени достигнута в представленном сейчас на широкое обсуждение проекте языка АЛГОЛ 68. Члены Рабочей группы внесли и частично опубликовали много предложений, содержащих интересные идеи. Большинство этих идей в той или иной степени отражены в проекте.

Описание языка еще более формализовано, что способствует большей точности и облегчает создание трансляторов, но и затрудняет понимание языка. Много внимания в проекте уделено эффективности языка - упрощению трансляции и повышению качества машинных программ, производимых транслятором. Предусмотрена возможность независимой трансляции отдельных частей программы. Некоторые искусственные и малоупотребительные понятия АЛГОЛа 60 не вошли в проект нового языка, остав-

шиеся подверглись в ряде случаев глубокой переработке, обогатившей их содержание.

Одним из наиболее существенных нововведений являются имена - аналоги понятия адреса ячейки вычислительной машины. Программист получил возможность манипулировать не только с содержимым ячеек, но и с их именами. Обогатилось и понятие значения (аналог содержимого ячеек). Кроме простых значений и их массивов, в качестве значений могут выступать имена, подпрограммы и структуры - составные значения, элементы которых могут быть разных видов, тогда как элементы массива должны быть все одного и того же, хотя и произвольного вида.

Для описания представления значений на внешних носителях информации используются форматы. С их помощью можно описывать практически любые процедуры ввода и вывода. Программист может вводить новые знаки операций и описывать связываемые с ними действия.

Предусмотрен аппарат для выделения из массивов их частей - подмассивов той же или меньшей размерности. Массивы, подмассивы и структуры могут рассматриваться как единые операнды при определении новых операций. Таким образом, в язык может быть, например, введен аппарат операций над векторами и матрицами, операций над строками и т.п.

Есть возможность указать, что некоторые ветви вычислительного процесса могут выполняться параллельно, предусмотрены средства для синхронизации таких параллельных ветвей в заданных программистом спорных точках.

Конечно, пройдет время, и АЛГОЛ 68 перестанет быть универсальным языком, но сейчас кажется, что этот язык способен обслужить почти все нужды программистов.

5. Задачи, плохо решаемые машиной

Обычно утверждают, что существует довольно обширный класс задач, которые современные машины решают хуже, чем человек. К ним относят так называемые „думательные" задачи. Впрочем, точного определения этого класса задач не существует. В качестве типичного представителя задач этого класса называлась, например, игра в шахматы. Однако принципиальные основы алгоритма этой игры были сформулированы Шенноном еще в 1950 г. Сейчас составлены программы, довольно прилично играющие в шахматы, и, если мода на них не пройдет, то через несколько лет гроссмейстеру нелегко будет состязаться с машиной (хотя бы потому, что технические возможности машин продолжают довольно быстро нарастать).

Другой пример - сама автоматизация программирования. До сих пор раздаются еще голоса, что это - никому не нужная затея, что человек всегда напишет на языке машины более эффективную программу, чем любой транслятор. Но это верно, если только речь идет о хорошем программисте, среднем трансляторе и сложной программе. Подавляющее же большинство задач могут быть решены на машине с помощью довольно шаблонных приемов программирования, которыми хороший транслятор (например, ТА-1М АЛЬФА-транслятор для машин типа М-20) владеет не хуже, чем рядовой программист. Чисто техническую работу по программированию таких задач транслятор выполнит быстрее и аккуратнее, чем человек.

Третий пример - распознавание образов. Недавно в журнале "Знание - сила" была напечатана любопытная статья, где выяснялось, чем кошка отличается от собаки. Приводился воображаемый диалог мамы с любознательным ребенком, легко опро-

вергавшим такие критерии, как наличие усов ("А если их остричь - кошка станет собакой?"), форма зрачка ("Значит, спящую кошку не отличить от собаки?"), умение прятать когти, размер животного и т.п. В конце концов, автор кладет в основу такой критерий: кошка - подкарауливающий хищник, а собака - преследующий, откуда вытекают все или большинство вторичных признаков.

