№ 11 (204) март 1999

еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИК

Электронные версии газеты “Первое сентября” и приложений http://www.1september.ru

Интернет для начинающих

А.А. ДУВАНОВ,
duvanov@robotland.users.botik.ru

Продолжение. См. № 36, 40, 44, 46/98; 2, 5, 7, 9/99

1.9. А где мне взять такую песню? (Поиск информации)

Вася. Архивы Интернета, конечно, огромны, но тем сложнее найти в них нужный файл. Если бы у меня был список всех ftp-серверов и названия всех файлов, которые они содержат, тогда бы я легко нашел то, что мне нужно.

Папа. Такие списки тебе не очень помогут: они слишком длинны и, главное, очень быстро меняются. Одни файлы в архивы добавляются, другие удаляются, появляются новые архивы, а старые “закрываются”. Все это происходит в сети каждую минуту.

Вася. Интернет все больше напоминает мне переславльский базар: каждый участник выставляет на продажу то, что пожелает, и закрывает свою лавочку тогда, когда ему захочется!

Папа. Сравнение поучительное! Интернет не поддается планированию: так он изначально был задуман. Все, что происходит в сети, делается по желанию каждого отдельного пользователя в соответствии с его вкусами и возможностями.

Вася. Значит, найти в сети файл — задача невыполнимая? Зачем же тогда все это информационное богатство, если реально им нельзя воспользоваться?

Папа. Не все так мрачно. Для поиска в сети можно воспользоваться услугами какого-нибудь archie(арчи)-сервера. Эти серверы хранят списки с указателями файлов, находящихся в ftp-архивах. Доступ к archie-серверу по электронной почте выполняется при помощи почтового archie-робота, который работает так же, как и ftp-робот.

Вася. Иными словами, archie-сервер не умеет работать с электронной почтой, и это делает за него archie-робот.

Папа. Верно!

Вася. Вероятно, нужно знать адрес archie-сервера и адрес archie-робота для того, чтобы можно было начать поиск.

Папа. Это так, но ситуация упрощается тем, что каждый archie-сервер имеет своего собственного archie-робота. Почтовый адрес робота складывается из адреса сервера добавлением слова “archie” и символа “@”. Вот, например, мне известен адрес хорошего английского archie-сервера: **archie.hensa.ac.uk**.

Вася. Тогда адрес его почтового исполнителя запишется как **archie@archie.hensa.ac.uk**.

Папа. Видишь, как просто! Как ты думаешь, какую команду нужно послать самой первой?

Вася. Учитывая опыт работы с ftp-исполнителями, думаю, команду **`Help`**.

Папа. Это самая главная команда всех почтовых роботов!

Вася. Программа для archie составляется так же, как для ftp?

Продолжение на с. 15

ДОРОГИЕ КОЛЛЕГИ!

В прошлом номере были опубликованы результаты розыгрыша анкет из первого номера этого года. Теперь мы хотим предложить вам новую анкету, которую планируем сделать регулярной. Пожалуйста, оцените первые десять номеров этого года, используя привычную пятибалльную систему, по трем критериям:

1. Насколько данный номер был полезен вам в работе?
2. Насколько данный номер был интересен вам для чтения?
3. Насколько удобно были расположены материалы в данном номере?

Например: 1. 4 2. 5 3. 4.

Как всегда, среди приславших анкеты мы разыграем призы. Письма, пожалуйста, присылайте до 31.04.99.

Спасибо!

Редакция “Информатики”

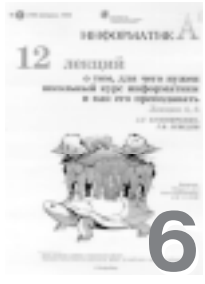
1. __ 2. __ 3. __

1. __ 2. __ 3. __

1. __ 2. __ 3. __

1. __ 2. __ 3. __

1. __ 2. __ 3. __



1. __ 2. __ 3. __

1. __ 2. __ 3. __

1. __ 2. __ 3. __

1. __ 2. __ 3. __

1. __ 2. __ 3. __

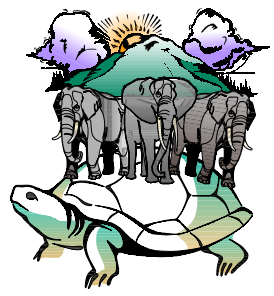
Мы будем благодарны издательствам и компьютерным фирмам, которые пожелают предоставить призы нашим читателям.

НАШИ ДЕТИ БУДУТ ЖИТЬ В XXI ВЕКЕ

12 лекций о том, для чего нужен школьный курс информатики и как его преподавать

Лекция 10

А.Г. КУШНИРЕНКО, Г.В. ЛЕБЕДЕВ



Об основных понятиях, идеях и целях школьного курса информатики “по учебнику” *А.Г. Кушниренко, Г.В. Лебедева, Р.А. Свореня* “Основы информатики и вычислительной техники” (М.: Просвещение, 1990, 1991, 1993, 1996). Дается также ряд практических советов, предлагаются соответствующие методические приемы.

Авторы надеются, что материал окажется полезным для учителей и методистов, использующих указанный учебник, а также для тех, кто желает сравнить разные подходы к преподаванию школьного курса информатики или разработать свой собственный курс.

Продолжение следует

История компьютеров

А.Г. ЛЕОНОВ, О.В. ЧЕТВЕРГОВА

Продолжение. См. № 35, 41, 48/98

4. Четвертое поколение. Большие и сверхбольшие интегральные схемы. Микропроцессоры. Микрокомпьютеры. Микропроцессоры Motorola и Intel

1. Прорыв в технологии производства компьютеров

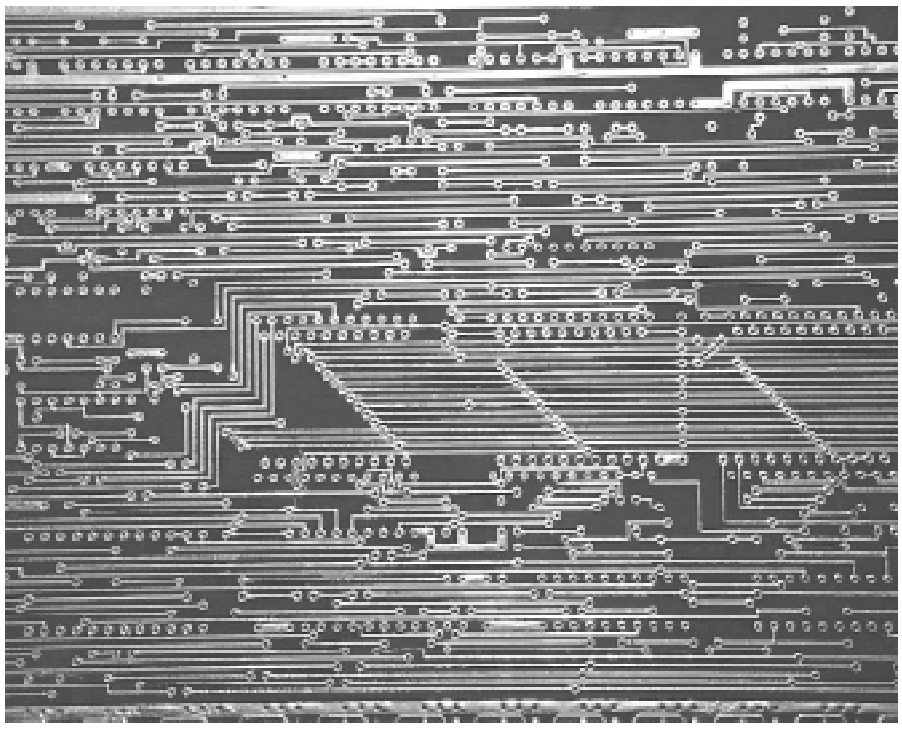
Начиная с 60-х годов в качестве элементной базы при производстве компьютеров стали использоваться преимущественно интегральные схемы (микросхемы). Долог был их путь от чертежной доски инженера до сборочного цеха. Они прошли этапы развития от схем, содержащих всего несколько транзисторов, до интегральных схем с сотнями тысяч и более (!) транзисторов, получивших название сверхбольших интегральных схем (или СБИС). Влияние СБИС-технологии на развитие ЭВМ трудно переоценить: с ней впервые появилась возможность производить центральные процессоры, память и прочие устройства компьютера в виде отдельных СБИС, что, в свою очередь, позволило наладить массовый выпуск дешевых (в перспективе доступных практически каждому) компьютеров. Помимо нового класса ЭВМ — недорогих персональных компьютеров, были созданы высокоскоростные параллельные вычислительные системы, в которых центральный процессор был заменен тысячами процессоров. Когда говорят "четвертое поколение ЭВМ", обычно имеют в виду машины, использующие технологии, основанные на СБИС.

2. Интегральные схемы

Основные элементы ЭВМ — транзисторы сотнями и тысячами содержатся в интегральных схемах, называемых чипами (*chip* — щепка, обломок). Чип — это небольшой прямоугольник полупроводника. Изготовлены чипы, как правило, из кремния, упакованы в изолирующий материал и снаружи имеют металлические контакты. Видов интегральных схем очень много. В ДИПах (DIP — *Dual-In-line Package* — корпус с двухрядным расположением выводов) контакты образуют два параллельных ряда, при этом расстояние между двумя ближайшими контактами строго выдерживается равным 2,54 мм. Иногда выводы размещаются только с одной стороны интегральной схемы. Такая схема носит название СИП (SIP — *Single-In-line Package* — корпус с расположением выводов в один ряд). Если чип сложный (с сотней или бóльшим количеством контактов), то конфигурация его называется ПГА (PGA — *Pin Grid Array* — корпус с матрицей штырьковых выводов).

В современном компьютере объединено множество интегральных схем, связанных между собой. В компьютере также имеются источник питания и другие элементы (резисторы, конденсаторы и пр.).

Интегральные схемы можно соединять, используя, например, толстые пучки проводов, однако гораздо дешевле (и проще) другая технология. На тонком листе изолятора (тут может употребляться, к примеру, стекловолокно) располагаются интегральные схемы и другие мелкие элементы. Контакты через подготовленные (просверленные) отверстия выходят на другую сторону листа (его называют платой). Плата также выполняет функцию механического крепежа. На обратной стороне листа-платы наносится рисунок из токопроводящего материала (им может быть, например, медь), и этот рисунок является заменой соединительных проводов. Для промышленности изготовить такую печатную плату (PCB — *Printed Circuit Board*) проще, чем соединять схемы проводами, да и обходится она намного дешевле. Подчас произвести все соединения на плате без пересечений довольно трудно. В подобных случаях применяется многослойный монтаж, как бы из нескольких плат. Чем больше интегральных схем на плате, тем она сложнее, и, наоборот, чем меньше элементов, тем плата проще (в ней меньше слоев).



При работе интегральные схемы излучают тепло. Поэтому для соблюдения температурного режима (чтобы микросхема не перегрелась и не сломалась) может потребоваться установка на интегральную схему металлического радиатора (или даже вентилятора) для рассеивания тепла (для охлаждения микросхемы).

3. Типы интегральных схем

Немного научных терминов.

Основными технологиями, используемыми в интегральных схемах, являются биполярная технология и МОП-технология (материалы Металл — Оксид — Полупроводник). Так же называются и интегральные схемы. В интегральных схемах каждого из этих двух видов применяются свои транзисторы в качестве базовых элементов (переключателей).

Как известно из курса физики, транзистор представляет собой трехслойную структуру из полупроводниковых материалов (внешние слои — эмиттер и коллектор, средний слой — база). Обычные (биполярные) транзисторы состоят из полупроводников двух типов. Если использовать такой транзистор в качестве переключателя, то большой ток, протекающий через коллектор, можно считать двоичной 1, малый — 0. Биполярные транзисторы за очень короткое время могут быть переведены из одного состояния в другое (из состояния 0 в состояние 1 или из состояния 1 в состояние 0). Они являются основой TTL-схем (*Transistor-Transistor Logic* — транзисторно-транзисторная логика). Поэтому у так называемых TTL-схем высокая скорость переключения.

МОП-транзисторы, в отличие от биполярных, очень экономичны (они практически не потребляют питающего их входного тока). Эти устройства, в свою очередь, являются основой МОП-схем.

Существуют еще и КМОП-схемы (комплементарные МОП-схемы), построенные на использовании пары комплементарных (мы не уточняем, что это) МОП-транзисторов. Такая пара почти не потребляет энергии, пока находится в одном из двух устойчивых состояний, которые обозначают соответственно 0 и 1. Уменьшая размеры МОП-устройств, можно увеличить скорость срабатывания (переключения) и плотность упаковки (число транзисторов в схеме). С помощью же КМОП-технологии удается существенно повысить производительность и уменьшить энергопотребление схем.

Большинство интегральных схем изготавливается из кремния, однако иногда используется другой материал — арсенид галлия — сложный полупроводник на основе соединения мышьяка с галлием. Производить из этого материала интегральные схемы труднее, но зато работают они примерно в пять раз быстрее кремниевых.

Интегральные схемы также делятся на различные категории в зависимости от плотности схемы. Определяется она количеством транзисторов, включенных в чип, или количеством логических вентилей. Вентиль представляет собой электрическую схему, преобразующую входные сигналы (таких сигналов может быть два и более) в один выходной (например, логическое И, логическое ИЛИ и пр.). Обычно для составления логического вентиля требуется примерно пять транзисторов. Самые маленькие по объему, первые интегральные схемы, содержащие около 10 вентилей, назывались малыми интегральными схемами (SSI — *Small-Scale Integration* — малая интеграция). Схемы с количеством вентилей от 10 до 100 получили название средних интегральных схем (MSI — *Medium-Scale Integration* — средняя интеграция). Что касается больших интегральных схем (LSI — *Large-Scale Integration* — большая интеграция), то они содержат сотни и тысячи вентилей. Ну а сверхбольшими интегральными схемами (VLSI — *Very-Large-Scale Integration* — очень большая интеграция) называют схемы, подобные чипам емкостью 1 Мбайт, появившимся только в 1986 году. В каждом таком чипе более миллиона МОП-транзисторов.

Поскольку для изготовления интегральных схем используется полностью автоматизированное производство, стоимость их изготовления невелика, а объемы производства могут быть весьма внушительными. Массовый выпуск больших интегральных схем начался на пороге 70-х годов. Большие интегральные схемы стали основами памяти ЭВМ и карманных калькуляторов.

Впервые появилась возможность построить процессор (а может быть, и целый компьютер) в одном-единственном чипе. Компьютеры (и программируемые роботы-контроллеры), процессор которых состоит из одной или нескольких интегральных схем, стали называться микрокомпьютерами.