Ясно, что этот критерий предполагает использование целого ряда сложных и отвлеченных понятий : хищник, охота, засада, погоня и т.п. Человек располагает этими понятиями с довольно юных лет. Что касается машины, то не очень понятно даже, при каких условиях можно считать, что она владеет тем или иным понятием. По-видимому, некоторые понятия являются для машины "врожденными". Например, в силу самой конструкции машины ей доступны понятия: адрес, сумматор, извлечение числа из ячейки памяти, умножение и т.п. Машина, снабженная транслятором с АЛГОЛа, способна оперировать понятиями: арифметическое выражение, условный переход, функция и пр. Но можно и оспорить все эти утверждения, сказав, что машина ни одним из этих понятий не владеет, поскольку она не сама их осмыслила, а получила в готовом виде от человека. Во всяком случае, пока еще никто не сформулировал идеи программы, которая позволила бы научить машину распознавать любой образ, овладеть любым понятием из любой области человеческого знания или хотя бы любым понятием физики.

Но нет и убедительных доказательств того, что такая программа не может быть создана. Когда же речь заходит о распознавании определенного класса образов, например, фотографий следов частиц в пузырьковой камере, то такая задача не кажется неразрешимой. Человек может достаточно отчетливо

сформулировать ряд признаков, по которым он анализирует эти фотографии, и заставить машину прибегать к тем же признакам. Задача будет решаться успешно до тех пор, пока не встретится фотография со слишком многими помехами или пока не будет зафиксирован след частицы с совершенно новыми свойствами. Примем за постулат, что такой след рано или поздно должен появиться. Недостаточно помехоустойчивая программа будет отбраковывать слишком много полезных фотографий, поддающихся расшифровке ценою дополнительных усилий. Черезчур помехоустойчивая программа совершит более опасную ошибку - она воспримет новое явление, как искаженный вариант известного.

Физический эксперимент в конечном счете нацелен на открытие новых явлений. Но говорят, и не без оснований, что и человек способен воспринять лишь такое новое явление, к которому он подготовлен, которое ожидает встретить. Тем более это справедливо по отношению к программе. Программа, настроенная на ожидание нового явления,- это программа, обращающаяся к помощи человека всякий раз, когда она не может уверенно отождествить предъявленный ей для обработки материал. Итак, в хорошей программе распознавания образов должна быть заложена разумная мера сомнений.

Как реагировать на обнаруженное явление - должен определить человек. Его решение может быть различным. Возможно оно будет относиться только к данному конкретному случаю: отбросить предъявленный материал как лишенный смысла, или внести в него такое-то уточнение и повторить его анализ, или ограничиться проделанной работой и перейти к следующему случаю. Но, возможно, человек признает встретившуюся ситуацию достаточно типичной, тогда он должен иметь возможность изменить программу, чтобы в следующий раз подобная ситуация не вызывала у машины затруднений.

Сам рассмотренный пример также достаточно типичен. Он свидетельствует о необходимости ряда средств общения человека с машиной. Программа должна уметь сигнализировать человеку о невозможности продолжать работу обычным образом. При этом требуется экономно и точно сообщить о характере встретившегося затруднения и сопровождающих его обстоятельствах - все это, разумеется, в терминах уже знакомых понятий. Далее, человек должен располагать средствами отдачи машине приказа о том или ином характере дальнейших действий, снабжения ее дополнительной информацией или дополнения ранее составленной программы новыми частями.

Задача осложняется тем, что человек и машина разговаривают на разных языках. Даже числовой материал человек обычно воспринимает в десятичной, а машина - в двоичной системе. Еще сложнее с программой, которую человек написал и представляет себе в терминах какого-либо алгоритмического языка: например: АЛГОЛа или ФОРТРАНа, а машина исполняет в виде машинных команд. Геометрическую информацию, особенно если речь идет о двумерных фигурах и образах, человек воспринимает особенно легко, для машины же опять требуется преобразование этой информации в форму последовательностей знаков. Обратный переход здесь особенно труден.

Чтобы эффективный обмен между человеком и машиной был вообще возможен, необходимо соблюсти ряд требований. Информация, исходящая от человека или даже поступающая из других источников, но проходящая его контроль, должна храниться в машине в том виде, в каком ее воспринимает человек, может быть с минимально-необходимой перекодировкой, допускающей легкое и однозначное обращение.

Например, тексты, вводимые в машину, должны храниться в буквенно кодированном виде. Теоретически возможно и даже выгодно применять другие способы кодировки текста, позволяющие значительно уменьшить объем памяти, занятой кодом. Но практически это делать нецелесообразно, так как простое исправление описки выливается при этом в довольно сложную задачу.