В связи с низкими производственными затратами стоимость микропроцессоров и микрокомпьютеров достаточно мала. Это сделало вычислительную технику доступной широким массам пользователей, причем небольшие размеры микрокомпьютеров позволяют применять их в тех областях, где ЭВМ предыдущих поколений не могли использоваться (автомобили, радио, бытовая техника, телефоны и пр.).

4. Проектирование интегральных схем

Интегральная технология дала жизнь не только новому массовому производству — сборке компьютеров из стандартных чипов и микропроцессоров, но и быст-

рому и недорогому проектированию и производству нестандартных (заказных, специализированных) чипов для компьютеров. Это стало возможно вследствие применения технологии автоматизированного проектирования (*Computer-Aided Design* — CAD) интегральных схем для полностью автоматизированного производства. Такое проектирование, как правило, имеет три стадии.

1. Задание интегральной схемы.

Проектируемая схема на рабочем месте проектировщика (CAD-компьютер) создается (рисуется на экране) и сохраняется в виде файла в некотором стандартном формате. Схема задается поэлементно, причем она может быть как простой, так и представлять собой схему сложного микропроцессора. Иногда элементы схемы задаются самостоятельно инженером-проектировщиком, но в целом ряде случаев целесообразно выбирать стандартные элементы из базы данных (причем интегральная схема может быть целиком взята из файла с ее прообразом и откорректирована).

2. Верификация интегральной схемы.

Заданная схема тестируется при помощи специализированного программного обеспечения для выяснения ее соответствия проекту. На этом этапе разрешается вносить поправки в проект, улучшать или полностью менять части схемы (с последующей ее проверкой). Вручную, без использования CAD-компьютера, делать такие вещи практически невозможно. Результат второго этапа — файл, в котором интегральная схема представлена поэлементно в некотором также стандартном формате.

3. Подготовка производства интегральной схемы.

База данных CAD-компьютера содержит описания этапов производства интегральных схем. Файл, в котором хранится описание проектируемой схемы, преобразуется в специальный "производственный" формат, полностью описывающий автоматизированный процесс производства интегральной схемы.

5. Производство интегральных схем

С появлением так называемой планарной технологии стало возможно использовать групповой метод изготовления интегральных схем, при котором на каждом этапе производства обрабатывают всю кремниевую пластину (а иногда и несколько пластин). Каждая такая пластина содержит сотни кристаллов (то есть интегральных схем или их частей). Для каждой технологической операции необходимо обеспечить выборочность воздействия на поверхность пластины. На пластине перед технологическими операциями легирования, травления, окисления и т.д. создается защитный слой (маска), где обнажаются только те участки поверхности, которые действительно должны быть подвергнуты обработке в процессе данной операции.

Интегральная схема после проектирования представлена в виде плоского многослойного рисунка, который составлен из геометрических фигур, изображающих отверстия каждой из масок. Это называется топологическим чертежом, или топологией. Основная задача при производстве интегральных схем и состоит в переносе топологического чертежа на пластины.

Площадь большинства интегральных схем не превышает 1 кв. см, так что на пластине обычно размещается 200—300 интегральных схем.

Основные этапы производства интегральных схем можно разделить на несколько групп (по технологическому признаку):

1. ***Механическая обработка:***
резка слитка на пластины;
шлифовка пластин;
механическая полировка пластин;
разделение пластин на интегральные схемы.

2. ***Химическая обработка:***
мойка (очистка) поверхности пластин;
травление поверхности.

3. ***Нанесение диэлектрических пленок:***
окисление поверхности;
осаждение диэлектрика;
ионное внедрение.

4. ***Нанесение металлических пленок.***

5. ***Гравировка диэлектрических и металлических пленок, пластины.***

6. ***Формирование выводов и упаковка интегральной схемы в корпус.***

6. Производство микропроцессоров

Первый настоящий микропроцессор Intel-4004 выпустила фирма Intel в 1971 году. Вообще микропроцессор 4004 был разработан для использования в калькуляторах в составе четырех чипов набора MSC-4 (большие интегральные МОП-схемы), но продавался отдельно как программируемый контроллер. Размер его машинного слова — 4 бита, однако инструкции имели длину 8 бит.

Данные хранились отдельно в памяти объемом 1 Кбайт, программа находилась в другой памяти объемом 4 Кбайта. Микропроцессор был оснащен 12-битным счетчиком команд (PC), стеком глубиной 4 (адреса) для выполнения инструкций вызова подпрограмм (`CALL`) и возврата из подпрограммы (`RET`), шестнадцатью 4-битными регистрами (из которых могли образовываться восемь 8-битных регистров). Набор команд Intel-4004 состоял из 46 инструкций.

Скромные возможности микропроцессора Intel-4004 объясняются производственными ограничениями, которые были связаны с плотностью расположения элементов в интегральных схемах, выпускаемых в 70-х годах (размер чипа микропроцессора Intel-4004 — 1 кв. см).

После MSC-4 появилось большое количество микропроцессоров различных производителей, которые использовали МОП- и КМОП-технологию.

С увеличением плотности интегральных схем возрастала и мощь микропроцессоров: размер машинного слова увеличился сначала с 4 до 8, затем до 16, а к середине 80-х годов — до 32 бит. Расширялся набор команд, несмотря на меньшее по сравнению с большими процессорами число инструкций (микропроцессоры, например, не имели команд для вычислений с плавающей точкой); повышалась производительность (количество выполняемых операций в секунду). Такие устройства уже вполне подходили для управления процессом впрыска бензиновых автомобильных двигателей и для микроволновых печей.

Позднее микропроцессоры стали постепенно приобретать черты процессоров больших ЭВМ.

С увеличением выпуска микропроцессоров де-факто появились индустриальные стандарты микропроцессоров и других многочисленных чипов в ЭВМ (включая ОЗУ, ПЗУ, устройства управления вводом-выводом и пр.).

7. Z80

Вскоре после того как в 1974 году корпорация Intel выпустила микропроцессор 8080, быстро завоевавший популярность, фирма Zilog создала микропроцессор Z80, который моментально затмил собой 8080, поскольку превосходил его по всем параметрам и вдобавок был с ним полностью совместим.

Это привело к тому, что фирма Intel покинула рынок 8-битных микропроцессоров и перешла к разработке 16-битных (Intel-8086), а затем и 32-битных процессоров (устройства данного типа, известные под маркой Pentium, являются основными микропроцессорами современных компьютеров).

Что же касается Z80, то этот процессор, используемый и сегодня, в 80-е годы произвел настоящий фурор. Спроектирован он был группой инженеров, разработчиков Intel-8080, которые покинули корпорацию Intel и перешли работать в фирму Zilog. В 8-битном микропроцессоре Z80 имелась возможность напрямую обращаться к памяти объемом 64 Кбайта. Было добавлено более 80 новых инструкций (по сравнению с Intel-8080), в частности, операции с 1-, 4-, 8- и 16-битными данными, а также блоковые операции (для пересылки за одну операцию до 64 Кбайт данных или поиска в таком блоке).

Было два набора регистров и предусматривались операции переключения между ними, что упрощало организацию обработки прерываний (не надо было запоминать состояние регистров в программе). Также были добавлены два 8-битных индексных регистра.

Быстродействие микропроцессора определялось особенностями интегральной схемы и при работе задавалось кварцевым генератором-часами. Частота колебаний такого генератора называется тактовой частотой. Первый процессор Z80 имел тактовую частоту 2,5 МГц, более поздняя модель Z80-C — 8 МГц. Самая короткая команда пересылки из одного регистра в другой выполнялась за 4 такта, то есть первый процессор Z80 выполнял более 60 тыс. операций в секунду. Весьма неплохо для малыша.

8. Motorola 601 RISC-микропроцессор

Большинство микропроцессоров относится к стандарту CISC (*Complex Instruction-Set Computing* — микропроцессор с полным набором команд). Такие устройства, в частности, и выпускает корпорация Intel. В 1970 году фирма IBM предложила новую технологию и соответствующий ей новый стандарт процессоров RISC (*Reduced Instruction-Set Computing* — микропроцессор с сокращенным набором команд). Концепция основана на применении в процессоре очень простых команд и простых способов адресации (около 100 базовых команд, большинство из которых являются однотактными). Процессор становится меньше, проще и, соответственно, быстрее. Для успешного его использования требуется специальное программное обеспечение — "умный" компилятор, способный генерировать соответствующий код. Часто RISC-микропроцессоры имеют режим эмуляции процессоров других стандартов (программа "думает", что она исполняется на совсем другом, эмулируемом, процессоре). Это позволяет находить RISC-процессорам более широкое применение, хотя в таком режиме программы выполняются медленнее, чем с помощью обычного CISC-микропроцессора. Для повышения эффективности нужна полная перекомпиляция программ (преобразование двоичного кода CISC-микропроцессора в код RISC-микропроцессора).

В идеале все инструкции RISC-микропроцессора (например, сложение содержимого двух регистров) исполняются за один такт. На практике некоторые инструкции (такие, как умножение и деление) выполняются чуть дольше, а другие инструкции (загрузка содержимого регистра из памяти) могут потребовать даже более двух тактов.

Еще одно отличие RISC-микропроцессоров связано с количеством регистров микропроцессора (обычно у процессоров стандарта RISC в несколько раз больше регистров, чем у CISC-микропроцессоров).

Motorola PowerPC RISC-микропроцессор — один из не очень старых 32-битных микропроцессоров с тактовой частотой 80 МГц.

На микропроцессорном форуме в октябре 1993 года создатели 601-го процессора фирмы IBM и Motorola объявили о новой модели PowerPC — 603. Этот КМОП-чип, будучи в 23 раза экономичнее 601-го процессора, был призван его заменить. Новое устройство содержало 1,6 млн транзисторов и имело тактовую частоту 80 МГц.

Очень скоро Motorola и IBM приступили к разработке следующих моделей серии PowerPC, и уже через несколько месяцев был создан микропроцессор PowerPC 604 с тактовой частотой 100 МГц. Затем появилась модель PowerPC 615, аппаратно совместимая с популярными процессорами семейства 80×86 кор-

Окончание на с. 4

МАТЕРИАЛЫ К УРОКУ

История компьютеров

Окончание. См. с. 2, 3

порации Intel (и составившая изделиям этой фирмы серьезную конкуренцию), а позже 64-битный процессор PowerPC 620.

9. Микрокомпьютеры

Производство микропроцессоров не является самоцелью, микропроцессор — это сердце компьютера. Прогресс в производстве и разработке микропроцессоров — это прогресс в компьютеростроении. В 1982 году Ху Мин Д. Туг и Амар Гупта писали в журнале Scientific American: "Если бы последние 25 лет авиационная промышленность развивалась столь же стремительно, как и вычислительная техника, то сегодня можно было бы приобрести "Боинг-767" за 50 долларов и облететь на нем земной шар за 20 минут, израсходовав при этом 19 литров горючего". Используя эту аналогию (хотя она и не совсем точна), можно судить о темпах снижения стоимости, энергопотребления и роста быстродействия вычислительных машин. Цены на логические процессорные элементы компьютеров ежегодно снижаются на 25%, а для устройств памяти данный показатель еще выше — 40%. За 25 лет скорость вычислений возросла в 200 раз; при этом размеры и потребление энергии у современных ЭВМ стали в 10 000 раз меньше, чем у машин сравнимой производительности 25-летней давности.

Стандартный микрокомпьютер с памятью достаточного объема и развитой периферией (множеством портов для устройств ввода-вывода) содержит около 50 интегральных схем и может быть реализован на одной печатной плате. Такие машины называются одноплатными микроЭВМ. Микрокомпьютер с маленькой памятью и небольшим числом портов ввода-вывода можно изготовить в одной-единственной СБИС (то есть в одном чипе). Одночиповые микроЭВМ получили широкое распространение благодаря небольшим размерам, низкой цене и, так сказать, простоте ремонта (выбросить неисправный, установить новый). Их производство началось в середине 70-х годов, вскоре после создания первых микропроцессоров, а применялись эти компьютеры в основном в качестве программируемых контроллеров для различных машин, механизмов и ЭВМ.

Примерно тогда же развитие компьютерной техники вступило в новую фазу, которая была ознаменована появлением нового класса вычислительных машин широкого применения, называемых персональными компьютерами. Эти маленькие дешевые микрокомпьютеры предназначены для установки на столе офиса, в кабинете ученого, на парте школьника и на кухне у домохозяйки. Некоторые персональные компьютеры настолько компактны, что их можно брать с собой в путешествие. Они питаются от аккумуляторов, имеют небольшой вес. В состав обычного персонального компьютера входят клавиатура, монитор и системный блок, где и размещается плата с микропроцессором. Для связи с внешним миром компьютер использует телефонные линии, для хранения данных — различные магнитные диски, для ввода графической информации — сканеры и манипуляторы, для создания твердых копий — принтеры и графопостроители.

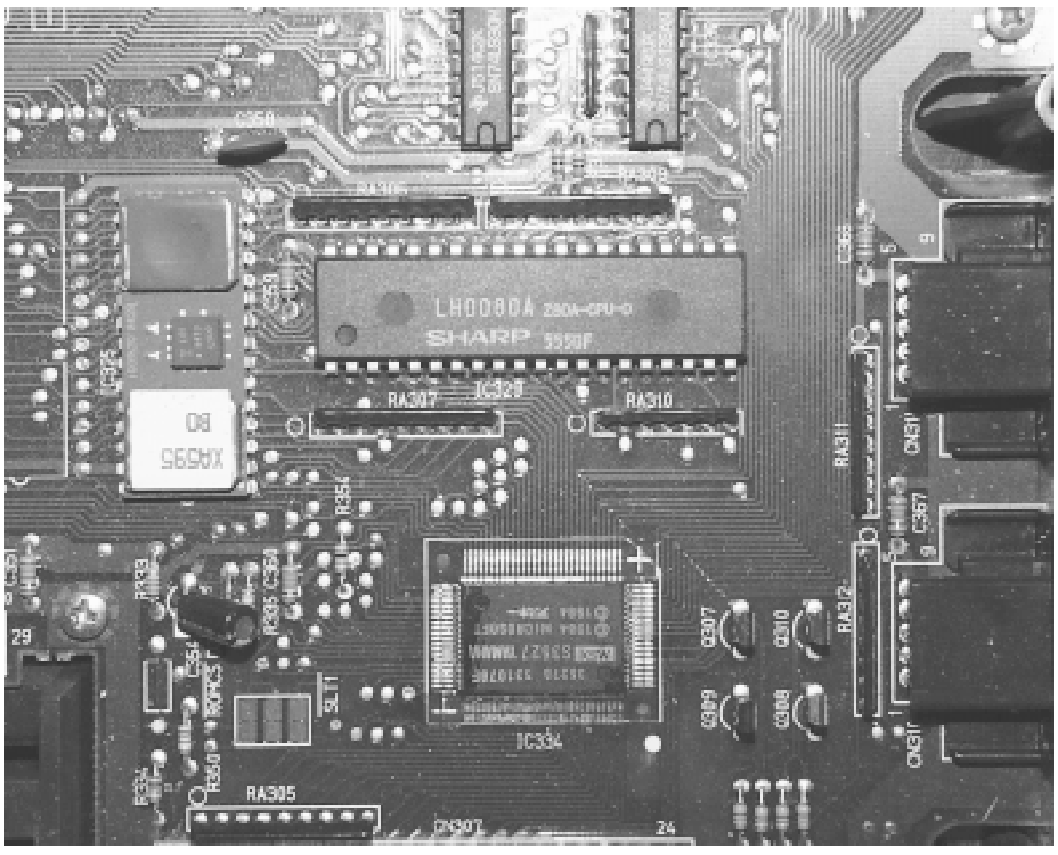
Одной из первых микроЭВМ можно считать компьютер Intellec-8 на основе микропроцессора Intel-8008 (более ранняя модель, чем 8080), который стоил около двух с половиной тысяч долларов.

В Великобритании в 80-х годах пользовалась популярностью модель персонального компьютера на основе процессора Z80 (и с памятью объемом 48 Кбайт), разработанная Клайвом Синклером, Sinclair ZX-Spectrum. Процессор размещался в одном корпусе с клавиатурой, а в качестве монитора использовался домашний телевизор. Стоила такая ЭВМ несколько сотен долларов. Это был по-настоящему домашний компьютер, доступный по цене практически каждому.

Другой популярной машиной на базе микропроцессора Z80 являлся компьютер MSX, появившийся в 1982 году.

Эти типы домашних компьютеров получили широкое распространение в Японии, Корее, Южной Америке, Голландии, Франции, а 4 тысячи таких машин, привезенных в СССР, позволили с 1986 года развиваться школьной информатике.

MSX представлял собой как бы гибрид приставок класса Nintendo *(Nintendo Entertainment System)* и обычного персонального компьютера на базе микропроцессора Z80 (с тактовой частотой 3,58 МГц). Видеопроцессор (процессор, формирующий изображение на экране) был взят у фирмы Texas Instruments (чипы TI9918 или TI9928). Эти процессоры значительно превосходили видеопроцессоры IBM PC XT и AT (1983—1988 гг.), например, по количеству отображаемых цветов. Машина имела аудиосистему — трехканальный музыкальный синтезатор, что вообще являлось редкостью для ЭВМ. У MSX-1 объем памяти составлял 64 Кбайта, а у MSX-2 — 128 Кбайт (максимально 4 Мбайта). По всей видимости, модели MSX были лучшими машинами, построенными на 8-битных процессорах.



Одним из наиболее распространенных семейств персональных компьютеров является семейство IBM PC, впервые представленное в 1981 году. Сегодня машины этого типа фактически стали стандартом персональных компьютеров. Надо отметить, что первые модели серий IBM PC XT (на базе микропроцессора Intel-8086) и IBM PC AT (на базе микропроцессора Intel-80286) имели очень серьезный недостаток — слабые возможности видеопроцессоров, однако после того как компьютеры семейства IBM PC начали выпускаться не только корпорацией IBM (такие машины называют совместимыми с IBM PC, или IBM-совместимыми), а к производителю процессоров добавилась фирма AMD (Advanced Micro Devices), характеристики машин существенно улучшились.

В США родоначальниками персональных компьютеров считаются Стивен Джобс и Стив Возняк, заложившие основы тоже весьма популярного сейчас компьютера Макинтош (Macintosh, или просто Mac). В 1976 году они основали фирму Apple в Паоло Альто (Калифорния).

Первый компьютер Apple (имевший всего 8 Кбайт памяти) был собран в их знаменитом гараже. А уже в середине 1977 года на рынке появилась новая модель — Apple II. Компьютер подключался прямо к телевизору, а используемый в Apple процессор (марки 6502) стоил всего 25 долларов. Цена являлась определяющим фактором при выборе процессора, возможности которого были довольно слабыми: 5 регистров, из них два служебных 12-битных (PC и SP), остальные 8-битные.

В 1983 году фирма Apple разработала компьютер Lisa со странным устройством — манипулятором для ввода графической информации. Большинство ученых и разработчиков скептически отнеслись к представленной новинке. Это странное устройство называлось мышью! Стоимость компьютера Lisa была очень высокой — около 10 тыс. долларов.

В 1984 году появился менее дорогой компьютер Макинтош, который имел черно-белый монитор, аудиосистему и операционную систему с графическим интерфейсом. Наконец, в 1994 году был представлен первый Макинтош на базе RISC-микропроцессора Motorola PowerPC.

Персональные компьютеры всего за 20 лет полностью захватили мир, они стали для офисов таким же обычным инструментом, как и пишущие машинки. Основная область применения персональных компьютеров (а к ней относилась главным образом обработка текстов) в последние годы пополнилась мультимедиа-приложениями и средствами мировой коммуникации. Персональные компьютеры становятся практически столь же мощными, как и большие машины. Круг задач, которые они способны решать, все больше расширяется.

МАТЕРИАЛЫ К УРОКУ

ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ. Вы, наверное, знаете, что в описанной выше картинке кипучей бюрократической (в хорошем смысле этого слова) деятельности (когда начальник расписывает подчиненным, что они должны делать, распределяет между ними работу) в реальной жизни также используются слова “исполнитель”, “подысполнитель” (или “субисполнитель”) и пр. И это, конечно, не случайно. Это хорошая аналогия, хорошее пояснение, и оно вам может пригодиться.

Как, например, делается какое-то новое большое дело? Создается организация, директору поручается все сделать, и он начинает думать, кто, какие отделы, люди ему для этого нужны, кто чем должен заниматься и т.п. Дальше директор утверждает штат, принимает на работу сотрудников, поручает им дела, и организация начинает как-то функционировать. Слово “исполнитель” в этом смысле взято из обыденной или, если хотите, “организационно-бюрократической” жизни. Мы придумываем, какие нам нужны вспомогательные исполнители, нанимаем их на работу и поручаем им сделать то или иное дело. Они, в свою очередь, поручают более мелкие дела своим подысполнителям, отделам и подотделам и т.д. Дальше все это выполняется, и результаты передаются вверх по цепочкам к начальству.

Правда, в отличие от обыденной жизни, в информатике все очень строго формализовано — ведь алгоритмы выполняет ЭВМ. Поэтому все команды выполняются ***в точности*** так, как мы их записали. Конечно, мы можем ошибиться при записи алгоритмов, но ЭВМ всегда выполняет именно то, что мы ей написали.

На этой же житейской аналогии я сейчас бегло поясню смысл термина параллельные процессы. Когда один исполнитель-человек просит что-либо сделать другого, то первому вовсе не обязательно сидеть сложа руки и ждать, пока второй завершит свою работу и сообщит результат. В реальной жизни первый человек вполне может заняться чем-то полезным, хотя бы, например, раздавать поручения другим своим подысполнителям.

В информатике же, как вы помните, при вызове вспомогательного алгоритма (в том числе и при вызове алгоритма из вспомогательного исполнителя) выполнение основного алгоритма приостанавливается и возобновляется только после полного выполнения вспомогательного алгоритма. Ведь все эти алгоритмы и исполнители помещаются в памяти ЭВМ, и процессор выполняет их последовательно — в каждый момент времени выполняется только какой-то один алгоритм.

Принцип последовательного выполнения, однако, не догма. Существует отдельный раздел информатики, посвященный методам организации параллельного (одновременного) исполнения нескольких алгоритмов (это могут быть несколько ЭВМ, исполняющих разные алгоритмы, или одна ЭВМ с несколькими процессорами, или даже одна ЭВМ с одним процессором, переключающимся с выполнения одного алгоритма на выполнение другого и обратно). Каждый такой параллельно (одновременно) исполняемый алгоритм называется ***процессом***, а соответствующая область информатики — теорией “параллельных процессов”. Существуют и специальные языки программирования, специальные понятия и языковые конструкции, предназначенные для организации таких процессов (сигналы, флаги, семафоры, критические области, захват и освобождение ресурсов, прерывания и т.п.). Ведь если “начальник”, отдав команду исполнителю и не ожидая результата, продолжает заниматься “своими делами” — т.е. при вызове вспомогательного алгоритма выполнение основного не приостанавливается, — то требуются некие специальные системы связи, чтобы подчиненный мог сообщить, что он закончил работу, или чтобы начальник мог поинтересоваться, не окончена ли уже порученная работа и каков результат.

В учебнике, впрочем, эта область совершенно не затронута. Учебник посвящен исключительно и только так называемому “последовательному” программированию.

Возвращаясь к учебнику, хочу заметить, что с изучением § 22 мы прошли все четыре фундаментальных понятия информатики и вышли на “плато”, на котором проложено много разных дорог в разные стороны. И теперь мы можем двигаться в любую сторону, углубляться в любую конкретную область информатики. Одна из дорог, например, ведет в сторону “параллельных процессов”, другая — в “конструирование типов и структур данных”, третья — в “методы алгоритмизации и доказательства правильности программ” и т.д.

В упражнениях после параграфа имеются самые разные задачи как на дополнение и изменение уже заданных исполнителей, так и на реализацию новых. Я обращаю ваше внимание на упр. 8 (и рекомендую его сделать), где требуется, используя в качестве вспомогательного исполнителя Чертежник, реализовать исполнителя Черепашка, система команд которого взята из черепашьей графики в языке Лого.

Теперь мы наконец можем перейти к заключительной части нашего курса, а именно — к применениям ЭВМ. § 21 и § 22 были подготовкой, необходимой для того, чтобы мы имели язык, на котором можно описывать, ***как*** представляется и обрабатывается информация в тех или иных применениях ЭВМ. Соответственно, эти применения (§ 23—27) можно рассматривать как примеры, поясняющие и углубляющие понятия, изложенные в параграфах 21—22. Никаких новых понятий больше в курсе вводиться не будет. Остались только “применения” уже изученных понятий к разным областям использования ЭВМ и изложение некоторых специфических методов обработки информации в конкретных областях (как, например, метод дискретизации непрерывных процессов в § 26).

Список литературы

[ПдМ] *Кушниренко А.Г., Лебедев Г.В.* Программирование для математиков. М.: Наука, 1988.

[Гейн] *Гейн А.Г., Житомирский В.Г., Линецкий Е.В. и др.* Основы информатики и вычислительной техники. М.: Просвещение, 1991.

№ 11 (204) март 1999

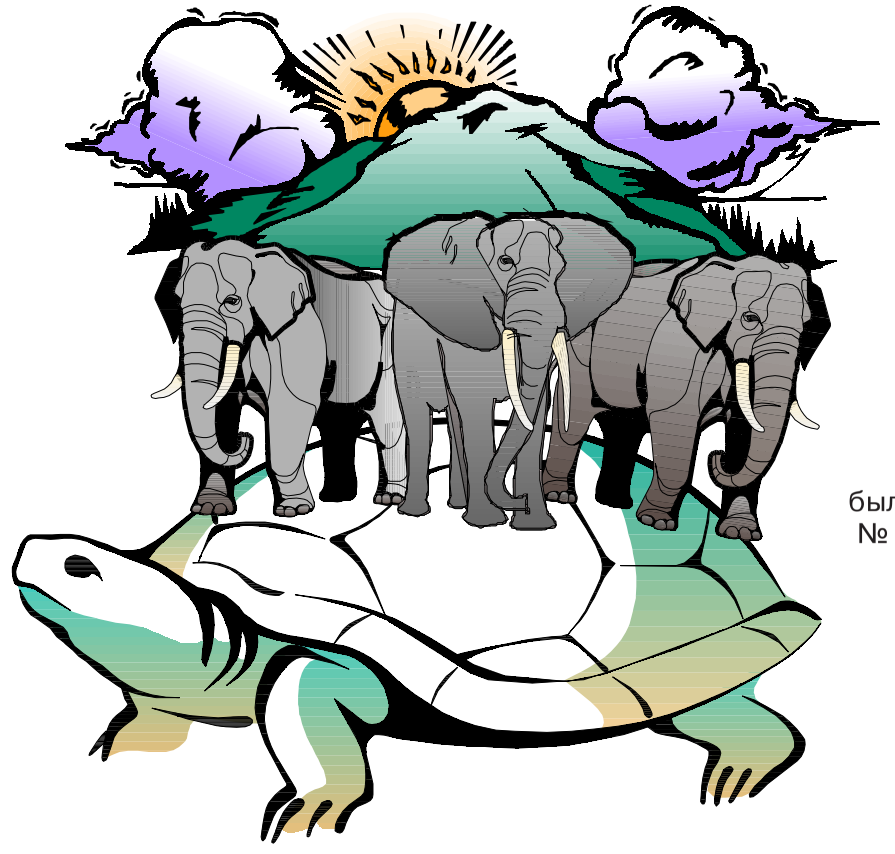
еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИК

12 лекций о том, для чего нужен школьный курс информатики и как его преподавать

Лекция 10

А.Г. КУШНИРЕНКО,
Г.В. ЛЕБЕДЕВ



Введение,
Лекции 1, 2, 3, 4,
5, 6, 7—8, 9
были опубликованы в
№ 1, 3, 5, 6, 8, 10/99

Выражаем признательность издательству “Дрофа” за содействие в подготовке публикации.

© ИнфоМир. Печатается с сокращениями.

СОДЕРЖАНИЕ

Именно в этом смысле — как средство алгоритмизации — понятие исполнителя является тем четвертым фундаментальным понятием современной информатики, про которое я говорил. Так же, как вспомогательные алгоритмы используются для структуризации действий, исполнители нужны для одновременной структуризации совокупности и объектов, и действий, связанных с этими объектами.

Этого понятия — исполнителя как конструкции языка и как средства алгоритмизации — пока нет в большинстве школьных учебников, поскольку оно появилось существенно позже понятия вспомогательных алгоритмов (подпрограмм) и поскольку реальная потребность в этом понятии возникает только при решении достаточно сложных задач. Хотя, повторю, примерно к 1980 году это понятие вошло абсолютно во все языки программирования (“пакет” в Аде, “модуль” в Модуле, “класс” в Симуле, “экземпляр объекта” в Smalltalk и т.д.). По-разному называемое, оно тем не менее есть во всех современных языках программирования.

> ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ. Мы называем это понятие исполнителем с легкой руки Владимира Борисовича Бетелина, который первоначально использовал слово “исполнитель” при русскоязычном описании понятия класс в языке Параллельный Паскаль. Потом, в ходе преподавания курса программирования на мехмате МГУ, это слово стало использоваться в его нынешнем значении (см. учебник). Как всякое удачное слово, оно начало распространяться и было задействовано Г.А. Звенигородским для описания устройств вроде Дежурика, Черепашки и пр. при преподавании программирования школьникам (т.е. в том значении, в котором мы его использовали в главе 1). Мы, в свою очередь, заимствовали у Звенигородского термины предписание и система предписаний, использующиеся в курсе программирования на мехмате МГУ вместо перегруженных команда и система команд. Исполнитель как конкретная конструкция языка программирования возник также в курсе программирования на мехмате МГУ (см. [ПдМ]) и оттуда, лишь слегка видоизменившись, перекочевал в школьный курс и школьный алгоритмический язык.