С другой стороны, необходимо хранить информацию и в удобном для машины виде, что особенно очевидно в случае, когда эта информация - программа работы машины. Можно было бы, конечно, хранить программу, скажем, только на АЛГОЛе и каждый раз, когда потребуется выполнить какой-то участок этой программы, осуществлять его перевод на язык машины. Такой способ использования программ, написанный на алгоритмических (не машинных) языках, называется интерпретацией. В общем случае он приводит к большим потерям времени при выполнении программы, так как перевод текста программы на машинный язык - работа обычно более трудоемкая, чем выполнение полученной программы. Правда, существует категория алгоритмических языков, приближающихся по типу к машинным языкам. Программа на таком языке имеет вид последовательности команд, причем каждая команда требует выполнения более или менее стандартных действий. Для таких команд могут быть составлены соответствующие машинные программы, и тогда перевод команды на язык машины будет означать просто обращение к соответствующей программе. Затраты времени на перевод в таком смысле относительно невелики, и интерпретация становится практически выгодной, так как сокращает размер хранимой программы. Но, по существу, и здесь мы находим подтверждение сделанного выше утверждения - в машине должны храниться как исходная программа на входном языке, так и ее машинное представление.

В данном случае это представление состоит из набора подпрограмм, интерпретирующих команды входного языка, и системы интерпретации, организующей обращение к этим подпрограммам. Заметим, что интерпретируемые языки, о которых здесь шла речь, часто используются как промежуточные между проблемно-ориентированными алгоритмическими языками и машинными языками. Программа, написанная на проблемно-ориентированном языке, переводится(один раз) на промежуточный язык (говорят, что осуществляется компиляция последовательности команд промежуточного языка), а затем результат перевода интерпретируется. Здесь уже каждая команда переводится (т.е. заменяется соответствующей подпрограммой) многократно - столько раз, сколько это требуется по ходу выполнения программы.

Кроме информации, относящейся непосредственно к решаемой задаче, как человеку, так и машине может понадобиться справочная информация, например, некоторые результаты ранее решавшихся задач или ранее проведенных экспериментов. Сюда же относятся различного рода словари, таблицы, стандарты и т.п.

6. Архивы

Итак, для эффективной совместной работы человека и машины, особенно при решении сложных задач, в машине должно быть сосредоточено большое количество разнородной информации, образующей так называемый архив. Архив состоит из информационных массивов, каждый из которых может быть массивом чисел в двоичном или десятичном представлении, массивом машинных команд, программой или частью программы на каком-либо алгоритмическом языке, текстом на обычном языке, более или менее сложно организованной таблицей и т.д.

Если к информационному массиву обращается машина, то в качестве его наименования естественно и удобно использовать адрес, т.е. краткое обозначение, единственное назначение которого - обеспечить быстрый поиск места расположения этого массива в памяти машины. Для человека такие кодовые обозначения, как правило, неудобны, так как они не вызывают у него никаких ассоциаций. Человеку естественнее использовать мнемонические наименования, подобно идентификаторам в АЛГОЛе, и даже более сложные. Желательно, чтобы "человеческое" наименование информационного массива само несло в себе информацию о содержании и назначении массива.

Отсюда следует, что архив должен быть снабжен каталогом, позволяющим по мнемоническому наименованию того или иного массива найти его "машинное" наименование, т.е. адрес места массива в памяти. Другие сведения о массиве - его длину, характер содержащейся в нем информации, способ кодировки и т.п. целесообразно хранить не в каталоге, а вместе с массивом.

Если на машине используется несколько систем программирования (а это при существующем вавилонском смешении языков вполне реально), то в принципе каждая система может располагать своим архивом. Но это неудобно и неэкономно, так как архив может содержать информационные массивы справочного характера, иметь доступ к которым желательно приверженцам любой системы программирования. Поэтому архив должен быть органической частью не какой-либо отдельной системы программирования, а операционной системы, под которой мы в данном случае понимаем программу, координирующую работу всех систем программирования, реализованных на данной машине.

Впрочем, отдельные информационные массивы, например, программы, написанные на определенном алгоритмическом языке, могут представлять интерес лишь для пользователей определенной системы программирования. Более того, среди пользователей могут существовать группы, решающие совместными усилиями одну и ту же задачу или несколько родственных задач (например, задач обработки спектрометрических экспериментов в ядерной физике). У таких групп пользователей появляются информационные массивы, в которых другие пользователи не заинтересованы. Целесообразно (по причинам, которые станут ясны позже) объединить эти массивы в групповой архив данной группы пользователей. В пределе группа может состоять из одного пользователя, в распоряжении которого находится индивидуальный архив. Упомянем коротко еще об одном виде частного архива - архиве отдельной задачи. Существуют задачи, которые невозможно решить за один прием, т.е. за один выход на машину. Пожалуй, даже большинство задач относятся к этому классу, но одни задачи нетрудно каждый раз вводить в машину заново, а для других это совершенно неразумно. Информация, которая остается в машине от одного запуска в решение такой задачи до следующего, и образует архив задачи.