Возвращаясь к понятию исполнителя как инструменту алгоритмизации, я еще раз хочу отметить, что нужда в этом инструменте возникает только при решении достаточно сложных, больших задач. Подобно тому как для составления алгоритма из двух действий не требуется понятие вспомогательного алгоритма, для составления простых алгоритмов из 10—15 строк не требуется понятие исполнителя. И даже алгоритм из 100 строк можно составить, не пользуясь этим понятием.

Поэтому мы не можем продемонстрировать в полной мере ту чрезвычайную важность, которую это понятие имеет при решении сложных задач. И в учебнике мы ограничились лишь несколькими словами (п. 22.7), констатирующими, что метод последовательного уточнения с использованием исполнителей — ***основной метод алгоритмизации сложных задач***. Этот пункт (22.7) помечен звездочкой, поскольку не поддержан упражнениями и, соответственно (вспомните “черепаху”!), нормально усвоен быть не может. Тем не менее, поскольку это ***основной*** метод, совсем мы его опустить не могли и привели словесное описание.

Дело в том, что реальные программные системы создаются так.

Во-первых, исходная задача обычно формулируется как требование создать исполнителя, который будет уметь выполнять некоторый набор команд. (Вместо слова “исполнитель” часто говорят “система”.) Например, система продажи билетов должна уметь отвечать на вопросы про проданные и непроданные билеты и наличие рейсов: продать билет и учесть возврат билета, подсчитать, сколько всего денег должен сдать кассир в конце смены, выдавать среднюю загрузку рейсов, чтобы можно было принимать решения, на каких направлениях надо вводить дополнительные рейсы, и т.д. и т.п. Получается очень много разных команд.

Во-вторых, основной метод решения состоит в том, чтобы сказать, что если бы у нас были такие-то исполнители, умеющие делать то-то и то-то, тогда мы нашу задачу могли бы решить, и соответствующие алгоритмы (и исполнители) записываются так-то. Важно, что в этот момент можно ***придумать*** себе в помощь таких исполнителей, использование которых позволяет достаточно легко решить исходную задачу. Такой переход называется шаг декомпозиции, или шаг детализации, он сводит исходную задачу к нескольким более простым задачам, для каждой из которых, в свою очередь, придумываются еще чуть более простые вспомогательные исполнители. И так постепенно, шаг за шагом, все упрощая и упрощая исполнителей, доходят до совсем простых, которые уже можно просто написать на языке программирования.

Обычно при этом происходит еще и разделение труда. Тот, кто отвечает за создание всей системы целиком, делает первый шаг детализации, получает описание, скажем, 10 подысполнителей попроще и раздает их на реализацию 10 разным людям (или даже 10 разным отделам). В этих отделах происходит следующий шаг детализации — и описания подподысполнителей раздаются на реализацию подотделам и т.п.

Чтобы вы представили себе, о чем идет речь, замечу, что трудозатраты на создание системы продажи авиабилетов “Сирена” оцениваются в 300 человеко-лет. Это значит, что коллектив из 60 человек работал над системой в течение 5 лет или коллектив из 100 человек — в течение 3 лет. Конечно, в это время входят и поиск ошибок, и исправления, и переделки. Но все равно любая реальная система — это, как правило, большая и сложная работа. Продемонстрировать методы решения таких задач мы можем только на очень упрощенных — школьных — примерах. И именно этому и будут посвящены § 23—27 (применения ЭВМ).

Самое главное в этом методе алгоритмизации с использованием исполнителей то, что при проведении шага детализации люди придумывают не отдельные вспомогательные действия (алгоритмы), а целиком вспомогательных исполнителей. И лишь потом — в ходе их реализации — уточняют, какие в исполнителях будут общие величины и что с этими величинами будет происходить при выполнении команд.

надо его ***реализовать*** (т.е. написать на алгоритмическом языке). А потом им можно будет пользоваться — совсем как "настоящим", и не придется обращаться к инженерам и закупать дорогостоящие материалы.

Поэтому мы будем описывать реализацию (информационную модель) исполнителя И1 на алгоритмическом языке. Она приведена целиком на с. 182—183 учебника. Этот текст, эта конструкция **исп...кон**, и есть изготовление исполнителя И1. Сверху заданы общие величины исполнителя И1:

— вещественная таблица **a[1:100]**, где будут запоминаться числа (т.е. наш И1 будет уметь запоминать не более 100 чисел), и

— целочисленная величина **n** — количество реально запомненных чисел.

Далее следует установка начальных значений: **n:=0**, т.е. вначале ни одного числа не запомнено. А потом идут алгоритмы-"лепестки" нашего исполнителя. Их у нас три: алгоритм "запомнить", алгоритм "число больших" и алгоритм "минимум".

Важно, что теперь, когда исполнитель И1 реализован, мы можем поместить в ЭВМ программу, которая содержит и алгоритм А96, и исполнителя И1, что позволит нам реально выполнить алгоритм А96. При этом ЭВМ, выполняя "команды" И1, запомнит соответствующие числа в таблице **a[1:100]**, сравнит их с уровнем радиации в конечной точке и выдаст ответ. Таким образом, вместо изготовления реального железного исполнителя И1 мы заставили ЭВМ его проимитировать, изготовили его информационную модель.

Но поскольку при написании алгоритма А96 нет абсолютно никакой разницы между "железным" и "информационным" исполнителем (алгоритм А96 вообще был написан ***до*** того, как мы решили, каким образом изготавливать И1), в дальнейшем мы можем говорить не "информационная модель", а просто "исполнитель". Нас совершенно не интересует, работает при этом ЭВМ или специальное устройство, спаянное инженерами. Главное, чтобы выполнялись команды исполнителя. А уж будут ли они выполняться "прибором", подключенным к ЭВМ, или программой, помещенной в память ЭВМ, — совсем не важно.

Конечно, при ***реализации*** исполнителя надо знать и понимать, как записывается информационная модель исполнителя, что значат и как понимаются и выполняются в ЭВМ различные элементы "ромашки" исполнителя, где и как размещаются общие величины исполнителя, когда они появляются в памяти ЭВМ и когда исчезают из нее, и т.д., и т.п. Но при ***использовании*** исполнителя все это совершенно не должно нас интересовать.

Как я уже говорил, один из методических приемов учебника — показывать и описывать новые понятия со всех возможных сторон. И следующий пункт параграфа (п. 22.5) посвящен описанию того, как ЭВМ размещает общие величины в своей памяти, а также содержит пример их изменения в ходе выполнения алгоритма А96. Важно отметить, что общие величины исполнителей заводятся в памяти ЭВМ ***до начала выполнения*** каких-либо алгоритмов. Также ***до начала выполнения алгоритмов*** выполняются фрагменты, которые я называл установкой начальных значений, — то, что расположено в конструкциях **исп...кон** перед текстами алгоритмов. И только ***после*** этого начинается исполнение "основного" алгоритма. Если внутри алгоритма промежуточные величины появляются при начале выполнения алгоритма и исчезают, когда выполнение алгоритма заканчивается, то общие величины исполнителей существуют все время (появляются до начала выполнения чего бы то ни было и исчезают после окончания всего).

> А теперь главное — понятие "исполнитель" как способ алгоритмизации, как метод мышления при решении алгоритмических задач.

Исполнителей можно использовать, даже если задача чисто информационная, если никаких исполнителей в постановке задачи нет.

В алгоритме А96 все-таки, кроме И1, фигурировал и наш старый знакомый — Робот. В алгоритме А99 (п. 22.6, с. 185 учебника) решается задача, именуемая в науке "минимаксом": дана прямоугольная таблица (в А99 — **вещ таб t[1 : 60,1 : 60]**), в каждом столбце надо найти максимум (это будет 60 чисел) и среди них выбрать минимум.

Эта задача из так называемой теории игр. Я не буду вдаваться в детали (при желании вы можете найти их в литературе), но, грубо говоря, результат работы алгоритма "минимакс" — это результат игры, когда и вы, и ваш противник играют наилучшим возможным образом: вы стараетесь максимизировать свой выигрыш, а противник стремится его минимизировать.

А теперь забудем про всю эту теорию игр и посмотрим на нашу задачу просто как на упражнение по программированию. Итак, надо по прямоугольной таблице вычислить "минимакс". Никакого Робота, никаких исполнителей вообще, чисто информационная задача. Можно решать и на Бейсике. Но приведенные в учебнике алгоритмы А99—А100 решают эту задачу с использованием исполнителя И1! После нахождения максимума в очередном столбце А99 командует И1 "запомнить", а после запоминания всех максимумов от И1 получается минимум из запомненных чисел, т.е. "минимакс".

Решение, возможно, чуть-чуть искусственное, но обратите внимание: ни в постановке задачи, ни в процессе ее решения, ни в ответе никаких предопределенных исполнителей нет. Мы используем И1 не как нечто заданное, а как привнесенное, придуманное нами в ходе решения задачи. Как метод алгоритмизации. Как достояние нашего знания, нашей головы независимо от исходной постановки задачи.

Понятие исполнителя стало инструментом алгоритмизации. В отличие от Робота, который был задан извне, был предопределен и выступал как прием для освоения базовых понятий алгоритмизации, здесь произошла очень существенная замена. Мы используем придуманных нами исполнителей для решения задач, в которых и в помине нет никаких исполнителей, в совершенно информационных, в чисто вычислительных задачах. Используем их так же, как вспомогательные алгоритмы, циклы и **если**, как ***средство*** решения задач.

Лекция 10

Глава 3. Применения ЭВМ

Глава 3 — это в каком-то смысле то, ради чего затевалось все предыдущее.

Глава 1 была посвящена изучению школьного алгоритмического языка, базовым навыкам составления алгоритмов и простейшим (за исключением § 16) методам алгоритмизации. Это уровень "письма и счета": мы не обсуждаем, почему мы решаем те или иные задачи. Так, на уроках арифметики, когда мы учим складывать, никого не волнует, почему мы складываем, например, карандаши, а не груши. Мы в этот момент учим складывать, а не анализируем сравнительные достоинства карандашей перед грушами. Точно так же в главе 1 не обсуждается, почему мы решаем одни задачи, а не другие. Их просто надо решать, надо "набивать руку" и учиться составлять алгоритмы.

Глава 3 посвящена "Применениям ЭВМ", т.е. задачам из реальной жизни. И здесь уже мы будем объяснять и откуда берутся задачи, и как в реальной жизни они решаются.

Выбор такой последовательности изложения — это выбор методики обучения. Бывают и другие подходы. Например, в учебнике свердловчан [Гейн] практически все разделы с самого начала построены так: сначала идет мотивационная жизненная постановка задачи, а уж потом строятся модели (математические) и алгоритмы.

И у нашего, и у свердловского подхода имеются свои достоинства и недостатки. Достоинство свердловчан — наличие мотивации с самого начала, разнообразие постановок задач и как следствие — более разнообразный и занимательный курс. Но на первых порах, когда еще мало освоенных понятий, модели получаются куцыми и иногда неадекватными. И это — основной недостаток.

У нас в первой части создание мотивации возложено на учителя. Учебник ничего такого не содержит, является сухим и сжатым. Но зато при рассказе о применениях ЭВМ мы можем использовать весь круг понятий и демонстрировать полноценные информационные модели, т.е. работать на достаточно глубоком уровне, адекватном содержанию задачи.

Глава называется "Применения ЭВМ" и, как кажется, должна была бы сразу начинаться с рассмотрения каких-то областей применения ЭВМ, каких-то жизненных ситуаций. На самом деле это не так. Глава начинается с § 21 и § 22, которые, по сути, продолжают главу 1 — продолжают развитие базовых понятий и методов алгоритмизации.

§ 21 посвящен понятию информационной модели, а § 22 — четвертому, самому главному, фундаментальному понятию информатики, четвертому фундаментальному понятию алгоритмической культуры — понятию информационной модели исполнителя.

Действительное изложение применений ЭВМ начинается только с § 23 и продолжается до § 27. Здесь описываются информационные системы, текстовые редакторы, моделирование на ЭВМ и пр.

Все эти параграфы устроены одинаково: вначале идет беллетристика, т.е. обычное описание очередной области применения ЭВМ, например, системы продажи авиабилетов или системы, управляющей конвейером на Волжском автозаводе. После такого описания идет вторая — гораздо более содержательная — часть: разбираются учебные информационные системы, демонстрирующие методы представления и обработки информации в соответствующей области. Грубо говоря, если первая часть параграфа отвечает на вопрос "что?" могут делать ЭВМ в данной области (как это выглядит "снаружи"), то вторая часть пытается ответить на вопрос "как?" это делается — как представляется информация, каковы алгоритмы ее обработки, т.е. как это выглядит "изнутри".

Такой подход (кажется, с легкой руки Зайдельмана) получил название "принцип корыта". В самом учебнике, описывая структуру главы (с. 169), мы написали, что, рассказав о реальных применениях ЭВМ, мы на учебных информационных системах покажем, как такие системы могут работать. И привели аналогию: рассказав об океанских лайнерах, мы покажем "корыто" и объясним, почему оно плавает. Конечно, мы имели в виду, что не сможем показывать и обсуждать детали, а объясним только самое, на наш взгляд, важное — "почему оно плавает". Но народ немедленно обозвал этот подход "принципом корыта", и ввиду образности и легкости запоминания это название так и пошло.

Хочу обратить ваше внимание, что первые (описательные) части § 23—27 являются традиционными и аналогичный (и даже более интересный и увлекательный) материал вы можете найти практически в любой популярной или учебной книге, посвященной ЭВМ. Но в отношении содержательной второй части § 23—27 это не так. По сути, мне неизвестны ни учебники, ни научно-популярные издания, которые бы содержательно отвечали на вопрос "как?", объясняли бы, "почему же оно все-таки плавает", демонстрировали бы применения ЭВМ "изнутри".

Замечу, впрочем, что в § 27 эта вторая, содержательная, часть приведена в весьма ограниченном и усеченном виде: показана только информационная модель без примеров алгоритмов ее обработки.

> ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ. Я честно признаюсь, что это отнюдь не замысел авторов. В действительности все куда проще и прозаичнее. По техническим причинам учебник обязан был иметь объем, кратный трем печатным листам, — какие-то машины в типографии почему-то сшивали толь-

ко такие книги. А три печатных листа — это 48 страниц учебника! И мы должны были подгонять объем учебника под это требование. Поэтому нам пришлось выкинуть из учебника некоторое количество "применений" и, в частности, в § 27 остался только маленький — на наш взгляд, самый важный — фрагмент от раздела "как?". Никакие методические, педагогические или иные содержательные соображения за этим не стоят.

Наконец, последний § 28, хотя и описывает некоторые применения ЭВМ, в целом является скорее заключением ко всему учебнику, чем к главе 3. Этот параграф, на мой взгляд, важен, поскольку он единственный, где говорится, что ЭВМ не панацея и их применение может быть не только полезным, но и вредным.

Такова структура главы 3: первые два параграфа — самые важные и посвящены четвертому фундаментальному понятию информатики — информационным моделям исполнителей; параграфы с 23-го по 27-й посвящены применениям ЭВМ и их описанию "снаружи" и "изнутри"; наконец, § 28 является заключением ко всему учебнику и посвящен роли и месту применений ЭВМ в нашей жизни.

§ 21. "Информационные модели" или по-другому — "Кодирование информации величинами алгоритмического языка"

Сказать по правде, кодированием информации мы занимались непрерывно в течение всего курса, только раньше это никак не называли.

Самый последний пример (из § 20) — кодирование буквами "В", "Н", "Л", "П", "К" команд Робота, а строками из этих букв — алгоритмов по управлению Роботом. Да и до этого много похожего встречалось. Один из самых ярких примеров — занесение в линейную таблицу информации об уровнях радиации в коридоре (см. с. 110 учебника — алгоритм А62).

Но в главе 3 для нас уже важна "жизненность" примеров, так что Робот тут не годится. И поэтому параграф начинается с построения информационной модели кинозала.

> ОПРЕДЕЛЕНИЕ: информационной моделью объекта, явления и пр. называется набор величин языка программирования (для нас — школьного алгоритмического языка), с помощью которого мы задаем этот объект, явление и пр. Когда мы задаем какой-то объект с помощью некоторого набора величин алгоритмического языка, этот набор величин называется информационной моделью объекта.

Если мы хотим написать программу, продающую билеты на поезда, то набор величин, задающий все возможные маршруты, все проданные и непроданные места, цены билетов и пр., — это и есть информационная модель. Для программы на школьном языке информационная модель — это набор величин школьного алгоритмического языка. Для программы на Паскале информационная модель — набор переменных Паскаля, используемых в программе. Таким образом, информационная модель — просто набор величин выбранного языка программирования.

Итак, глава 3 начинается с информационной модели кинозала. Пусть надо обеспечить продажу билетов в один кинозал, который содержит 28 рядов по 20 мест. Мы собираемся продавать билеты всего на один сеанс. Цена билета зависит от ряда. Какую информацию нам придется хранить? Про каждое место надо знать, продан или не продан на него билет (т.е. свободно это место или уже продано). Надо также где-то хранить информацию о ценах на билеты, чтобы знать, почем их продавать. Все остальное для продажи билетов вроде бы не важно.

В учебнике вводятся две величины для хранения этой информации: первая — логическая таблица **`продано[1:20,1:28]`** — прямоугольная таблица, в точности соответствующая кинозалу, — в каждой клетке будет **`да`**, если место продано, и **`нет`**, если место не продано. Другими словами, **`продано[i, j] = да`** означает, что *i*-е место в *j*-м ряду продано. И вторая величина — целочисленная таблица **`цена[1:28]`**, которая по номеру ряда говорит, сколько стоит билет в этом ряду. Например, **`цена[5]`** — это цена билета в 5-м ряду, **`цена[28]`** — это цена билета в 28-м ряду.

Эти две величины и есть информационная модель кинозала (обозначенная в учебнике как модель М1). Алгоритмы обозначаются буквой "А" и номером; информационные модели — буквой "М" и номером. М1 — первая информационная модель нашего учебника. Всего в учебнике их будет 20. Итак, эти две величины (М1) — информационная модель кинозала для поставленной нами задачи продажи билетов на один сеанс — я об этом еще скажу.

Но сейчас я хочу еще и еще раз подчеркнуть, что информационная модель — это набор величин. Заметьте, самих величин, а не их значений, что очень важно понимать. Информационная модель — это сами величины "продано" и "цена". Мы пока не знаем, что там внутри, каковы значения величин, какие билеты проданы, какие нет и каковы цены — все это не важно. В информационную модель значения величин не входят. Другое дело, что потом мы можем использовать эту модель для представления (кодирования, задания) той или иной конкретной ситуации. Но это будет уже использование информационной модели. Сама же информационная модель — это просто набор величин, безотносительно значений этих величин.

А теперь вернемся к тезису "модель ***для*** поставленной нами задачи". Пусть у нас где-то есть настоящий, реальный кинозал, в котором 28 рядов по 20 мест. Когда мы придумываем информационную модель для представления информации об этом кинозале, наш выбор в первую очередь определяется тем, какие задачи мы собираемся решать с помощью информационной модели. Ведь реальный кинозал содержит

"вправо" — значит, ЭВМ скомандует **`вправо`**, а будет ли при этом выполнен вспомогательный алгоритм или же подана команда "железному" Роботу, зависит от того, как на самом деле устроена жизнь, т.е. подключен ли к ЭВМ реальный Робот или в ЭВМ введена реализация (информационная модель) Робота.

Исполнитель И1.

Первый содержательный пример исполнителя в учебнике — исполнитель И1 из п. 22.3 (с. 181—183). Почему я говорю "содержательный"? Потому что ценность понятия исполнителя не в том, чтобы промоделировать Робота, а в том, чтобы в задаче, не имеющей, казалось бы, никакого отношения ни к каким исполнителям, воспользоваться понятием исполнителя как методом алгоритмизации, т.е. использовать конструкцию **`исп...кон`** для облегчения жизни, как метод решения задачи.

> МЕТОДИЧЕСКОЕ ОТСТУПЛЕНИЕ. Естественно, наш первый "содержательный" пример на самом деле тоже будет простым, учебным. В реальной жизни потребность в использовании исполнителей возникает только при решении достаточно сложных задач. Мы же пытаемся приводить примеры попроще. Поэтому и исполнители у нас будут упрощенными, учебными.
>
> По этой же причине при изложении темы исполнителей мы отошли от проблемного подхода, иначе нам пришлось бы формулировать слишком трудоемкие "проблемы" и тратить слишком много времени, прежде чем потребность в исполнителях (как конструкциях алгоритмического языка) была бы осознана. В частности, наш первый исполнитель И1 возникает не в процессе решения задач, а "сам по себе" — он сначала описывается, а уж потом используется. Хотя при реальной работе обычно сначала возникает потребность в исполнителе, формулируется его система команд и проч. и только потом пишется реализация исполнителя на языке программирования.

Итак, в учебнике (с. 181) приводится пример исполнителя И1 — абсолютно "чистого" внутри машинного исполнителя, не имеющего отношения ни к каким Роботам, Чертежникам и пр. Название исполнителя первоначально было как-то связано с фразой *"**И1 — в поле воин**"* (и один в поле воин) из книги "Физики шутят". Впрочем, можно считать, что И1 — это просто первый исполнитель учебника.

Как и всякий исполнитель, И1 содержит некоторую информационную модель и алгоритмы-"лепестки". У И1 три "лепестка", три команды, которые можно выполнить, три алгоритма: **`запомнить`**, **`число больших`** (с ударением на букву "о") и **`минимум`**. Исполнителя И1 можно использовать, как и любого другого исполнителя, отдавая ему команды. Например, мы можем скомандовать ему: **`запомни(a)`** — и он запомнит число "а"; потом скомандовать: **`запомни(b)`** и т. д. А после того, как запомнено какое-то количество чисел, мы можем скомандовать: **`число больших (7.5)`** — и И1 выдаст нужный ответ. Или можем с помощью команды **`минимум`** получить минимальное из запомненных чисел.

Обратите внимание, что:

— с одной стороны, исполнитель И1 — совершенно "надуманный" (придуманный), абсолютно информационный и не "железный" по сути;

— с другой стороны, я описал его (а сказать по правде, и придумал) как некоторое внешнее ("железное") устройство — этакий прибор для запоминания чисел. Как обычного "внешнего" исполнителя. Как устройство, про которое совершенно неизвестно, каким образом оно запоминает числа и выполняет другие команды. Я придумал систему команд и описал, что эти команды значат, но я совершенно пока не представляю себе, как И1 будет устроен, как он будет эти команды выполнять, какого он будет размера и веса и сколько электроэнергии он будет потреблять. Тем не менее мы вполне можем уже этим исполнителем ***пользоваться***. И в учебнике на с. 182 приведен алгоритм А96, который, "управляя" исполнителями Робот и И1, подсчитывает число опасных клеток в коридоре.

Как создать исполнителя И1.

Теперь важный сущностный момент. Предположим, для решения каких-то задач нам нужен исполнитель И1, т.е. исполнитель, который умеет запоминать числа, а потом выдавать их минимум и отвечать, сколько чисел среди них больше, чем некоторая заданная величина. Что делать, чтобы в действительности выполнить алгоритм А96 и получить ответ? Есть два пути.

1. Обратиться к инженерам и попросить их изготовить нам такое устройство. Они, видимо, повозятся со своими микросхемами, паяльниками и проводами, соберут нужное устройство, подключат его к ЭВМ, и у нас появится исполнитель И1 (именно таких исполнителей я называю "внешними", "железными", "настоящими"). Это один подход.

2. Ни к кому не обращаться. Вместо "настоящего" исполнителя И1 написать его реализацию (информационную модель исполнителя) на алгоритмическом языке. Для этого придется придумать, где и как будут запоминаться числа, составить алгоритмы, реализующие команды исполнителя, написать соответствующую конструкцию **`исп...кон`** и ввести все это в ЭВМ, т.е. "научить" ЭВМ делать (моделировать, имитировать) работу исполнителя И1. ЭВМ — универсальная машина для переработки информации и потому может переработать любую информацию. (А значит, в состоянии сама выполнить любую работу, в которой не требуется что-либо штамповать, резать или иным образом делать что-то в реальном мире.) Это второй подход.

В курсе информатики (и в учебнике) нас, естественно, интересует второй подход. Мы придумали исполнителя И1, сказали, что он нам нужен, и теперь нам

точнее, объектно-ориентированного подхода к составлению алгоритмов, как раз и объясняется тем, что в основу и методов придумывания, и способов записывания алгоритмов положено понятие исполнителя (объекта).

Я сейчас раскрою роль понятия исполнителя как способа алгоритмизации. Но прежде я хочу напомнить вам рисунок (см. № 3/99, с. 6), демонстрирующий роль понятия исполнитель в нашем курсе и, соответственно, в учебнике. Где-то к § 12 исполнители практически сошли на нет. Мы решали задачи, ни в постановке, ни в решении которых никаких исполнителей не было вообще. И вот мы снова вернулись к исполнителям, но на совершенно другом уровне (как иногда говорят, "на новом витке диалектической спирали"). Вернулись к исполнителям как к внутреннему понятию алгоритмического языка, как к способу решения алгоритмических задач. Именно это и было отражено на графике.

Прежде чем перейти к анализу понятия исполнитель как способа алгоритмизации, давайте уточним, что теперь представляет собой программа на алгоритмическом языке в целом, т.е. что именно мы отдаем на исполнение ЭВМ в схеме программного управления. Вначале это был один отдельно взятый алгоритм. Потом — алгоритм и набор вспомогательных алгоритмов. Теперь, после введения конструкции **исп...кон**, программа, кроме "основного" и вспомогательных алгоритмов, может содержать еще и какое-то количество исполнителей (или, точнее, реализации исполнителей), внутри которых могут быть свои алгоритмы — алгоритмы команд исполнителя.

ПРОГРАММНОЕ ОТСТУПЛЕНИЕ. В реальности — в системе КуМир — "основной" и вспомогательные к нему алгоритмы, которые мы раньше представляли как находящиеся "вне" всяких исполнителей, на самом деле считаются расположенными внутри специального — безымянного и никак особым образом не оформленного — "основного" исполнителя. Соответственно, можно задать общие величины этого исполнителя, доступные всем его алгоритмам, написать фрагмент, устанавливающий их начальные значения (и выполняющийся до запуска "основного" алгоритма), и т.п.

И вот это представление о программе как о наборе алгоритмов и исполнителей является полным в смысле современного состояния информатики.

ЕЩЕ ОДНО ПРОГРАММНОЕ ОТСТУПЛЕНИЕ. В производственных объектно-ориентированных языках программирования внутри исполнителей, как правило, могут быть не только алгоритмы, но и другие исполнители. У нас это не так, поскольку никаких новых фундаментальных понятий такое "упрятывание" одного исполнителя внутрь другого не добавляет.

А теперь представьте себе программу, состоящую из двух частей:

1) некоторый алгоритм управления Роботом, за которым следует

2) реализация исполнителя Маленький Робот со с. 180 учебника.

Как будет выполняться такая программа? При выполнении алгоритма по управлению Роботом, встретив, например, команду **вправо**, ЭВМ вызовет вспомогательный алгоритм "вправо" исполнителя Маленький Робот.

МЕТОДИЧЕСКОЕ ОТСТУПЛЕНИЕ. Обратите внимание, что на с. 180 учебника дана ***упрощенная*** реализация исполнителя Робот. И дело не только в ограниченном размере поля Робота. Например, в приведенном алгоритме "вправо" реализован (промоделирован) только сам шаг вправо. Если при выполнении этого алгоритма справа от Робота будет стена или край поля, то Робот спокойно "пройдет сквозь стену" или выйдет за край поля. Хотя при правильном моделировании в этой ситуации должен возникать отказ. Вы можете указать на эту ***ошибку реализации*** школьникам и попросить их исправить алгоритм. Скорее всего школьники при этом напишут что-то типа:

```
алг вправо
нач
| если справа стена или x=32
| | то отказ
| | иначе x:=x+1
| все
кон
```

Вы можете привести им более "правильный" алгоритм с проверкой в **дано**:

```
алг вправо
  дано справа свободно и x<32
нач
| x:=x+1
кон
```

После чего пояснить им, чем второй алгоритм лучше первого. Вы можете также сказать, что проверки, находится ли Робот на краю поля, можно всюду исключить, если "обнести" поле по краю сплошной стеной. И попросите их записать соответствующий фрагмент установки начальных значений общих величин исполнителя Маленький Робот.

И еще раз обратите внимание, что ***вызов*** алгоритма **вправо** ничем не отличается от ***вызова*** команды **вправо** "настоящего" Робота. Для ***использования***, для вызова команд совершенно все равно, подключен ли исполнитель к ЭВМ реально или же только реализован на алгоритмическом языке. Мы не в состоянии различить такие ***вызовы*** по внешнему виду. Написано

огромную массу информации, которая может быть важна для решения одних задач и совершенно не нужна для других. Любая модель будет содержать не всю информацию о кинозале, а лишь некоторую "избранную". В нашей модели М1, например, нет никакой информации о цвете стен, расстоянии между рядами, наличии поломанных кресел и т.п. Ничего этого в нашей модели нет. Модель, как и всякая модель, математическая или любая другая, является отражением лишь некоторых сторон реальности, а не всего ее разнообразия. Реальность всегда богаче.