Все частные, т.е. групповые и индивидуальные архивы и архивы задач, входят в общий архив как его части, обладающие известной автономией, но в то же время имеющие много общих свойств и обслуживаемые единой системой архивного хозяйства. Чтобы уяснить разницу между разновидностями частных архивов, рассмотрим типичные операции над информационными массивами (см. таблицу 2).

Таблица 2.

№№ пп	Операция	Характер массива
I.	Включить массив в архив	Любой массив
2.	Исключить массив из архива	" – "
3.	Присвоить массиву новое наименование	" – "
4.	Переписать массив на новое место в памяти	" – "
5.	Отпечатать содержимое массива	" – "
6.	Передать массив в полное распоряжение задачи	" – "
7.	Передать массив в распоряжение задачи без права изменять его	" – "
8.	Выбрать из массива элементы (элемент) с данным свойством	Таблица, словарь и т.п.
9.	Выполнить массив (возможно, при заданных значениях параметров)	Программа (подпрограмма) на машинном языке.
IO.	Транслировать массив	Программа на алгоритмическом языке.
II.	Перевести массив в другую систему счисления.	Массив чисел

Операции I и 2 составляют основу системы комплектации архива. При комплектации необходимо решать ряд важных задач: защитить массивы, включенные в архив, от непреднамеренной порчи, обеспечить быстроту и удобство обращения к массивам, когда это обращение допустимо, предотвратить чрезмерное раз-

растание архива. Массивы общего пользования должны быть доступны всем пользователям для извлечения хранящейся в них информации. Включение же новых таких массивов в архив и, особенно, исключение и изменение информации, содержащейся в этих массивах, должны быть, напротив, максимально ограничены. Всякая ошибочная операция такого рода, в худшем случае, приводит к утрате ценной и невосполнимой информации, в лучшем - к потерям времени на восстановление информации. Изменение данных, хранящихся в архиве, может привести также к тому, что ранее отлаженная программа перестает работать вообще или начинает выдавать неверные результаты. Это может произойти, например, если справочные данные, относящиеся к какому-то промежутку времени, заменены аналогичными данными для более позднего времени, а в программе требуется произвести расчет для момента времени, не попадающего в новый диапазон. Другой пример - архивная подпрограмма, реализующая некоторый алгоритм, заменена подпрограммой, выполняющей другой, более быстрый алгоритм, но с меньшей областью применимости.

По мере перехода от общего архива к более частным на доступ к массивам должны вводиться более жесткие ограничения, а ограничения на переработку массивов должны постепенно сниматься. К массивам из архива задачи доступ может иметь только эта задача. Зато она может производить переработку массивов своего архива практически без всяких ограничений (кроме, может быть, ограничений на рост архива). Групповые и индивидуальные архивы должны находиться на промежуточной ступени по жесткости этих ограничений.

Операции 3 и 4 связаны с ведением каталога и имеют своей целью ускорение и упрощение доступа к материалам архива. Печать информационного массива (операция 5) предназначена для контроля за содержанием массива.

Операции 6 и 7 служат для того, чтобы программа могла производить с массивом нужную ей работу, причем операция 7 типична для общего архива, операция 6 может использоваться в частных архивах с целью снятия стандартных ограничений на переработку массива. После выполнения этих операций дальнейшая работа с массивом осуществляется программой без участия системы ведения архива.

Операции 8-II служат примером операций над массивами специального вида.

Включение в операционную систему способности производить над архивными массивами указанные (и, возможно, некоторые другие) операции, значительно облегчает задачу построения любых систем программирования. Особенно же упрощается организация связи человека с машиной. Если к тому же все программы математического обеспечения (стандартные подпрограммы, компилирующие и интерпретирующие программы, трансляторы и т.д.) сами включены в архив на правах общего пользования, то программист, работающий за выносным пультом, получает в свое распоряжение все это богатство и не ощущает никаких неудобств от своей удаленности от машины. Может быть, только замедляется ввод и вывод данных. Физическая скорость ввода-вывода снижается определенно, но для пользователя важна не она, а скорость поступления материалов в обработку и получение результатов. Эта скорость может даже увеличиваться.

Рукопись поступила в издательский отдел
5 марта 1969 года.