В модели М1 мы учитываем только то, что, по нашему мнению, нам нужно знать для продажи билетов. И мы считаем, что высота потолка в кинозале, цвет стен, расположение выходов и проходов, высота ряда над уровнем моря и другие характеристики не важны при продаже билетов. Это наше допущение, или, если угодно, это наша научная гипотеза. Быть может, в реальном кинозале места расположены с подъемом, "веером", с проходами или еще как-то. Но это все мы не учитываем. Именно поэтому информационная модель и называется моделью. Во всякой модели всегда отражается только та часть реальности, которую мы считаем важной для наших задач.

Примерно на этот переход от реальности к модели и сделан основной упор в свердловском учебнике информатики [Гейн]. Упор на анализ того, что теряется и что остается при таком переходе, что мы должны и что не должны учитывать при построении моделей. Однако у свердловчан строится ***математическая*** (а не информационная) модель. Например, моделью поверхности стола является прямоугольник.

У нас модели информационные, а их осознанное построение начинается только в главе 3. "Алгоритмическая составляющая" здесь не очень велика, и с алгоритмической точки зрения деятельность по составлению самих моделей проще, чем деятельность по составлению алгоритмов. Тем не менее построение информационной модели — первый шаг на пути от неформальной постановки задачи к работающему на ЭВМ алгоритму. Поэтому § 21 целиком посвящен построению информационных моделей и составлению алгоритмов с их использованием.

Итак, информационная модель кинозала. Построив модель, мы можем начать решать задачи из соответствующей области (по продаже билетов). Например, имея некоторое ***состояние модели*** (значения величин) в какой-то момент, можно узнать, сколько мест уже продано, а сколько свободно или какова общая выручка от продажи билетов. Эти фрагменты приведены в учебнике (с. 170). Вы также можете попросить учеников написать массу других фрагментов, например, "продать три места рядом", "продать два места рядом в последнем ряду" и т.п. Здесь вполне можно обыгрывать все жизненные ситуации. Причем фрагменты алгоритмов получаются очень простые, много проще тех, которые рассматривались в конце первой главы (т.е. их "алгоритмическая составляющая" не очень велика).

ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ. Я еще раз хочу подчеркнуть, что простота нашей модели и наших алгоритмов определяется тем, что значительную часть информации о кинозале мы не учитываем. Но вот пришел парень с девушкой в кино, попросил два места рядом в последнем ряду и получил места номер 10 и 11. Соседние номера? Да, соседние. А соседние ли места? Увы, неизвестно. Если в зале один центральный проход, то наша пара будет разочарована, получив места с соседними номерами, но разделенные проходом. Так что если мы собираемся с помощью компьютера продавать билеты парам (на соседние места), то в модель придется включить информацию о проходах.

Можно также слегка варьировать постановку задачи, например, потребовать, чтобы все билеты в 13-м ряду и на 13-е место в любом ряду продавались со скидкой. Здесь можно придумать много разных подобных задач, вплоть до таких, которые потребуют внести изменения в саму информационную модель. Например, если места по краям ряда стоят дешевле, чем в середине, то таблицу "цена" приходится делать прямоугольной. Если число мест в разных рядах будет разным, то для задания зала придется придумать еще какие-то величины. И т.п.

МЕТОДИЧЕСКОЕ ОТСТУПЛЕНИЕ. Поскольку вы учителя, я сразу обращаю ваше внимание, что в этом параграфе всюду пишутся ***фрагменты,*** а не алгоритмы. Посмотрите — нигде нет заголовков **алг, дано, надо, нач, кон**. И это не случайно. В этом месте нельзя написать алгоритм целиком. Потому что если мы вздумаем написать **алг**, то встанет вопрос, куда девать величины модели (М1). Мы вынуждены будем включить эти величины в параметры (аргументы и результаты) алгоритма. (Иначе после завершения алгоритма вся информация о состоянии модели пропадет.) Ведь нам надо, чтобы и алгоритм, который продает билеты, и алгоритм, который считает количество проданных билетов, и алгоритм, который считает общую выручку, и другие алгоритмы работали с одной и той же информационной моделью и чтобы ее состояние сохранялось между разными вызовами разных алгоритмов.

В рамках того, что мы уже прошли, есть только один путь — вписать величины модели в аргументы и результаты ***абсолютно всех*** алгоритмов, работающих с данной моделью, и вызывать эти алгоритмы из одного "главного" алгоритма. Для модели М1 это — пусть громоздко и некрасиво — еще можно сделать. А как быть с моделью М16, где 9 величин и из них 7 таблиц? А как быть с реальными моделями, где число величин может измеряться сотнями?

В учебнике мы здесь этот вопрос мягко обходим, записывая только ***фрагменты*** алгоритмов без заголовков.

Вопрос из зала. А как на компьютере выполнять?

Ответ. И на компьютере эти фрагменты не выполняются. Мы их пишем в тетрадях и на доске.

Реально мы можем оформить эти фрагменты в алгоритмы после прохождения § 22, который именно этому вопросу и посвящен. Информационная модель исполнителя и есть набор алгоритмов, работающих с общей информационной моделью. Но это — тема следующего параграфа.

Вы можете пойти по нашему пути: пишем фрагменты, а почему — объясним потом. Можете сразу объяснить проблему, как я ее только что вам описал, и сказать, что именно поэтому мы пишем фрагменты, а как выходить из этой ситуации, разберемся потом. Наконец, можете перемешать материал § 21 и § 22, введя сразу конструкцию **исп...кон**, а потом решать задачи, уже имея все необходимые средства алгоритмического языка.

Я еще раз обращаю ваше внимание, что мы пишем только фрагменты алгоритмов без начала и без конца. В § 21 по-другому нельзя. И если кто-то из учеников начинает писать **алг, дано, надо, нач** (что вполне естественно для них к этому времени), то я рекомендую просто сказать, что пока этого писать не надо, а почему не надо, разберемся в следующем параграфе.

Вот, собственно, и все про информационные модели. Понятие введено. Примеры вы посмотрите в учебнике — они все очень простые.

Итак, что же такое информационная модель? Если есть некоторая реальность, с которой мы собираемся работать, то мы придумываем величины алгоритмического языка, которые эту реальность отражают. Точнее, отражают то, что нам нужно для решения задач. Такое придумывание всегда проводится в рамках некоторого класса задач.

В примере выше мы не рассматривали задачу уборки кинозала и поэтому не интересовались величиной проходов (что может оказаться критически важным для машинной уборки зала). Мы также не рассматривали задачу продажи билетов на соседние места и потому не интересовались расположением проходов. Мы собирались продавать билеты, так сказать, по одному и потому отразили кинозал в виде модели М1. Если бы мы рассматривали другие задачи, например, управление системой вентиляции кинозала, в зависимости от того, на какие места проданы билеты, то у нас, по-видимому, получилась бы совсем другая информационная модель.

Дальше в параграфе начинается наша "черепаха": чтобы усвоить материал, школьники должны много работать, и поэтому дальше идет просто модель за моделью для разных объектов и задач.

Следующая модель — модель транспортной сети. 100 городов связаны между собой воздушными линиями так, что из любого города в любой другой можно добраться, хотя бы и с пересадками. Какая тут может быть информационная модель? Правильно, пока — никакой. Потому что, напоминаю, без формулировки класса задач, которые мы собираемся решать, никакую модель построить нельзя. Нас интересует класс задач, связанный с путешествием из одного города в другой, с выбором маршрута. Скажем, как добраться из Москвы в Париж с минимальным числом пересадок? Или есть ли маршрут из Парижа в Архангельск не более чем с одной пересадкой? Кроме того, нас будет интересовать длина маршрута (расстояние). Например, если мы летим из Москвы в Париж по выбранному маршруту, то сколько километров мы при этом пролетим. Вот класс задач, исходя из которого мы будем строить модель.

После того как класс задач сформулирован, можно строить информационную модель. Мы построили модель М2 (с. 171 учебника), я ее повторять не буду. Важно лишь понимать, что здесь нет никакой единственности — можно построить много разных "правильных" информационных моделей. М2 — одна из них.

Далее, как всегда, можно решать разные задачи, т.е. писать разные фрагменты обработки информации этой информационной модели. У нас приведен фрагмент алгоритма, который ищет, есть ли из одного города в другой маршрут не более чем с одной пересадкой. И опять фрагменты алгоритмов получаются достаточно простые. Вы можете усложнять задачи, например, попросить найти все маршруты из одного города в другой не более чем с одной или двумя пересадками. И т.п.

Я лишь хочу обратить ваше внимание, что информация о расстояниях в приведенном в учебнике фрагменте не задействована. Но можно загрузить сильных учеников вполне содержательной задачей "найти длину кратчайшего маршрута" между двумя заданными городами. При всей простоте модели эта задача уже достаточно сложна.

Примеры информационных моделей, которые идут дальше, призваны продемонстрировать, что информационную модель можно построить почти для чего угодно.

Информационные модели геометрических объектов.

В разделе 21.4 приводятся информационные модели для точки, отрезка, прямоугольника, квадрата, окружности и т.д.

На примере информационной модели точки на плоскости очень хорошо демонстрировать разницу между математической и информационной моделями. "Точка на плоскости" — геометрическое понятие. Ее можно задать координатами — и это будет алгебраическое (математическое) представление (или модель) точки. А вот если мы напишем **вещ** **x, y**, т.е. две вещественные величины алгоритмического языка, то получим информационную модель точки. Точка в математическом смысле — это одна точка, конкретная пара чисел — координат точки. Информационная модель точки — это две величины, значения которых могут меняться с течением времени, в ходе исполнения алгоритма. Значения "алгоритмической" точки могут меняться. Информационной моделью точки называются сами величины **x** и **y**, а не их значения.

И еще. Координаты точки можно задать в виде **вещ таб** **x [1:2]** (модель М13) — и это будет *другая* информационная модель. Обратите внимание: с математической ...

бодны, а также задаются цены билетов. После этого один за другим пишутся все алгоритмы — все команды исполнителя. Завершает все служебное слово **кон** (конец исполнителя). Это та самая новая конструкция алгоритмического языка (четвертое фундаментальное понятие алгоритмизации), которая за последние 20 лет (по-разному называемая) появилась во всех современных языках программирования.

Можно представлять себе конструкцию **исп...кон** образно, в виде ромашки (*рис.* 12). Сердцевина — информационная модель, набор величин и задание их начальных значений. Лепестки — алгоритмы, работающие с величинами данной модели.

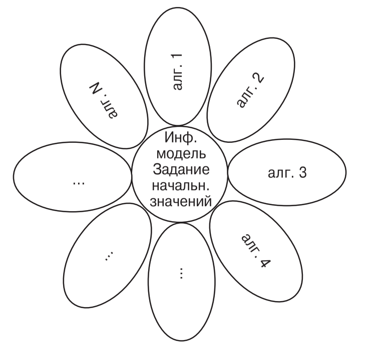


Рис. 12

Величины информационной модели принято называть ***общими величинами исполнителя***. Это новое понятие, новый ***вид*** величин алгоритмического языка: это и не аргументы, и не результаты, и не промежуточные величины.

Обратите внимание, что я уже перестал говорить "информационная модель исполнителя" и говорю просто "исполнитель". Давайте я еще раз поясню, почему это так, почему мы не делаем между этими понятиями никакой разницы. Т.е. почему мы не делаем разницы между внешним исполнителем ("в железе"), подключенным к ЭВМ, и информационной моделью исполнителя, обрабатываемой внутри ЭВМ.

Вспомните: на протяжении всей первой главы мы рассматривали Робота как внешнего исполнителя, существующего "в металле", подключенного к ЭВМ. И мы писали алгоритмы по управлению этим исполнителем, считая Робота "внешним". Лишь совсем недавно я вам признался, что ***на самом деле*** мы работали с информационной моделью исполнителя Робот, входящей в состав системы КуМир. Мы считали Робота "внешним", и это нам совсем не мешало. Наоборот, мы могли не задумываться о том, как Робот "реализован", какими величинами задаются стены, какими величинами задается положение Робота и пр. Очень трудно, почти невозможно думать одновременно и над тем, как составить алгоритм по управлению Роботом, и над тем, как это все внутри информационной модели Робота будет выполняться.

Поэтому первое утверждение звучит так: человеку при решении задач ***удобно*** представлять себе исполнителей как внешних, подключенных к ЭВМ, даже если они ***на самом деле*** всего лишь информационные модели, обрабатываемые внутри ЭВМ.

В чем, собственно, разница между этими ситуациями, если говорить об использовании исполнителей, об управлении ими? В случае "внешнего" исполнителя ЭВМ, выполняя написанный нами алгоритм, командует "настоящему" Роботу, который исполняет команды. В случае информационной модели ЭВМ, выполняя написанный нами алгоритм, вызывает соответствующие алгоритмы информационной модели Робота ("команды" исполнителя Робот) как вспомогательные, т.е. сама же их и исполняет.

Вся разница в том, как и кем исполняются "команды" Робота. Но сам алгоритм управления Роботом в обоих случаях ***абсолютно один и тот же***. При составлении этого алгоритма совершенно не важно, есть ли Робот на самом деле или он будет имитироваться (моделироваться) с помощью ЭВМ. Именно этим мы и воспользовались, когда говорили, что пишем алгоритмы управления "настоящим" Роботом, а сами выполняли алгоритмы и наблюдали за полем Робота на экране ЭВМ.

А поскольку представлять себе "внешних" исполнителей проще, чем информационные модели, и для "внешних" исполнителей можно вообще не думать, как они устроены внутри, мы и используем более простое слово "исполнитель" при управлении как "внешними", так и "информационными", "внутренними" исполнителями. Это первое.

Кстати, для задания информационной модели исполнителя часто используется термин "реализация исполнителя". "Реализация исполнителя" — это текст на алгоритмическом языке от **исп** до **кон**. Так, например, на с. 227 приведена ***реализация исполнителя Кинозал***. В учебнике (с. 180) ***показана реализация исполнителя Маленький Робот***.

И второе, самое важное. Это понятие информационной модели исполнителя, эта конструкция алгоритмического языка **исп...кон**, это четвертое фундаментальное понятие алгоритмической культуры может (и должно) использоваться нами как ***инструмент*** алгоритмизации, как способ структуризации нашего мышления при решении алгоритмических задач. Основная ценность понятия исполнителя состоит в том, что это — способ структуризации мышления, способ накопления алгоритмических "знаний". Ученикам достаточно овладеть конструкцией **исп...кон** и научиться ее применять, что достаточно просто. Вы же, как учителя и методисты, должны еще и понимать место и роль этого понятия в информатике в целом. И именно поэтому я все время повторяю про четвертое фундаментальное понятие алгоритмической культуры.

ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ. Скорее всего вы уже слышали про объектно-ориентированные языки программирования. В этих языках наше четвертое фундаментальное понятие алгоритмической культуры, понятие объекта (в нашей терминологии — исполнителя) поставлено на первое место. И успех объектно-ориентированных языков, или,

действия? Действие (команду) “вправо” лучше всего отразить алгоритмом — мы это уже знаем, мы уже писали **x:= x + 1**. Но до сих пор у нас все было кусочками и по отдельности. Отдельно — информационная модель М16. Отдельно — команда **x:= x + 1**. Как это все собрать воедино? Как сказать, что **x:= x + 1** — это и есть команда **вправо** в нашей модели?

Подойдем к этому как к информационному моделированию некоторой реальности: есть исполнитель Робот; как промоделировать его целиком вместе с обстановкой (что уже было) и с командами (чего пока не было)?

Второй момент — чисто прагматический, или, если хотите, технический. Даже для модели кинозала нам пришлось писать фрагменты алгоритмов, потому что было непонятно, что делать с информационной моделью, куда помещать ее величины. Что же касается модели М16 для поля Робота, то очевидно, что повторять описание всех величин модели в качестве аргументов и результатов алгоритмов “вправо”, “влево” и т.д. — просто какое-то безумие. Поэтому нам нужны такие новые конструкции языка программирования, которые позволят разным алгоритмам работать с одними и теми же величинами. Чтобы значения этих величин не пропадали от вызова одного алгоритма до вызова другого. Чтобы алгоритм, считающий выручку в модели кинозала, учел результаты работы и алгоритма по продаже билетов, и алгоритма по возврату билетов. Вы можете также повторить здесь аналогию со строительными бригадами (с. 26), которая является абсолютно точной.

Для задания группы некоторых общих величин и группы алгоритмов, работающих с использованием этих величин, т.е. для задания и информационной модели, и алгоритмов обработки информации в этой модели, в алгоритмическом языке есть специальная конструкция **исп...кон**, называемая информационной моделью исполнителя, или просто исполнителем. Это и есть то четвертое фундаментальное понятие алгоритмизации, то четвертое фундаментальное понятие информационной культуры, о котором я говорил в самом начале.

Обратите внимание, что информационная модель исполнителя — это новое понятие, не укладывающееся в понятие информационной модели как набора величин. Точнее, информационная модель исполнителя — более сложное понятие, содержащее внутри себя и набор величин (информационную модель в смысле § 21 — модель “обстановки” исполнителя), и алгоритмы обработки этих величин (“команды” исполнителя). Именно поэтому понадобилась и новая конструкция алгоритмического языка **исп...кон** (“исполнитель ... конец исполнителя”).

Используя это понятие, мы можем от записи фрагментов перейти к полной записи алгоритмов и (впервые!) к полной ***записи исполнителей на алгоритмическом языке***. Рассмотрим, например, исполнителя Кинозал. Обратите внимание: не “информационная модель”, а исполнитель Кинозал. В чем разница? “Информационная модель кинозала” — это набор величин алгоритмического языка (М1). Исполнитель Кинозал — это и величины модели М1, и алгоритмы ее обработки — те самые алгоритмы, фрагменты которых мы приводили в предыдущем параграфе, но не могли записать.

Исполнитель Кинозал на алгоритмическом языке можно записать так:

```
исп Кинозал
| лог таб продано [1:20,1:28]
| |продано [i,j] ⇔ I место i ряда j продано
| цел таб цена[1:28]
| |цена[j]=цена билета в j-м ряду
| цел i, j
| нц для j от 1 до 28 |для каждого ряда
| | нц для i от 1 до 20
| | | |для каждого места в ряду
| | | продано [i, j]:= нет
| | |       |запомнить, что место свободно
| | кц
| | выбор
| | | при j< 5 : цена [j]:= 1000
| | | |установить цену билета в
| | | |зависимости от номера ряда
| | | при j<10 : цена [j]:= 3000
| | | при j<20 : цена [j]:= 5000
| | | иначе : цена [j]:=8000
| | все
| кц

| алг цел число проданных
|   надо  |знач=число проданных билетов
| нач
| | знач:= 0
| | нц для j от 1 до 28 |для каждого ряда
| | | нц для i от 1 до 20
| | | |                 |для каждого места в ряду
| | | | если продано [i, j]
| | | |  | то знач:= знач + 1
| | | все
| | кц
| кц
| кон

| алг цел выручка
|   надо  |знач = выручка от продажи билетов
| нач
| | знач:= 0
| | нц для j от 1 до 28 |для каждого ряда
| | | нц для i от 1 до 20
| | | |                 |для каждого места в ряду
| | | | если продано [i, j]
| | | | | то знач:= знач + цена [j]
| | | все
| | кц
| кц
| кон
| ...(остальные алгоритмы исполнителя)...
кон
```

После строки **исп** (исполнитель) задается информационная модель кинозала (модель М1), т.е. описываются величины алгоритмического языка. Потом идет текст на алгоритмическом языке, задающий начальные значения величин модели. В нашем примере в качестве начальных значений запоминается, что все места свободны.

матической точки зрения ничего не изменилось — точка по-прежнему задается той же парой своих координат. Но теперь вместо двух величин у нас таблица из двух элементов, т.е. ***другой*** набор величин алгоритмического языка, ***другая*** информационная модель.

Из информационных моделей геометрических объектов следует обратить внимание на информационную модель квадрата (М11). Задав квадрат в виде координат центра (a, b) и координат вектора (u, v) из центра в одну из вершин, легко выразить и координаты вершин квадрата: это будут $(a + u, b + v)$, $(a + u, b - v)$, $(a - u, b + v)$, $(a - u, b - v)$.

Конечно, геометрические объекты — это не объекты из реальной жизни. Поэтому мы построили их информационные модели без труда, не уточняя “класс задач”. Точнее, эти модели годятся для любых геометрических задач. Вот если бы мы рассматривали не геометрические, а “реальные” точки, то нам понадобилось бы уточнить, что мы собираемся решать геометрические задачи и потому размеры точки, чем и на чем она нарисована, какого она цвета и пр. нас не интересуют. Однако эту часть работы за нас выполнил Евклид (и его предшественники). Евклид в начале своего курса геометрии объяснил, что у точки нет никакого размера и никаких различимых частей. Этому мы и следовали, создавая информационную модель точки.

И, как всегда, когда модели построены, можно решать разнообразные задачи, писать соответствующие фрагменты алгоритмов. В учебнике приводится фрагмент для нахождения периметра n-угольника, алгоритмическое условие (формула), соответствующая геометрическому высказыванию “окружность, заданная моделью М4, находится внутри прямоугольника, заданного моделью М10”, и т.п. Я еще и еще раз подчеркиваю, что всюду в этих фрагментах мы используем величины модели, получаем формулы и алгоритмы независимо от значений этих величин. Значения могут меняться в ходе выполнения алгоритма, и это будет означать, что в ходе выполнения алгоритма обрабатываются ***разные*** геометрические объекты, заданные ***одной и той же*** информационной моделью. Это связка геометрических понятий и информационных моделей.

Эта часть параграфа, начиная с модели кинозала и кончая моделями геометрических объектов, чрезвычайно проста. За исключением собственно понятия информационной модели, в ней нечего понимать и объяснять. А вот дальше происходит некоторое усложнение, которое ученые назвали бы “метапереходом”.

По сути дела, материал и дальше ничуть не сложнее “кинозала”. Вся сложность состоит в том, чтобы преодолеть инерцию мышления и взглянуть на то, что мы уже проходили раньше, с совершенно другой точки зрения. Ибо дальше строятся информационные модели того, к чему школьники уже привыкли, — информационные модели обстановки на поле Робота и даже информационные модели алгоритмов! “Алгоритмы обработки информации для информационных моделей алгоритмов управления Роботом” — тут, конечно, голова кругом может пойти. Поэтому давайте разберем это постепенно, шаг за шагом.

Вспомните вещественную таблицу с информацией об уровнях радиации в коридоре (А62). Так вот вещественная таблица **a[1:n]** — это, как мы теперь знаем, информационная модель коридора для класса задач, в которых нам важна только радиация и ничего больше. Если, однако, мы захотим решать задачи типа “определить количество боковых выходов из коридора”, то нам понадобится более сложная информационная модель коридора, чтобы мы смогли задать и информацию о стенках тоже. Такая полная модель коридора приведена в учебнике на с. 174 (модель М15). Слово “полная” означает, что мы можем решать любые задачи про Робота в коридоре. Конечно, такие “полные” модели возможны только если моделируемые объекты не из реальной жизни, а сами в каком-то смысле являются моделями (как геометрические объекты или поле Робота).

Дальше происходит переход от модели “Робот в коридоре” к модели “Робот на ограниченном поле размером 16 на 32 клетки” (модель М16). Никакого принципиального усложнения здесь нет. Но в этот момент, и, по-моему, это очень важно, можно отвлечься от темы информационных моделей и объяснить школьникам, как они работают с Роботом на компьютере. Мы все время говорили о Роботе как об устройстве, действительно существующем. Однако при работе на компьютере в системе КуМир на экране ***изображается*** небольшое по размерам поле, ***изображаются*** стенки на поле, сам Робот и пр. Мы ***воображаем***, что Робот существует, что он как-то подключен к ЭВМ, а ***на самом деле*** наблюдаем все это на экране. Как это получается? А ответ очень прост: в действительности мы работаем не с Роботом, а с его ***информационной моделью***, которая и изображается на экране компьютера.

Здесь происходит раскрытие того инструмента (системы КуМир), с которым работают школьники. Раньше мы говорили просто “Робот”, да и не могли сказать по-другому. А теперь говорим, что на самом деле вы работали не с Роботом, а с информационной моделью, т.е. просто с набором величин алгоритмического языка. И вы можете ***сами*** написать, что значит “вправо” (в модели М16 это команда **x: = x + 1**), что значит “закрасить”, и т.п. Все эти “команды Робота” превращаются просто во фрагменты алгоритмов, использующих и ***меняющих*** состояние информационной модели М16. Здесь, конечно, ничего не сказано про изображение модели М16 на экране. Но все остальное представлено и целиком соответствует реальности.

Это чрезвычайно глубокое и важное место, я к нему еще вернусь. А сейчас запомните, пожалуйста, хорошенько, что мы все время воображали Робота существующим, а работали с его информационной моделью. И это было удобно. И облегчало составление алгоритмов управления Роботом, поскольку позволяло вообще не думать, как Робот работает и как он устроен внутри.

Повторю, что материал этот очень простой, совсем как “кинозал”, и вся сложность состоит в новом взгляде на старое и давно знакомое понятие. Если бы мы взяли не Робота, а другого исполнителя, например, “Уборщика кинозала”, то не было бы никаких проблем. А когда мы берем понятие, к которому уже давно привыкли, с которым много работали, и пытаемся представить его совсем по-другому, то это немного сложно. Сложно отказаться от уже привычного и взглянуть на это понятие по-новому. Хотя это трудности скорее психологического плана, чем содержательного.

Информационные модели алгоритмов.

Но модель для Робота и его поля — это еще “цветочки”. “Ягодки” — это модели для алгоритмов управления Роботом. Ибо кинозал и поле Робота — объекты сходной природы. Алгоритм — что-то совсем другое. И следующий раздел должен продемонстрировать, что информационные модели — универсальное средство, информационно моделировать можно все что угодно. Это очень важный момент. Информационную модель можно построить для любого объекта, для любой реальности, ибо информация присутствует всюду. И алгоритм — тоже объект, содержащий какую-то информацию. А значит, его можно задать какой-то информационной моделью, обработать эту модель и т.п.

Но для этого нужно совершить “метапереход”: если раньше объектом был Робот на поле и мы составляли алгоритмы по управлению этим объектом, то теперь сам такой алгоритм мы должны рассмотреть как объект, задать его какой-то моделью и составлять алгоритмы по обработке этой модели, т.е. алгоритмы, обрабатывающие алгоритмы.

В учебнике для этого используется кодировка простейших алгоритмов управления Роботом с помощью букв “В”, “Н”, “Л”, “П”, “К” (см. главу 2). Я вам напоминаю, что если вы раздел 2 пропускаете, то в этом месте надо вместо ссылки на главу 2 ввести эту кодировку непосредственно, благо это очень просто. Величина **лит t** из алгоритма А91 главы 2 — это и есть информационная модель (М17) алгоритмов из соответствующего класса, т.е. алгоритмов, состоящих только из команд **вверх**, **вниз**, **вправо**, **влево**, **закрасить**. (Каждая команда кодируется одной буквой, а весь алгоритм — последовательность команд — литерной строкой, состоящей из этих букв.)

Как всегда, когда информационная модель задана, можно решать разные задачи по обработке модели. Например: “Определить, на сколько клеток по горизонтали и вертикали от начального положения сместится Робот в результате выполнения алгоритма, заданного моделью М17”. Это простая задача. Надо завести величины для подсчета смещения, например, **цел x, y: x** — смещение по горизонтали, **y** — по вертикали. Вначале присвоить **x := 0; y := 0**, а потом анализировать модель М17 (**лит t**) буква за буквой и на каждое “В” (вверх) прибавлять единицу к **y**; на “Н” (вниз) — вычитать единицу из **y**; на “П” (вправо) — прибавлять единичку к **x**, а на “Л” (влево) — вычитать единицу из **x**. В итоге у нас в **x, y** получится искомое смещение конечного положения Робота от начального — если выполнить алгоритм, заданный строкой **t** (модель М17).

Заметьте, что я вам описал простенький алгоритм (вы легко можете сами записать его целиком), который по состоянию модели М17 вычисляет, на сколько сместится Робот. Хотя нет ни самого Робота, ни алгоритма по управлению Роботом на алгоритмическом языке, ни процесса его выполнения. Мы просто анализируем какую-то информацию, строку из букв и вычисляем смещение Робота, т.е. работаем с ***информационной моделью алгоритма по управлению Роботом.***

Или более сложная задача — определить, сколько клеток закрашивает Робот (имея в виду, что если одна и та же клетка красится несколько раз, то ее надо учесть только единожды). Упрощенный вариант этой задачи приведен в упр. 18 к параграфу.

Еще раз обращаю ваше внимание, что при решении этих задач алгоритм, заданный моделью М17, лишь анализируется, но отнюдь не исполняется! Никаких команд Роботу вообще не отдается. Мы лишь смотрим, какие буквы записаны в строке, и что-то в каких-то величинах считаем!

Вся сложность здесь — в переходе: мы привыкли, что объектом всегда был Робот, но не алгоритм. Алгоритм — это то, что мы писали, а не то, над чем работали. А тут все меняется местами: алгоритм становится объектом. Робот вообще превращается в элемент описания, поскольку ему никто никаких команд уже не отдает, — он лишь “имеется в виду”, придает некий воображаемый смысл командам, а мы пишем алгоритм по обработке алгоритма.

Здесь важно, что какой бы объект мы ни рассматривали” — будь то кинозал, обстановка на поле Робота, алгоритм управления Роботом или что-то еще, — на все можно смотреть с информационной точки зрения. Мы можем закодировать величинами алгоритмического языка (т.е. представить в виде информационной модели) информацию и о кинозале, и об обстановке на поле Робота, и об алгоритме. Для любой реальности, любой действительности можно построить информационную модель. И, кстати, информационная модель — это тоже информация, и для нее тоже можно построить информационную модель информационной модели. И т.д.

Но если вы считаете алгоритмы обработки алгоритмов чересчур сложными для восприятия, можете их опустить (вместе со смыслом понятия интерпретация) либо ограничиться изложением моделей для алгоритмов управления Чертежником (п. 21.6), которые, на мой взгляд, психологически воспринимаются проще, чем модели для алгоритмов управления Роботом. Соответствующая модель (М18), приведенная на с. 175 учебника, очень проста, и я ее повторять не буду.

Обратите внимание на алгоритм А92 (с. 176 учебника), который, анализируя состояние модели, выдает соответствующие команды Чертежнику. Это еще один пример программы-интерпретатора, которая, анализируя некоторую информацию, некоторое представление алгоритма, этот “закодированный” алгоритм выполняет. В рамках связи с главой 2 и устройством ЭВМ обратите внимание на замечание на с. 176 учебника непосредственно перед упражнениями. Индекс **i** в ходе исполнения алгоритма перемещается от одной команды Чертежника к другой точно так же, как счетчик команд перемещается от одной машинной команды к другой, каждый раз показывая, какую следующую команду надо выполнить. Здесь происходит еще одна связка с устройством ЭВМ, поскольку интерпретация алгоритмов и выполнение машинных программ в общем очень похожи друг на друга.

И последнее. Я уже говорил, что при работе в системе КуМир школьник видит на экране поле Робота и вообще работает не с “настоящим” Роботом, а с его информационной моделью. Но ведь школьник видит на экране и свой алгоритм по управлению Роботом, а компьютер, как мы говорили, всегда выполняет некоторую машинную программу. Какова же связь между изображаемым на экране алгоритмом и работой компьютера? Как вообще все происходит ***на самом деле***? На мой взгляд, глубокое реальное понимание этих процессов возможно только после изучения главы 2, информационных моделей, понятия информационной модели алгоритма и понятия программы-интерпретатора. Именно поэтому, несмотря на некоторые сложности восприятия (повторю, при простом по сути материале), эти вопросы попали в школьный учебник как необходимый элемент демистификации ЭВМ и формирования адекватного представления о современной информационной реальности. Мы обязаны хотя бы в принципе сформировать у школьников адекватное представление о том, как ***на самом деле*** работает ЭВМ.

Резюме.

Итак, резюме по этому параграфу. Во-первых, материал параграфа очень прост. Для изложения понятия информационной модели вообще достаточно кинозала. Все возможные сложности связаны не с понятием информационной модели как таковой, а с приложением этого понятия к новым объектам: обстановке на поле Робота и особенно к алгоритмам.

Во-вторых, с методической точки зрения крайне важно понимание универсальности информационного моделирования. Если это будет понято, то станет понятным фактически все: почему на экране изображается лабиринт (потому что это — изображение информационной модели, с которой работает ЭВМ), как ЭВМ выполняет алгоритм (она строит внутри себя информационную модель алгоритма и потом его “интерпретирует”, т.е. выполняет). Здесь же следует повторить про начальную загрузку, операционную систему и пр. из § 20.

Если это не будет понято, то останется пробел — не в понятии информационной модели, а в общей картине мира, в представлении о том, как на самом деле работает компьютер. И место отсутствующих знаний могут занять мифы, формирующие неадекватное, неверное представление о реальности.

Упражнения после параграфа можно разбить на три класса:

1) в рамках заданной информационной модели написать фрагмент какого-то алгоритма (типа “продать три места в одном ряду” в модели кинозала);

2) изменить информационную модель так, чтобы учитывалась какая-то дополнительная информация, чтобы можно было решать более широкий класс задач (скажем, продавать места на два сеанса или еще что-нибудь);

3) придумать “с нуля” информационную модель. Например, информационную модель прямоугольной пирамиды, расписания уроков, паспорта или свидетельства о рождении. В последних задачах имеется в виду обработка информации, ***записанной*** в паспорте или свидетельстве о рождении (т.е. внешний вид, толщина бумаги, наличие жирных пятен и пр. для нас не важны). Сложность модели будет зависеть от детальности представления информации и ваших требований. Например, если для информационной модели камеры хранения на вокзале вы потребуете, чтобы через трое суток хранения ячейка помечалась как просроченная (гудела и не открывалась), то модель будет чуть сложнее, чем если вы такого условия ставить не будете.

Я вам рекомендую использовать упражнения разных типов. Кроме того, вы без труда можете дополнить приведенный после параграфа набор упражнений своими, например, попросить учеников составить алгоритмы для выполнения простейших геометрических преобразований над теми или иными объектами планиметрии (модели М3—М11): симметрия относительно координатных осей, параллельный перенос, поворот на 90 градусов, произвольный поворот, гомотетия относительно начала координат и пр.

§ 22. Информационные модели исполнителей, или Исполнители в алгоритмическом языке

В начале изложения этого материала необходимо отметить два момента.

Первый — исходящий из фундаментальной сущности информационного моделирования. Поскольку информационные модели можно строить для чего угодно, давайте построим информационную модель Робота как исполнителя целиком, т.е. вместе со всеми его командами.

Раньше мы строили информационные модели только для данных: кинозал, города, расстояния между ними, фигуры на плоскости и пр. (И даже алгоритмы в п. 21.6 мы рассматривали как данные, как объекты.) А теперь давайте рассмотрим исполнителя Робот целиком. У него, кроме данных, обстановки, поля, существуют еще и команды: он умеет ходить вправо и влево, закрашивать клетки, сообщать температуру и радиацию и пр., т.е. совершать ***действия.*** Как нам отразить, как задать эти

Решение задачи X международной олимпиады по информатике "Камелот"

Е.В. АНДРЕЕВА

Условие

В незапамятные времена король Артур и рыцари Круглого стола собирались обычно в канун Нового года, чествуя свой дружеский союз.

В память об этом придумана настольная игра "Камелот" для одного игрока, в которой фигура короля и несколько фигур рыцарей произвольно располагаются в различных клетках доски.

Доска имеет размер 8×8 клеток (см. *рис.* 1).

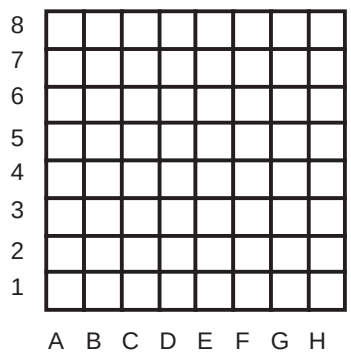


Рис. 1. Доска

Король может перемещаться в любую смежную клетку доски из ● в ○, как показано на *рис.* 2, при этом не выходя за пределы доски.

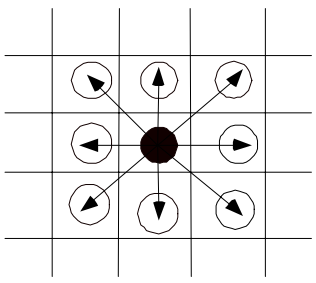


Рис. 2. Все возможные перемещения короля

Рыцарь может перемещаться из клетки ● в одну из клеток ○, как показано на *рис.* 3, при этом не выходя за пределы доски.

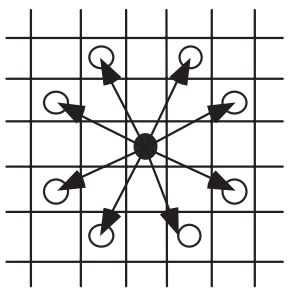


Рис. 3. Все возможные перемещения рыцаря

Во время игры игрок может поместить более одной фигуры в одну клетку. Клетки считаются достаточно большими, и не возникает препятствий для свободного перемещения фигур.

Игроку необходимо так перемещать фигуры короля и рыцарей, чтобы собрать их всех в какой-то одной клетке за наименьшее число ходов. Ходы фигурами необходимо делать по правилам, описанным выше. Дополнительно к этому, если король и один или более рыцарей находятся в одной клетке, игрок может короля и одного из рыцарей переместить вместе по правилам перемещения Рыцаря и считать это одним ходом.

Задача. Напишите программу для вычисления минимального количества ходов, необходимых для перемещения всех фигур в одну клетку доски.

Входные данные

Файл CAMELOT.IN содержит одну строку символов, описывающую начальное расположение фигур на доске. Строка содержит последовательность клеток доски, первая из которых — клетка короля, остальные — клетки рыцарей. Каждая клетка описывается парой буква — цифра. Буква обозначает горизонтальную, а цифра — вертикальную координату клетки доски. Все фигуры в начале игры расположены в разных клетках.

Пример входного файла:

```
D4A3A8H1H8
```

Король расположен в клетке `D4`. Четыре рыцаря — в клетках `A3`, `A8`, `H1` и `H8`.

Выходные данные

Файл CAMELOT.OUT должен содержать единственную строку с положительным целым числом, обозначающим минимальное число ходов игрока, необходимых для перемещения всех фигур в одну клетку доски.

Пример выходного файла:

```
10
```

Ограничения

$0 \le$ количество рыцарей ≤ 63

Решение

На первый взгляд количество фигур на доске и неопределенность конечной клетки игры делают описанную в задаче игру не менее сложной, чем шахматы. Однако, к счастью, это не так. Действительно, предугадать заранее, какая клетка доски окажется местом сбора рыцарей и короля, невозможно. Поэтому мы сначала для каждой клетки решим задачу подсчета минимального количества ходов, требующихся для перемещения всех фигур в нее, а затем выберем из всех клеток наилучшую.

Для ускорения решения первой из указанных подзадач нужно для всех пар клеток заранее подсчитать минимальное количество ходов, необходимых для перемещения как рыцаря, так и короля из одной клетки в другую. Результаты этих вычислений будем запоминать в массивах `k1` и `king` соответственно. Тип этих переменных в языке программирования Паскаль можно описать как `array['A'..'H','1'..'8','A'..'H','1'..'8'] of byte`.

Здесь первая пара индексов обозначает начальное расположение фигуры, а вторая — конечное. Первоначально массив `k1` заполняется следующим образом: если рыцарь может переместиться из одной клетки в другую за один ход, то соответствующий элемент массива равен 1 (для рыцаря это условие выполняется тогда и только тогда, когда произведение модуля разности между вертикалями двух точек на модуль разности между горизонталями равно двум), в противном случае — "бесконечности", которая в нашем случае является любым числом, большим 63, например, 100 (так как больше чем за 63 хода из одной клетки в другую, не попадая дважды в одну и ту же клетку, рыцарь переместиться не может); значение же "диагональных" элементов матрицы, соответствующих совпадению начальной и конечной клеток, равно нулю (ходов в этом случае рыцарю делать не придется). Заполненная таким образом матрица называется в терминах теории графов матрицей смежности (вершинами графа в данном случае являются все 64 клетки доски, а ребрами — возможность перемещения из одной вершины графа в другую).

Поиск длины кратчайших путей между всеми парами клеток доски наиболее просто осуществить с помощью алгоритма Флойда. Основная идея алгоритма заключается в следующем. Пронумеруем все клетки доски от 1 до 64. Пусть мы уже знаем минимальное количество перемещений рыцаря для достижения им клетки с номером `j` из клетки с номером `i`, используя в качестве промежуточных только клетки с номерами от 1 до `m-1`. Обозначим это число $k^{m-1}[i,j]$ ($k^0[i,j]$ является матрицей смежности). Если возможность использования в качестве промежуточной вершины еще и вершины с номером `m` кратчайшего пути не изменит, то $k^m[i,j]=k^{m-1}[i,j]$, в противном случае кратчайший путь будет эту вершину содержать и его можно разделить на две части: от `i` до `m` и от `m` до `j`. Таким образом,

$$k^m[i,j]=\min\{k^{m-1}[i,j],k^{m-1}[i,m]+k^{m-1}[m,j]\}.$$

Трудоемкость работы алгоритма в нашем случае составляет 64^3 (в общем случае — N^3, где N — количество вершин в графе). Заполнение матрицы смежности `k1` и реализация алгоритма Флойда произведены в процедуре `Distance` приведенной ниже программы, только единый номер каждой клетки (вершины графа) заменен на два — название горизонтали и вертикали клетки. В этой же процедуре матрица `king` заполняется сразу как матрица минимального количества ходов короля при перемещении из одной клетки в другую. Для каждой пары клеток это число равно максимуму из модулей разности между вертикалями этих клеток и разности между их горизонталями.

Если бы в задаче отсутствовало условие возможности одновременного перемещения короля с одним из рыцарей, то решение поставленной задачи для каждой из конечных клеток `(c, i)` состояло бы в суммировании элементов матрицы `k1`, соответствующих начальному положению каждого из `n` рыцарей, — `(knight[h][1], knight[h][2])` и клетке `(c, i)` с `king[knight[0][1], knight[0][2],c,i]`, где `knight[0][1]` и `knight[0][2]` описывают начальное расположение короля. Сокращение полученного числа может быть произведено в следующих случаях:

1) если кратчайший путь одного из рыцарей из начальной точки в конечную проходит через начальное местоположение короля, то из полученной суммы самостоятельный путь короля вычитается и задача для конкретной конечной клетки является решенной;
2) в противном случае можно попытаться найти такую клетку, в которой король и один из рыцарей могли бы встретиться на пути к конечной клетке и дальше путешествовать совместно, уменьшая при этом общее количество ходов в игре. Для простоты реализации алгоритма в качестве клетки их встречи испытаем все 64 клетки доски. Трудоемкость алгоритма при этом составит 64^2 для каждого из рыцарей, или $n \cdot 64^2$ для всех рыцарей, что не превышает времени выполнения уже реализованного алгоритма Флойда.

Действия, описанные в этом абзаце, производятся в процедуре `Solve`. В конце данной процедуры из всех возможных конечных клеток ищется наилучшая (число ходов в игре для которой оказалось минимальным). Поставленная задача, таким образом, оказывается полностью решенной.

Приведем теперь полностью текст программы, решающей задачу "Камелот".

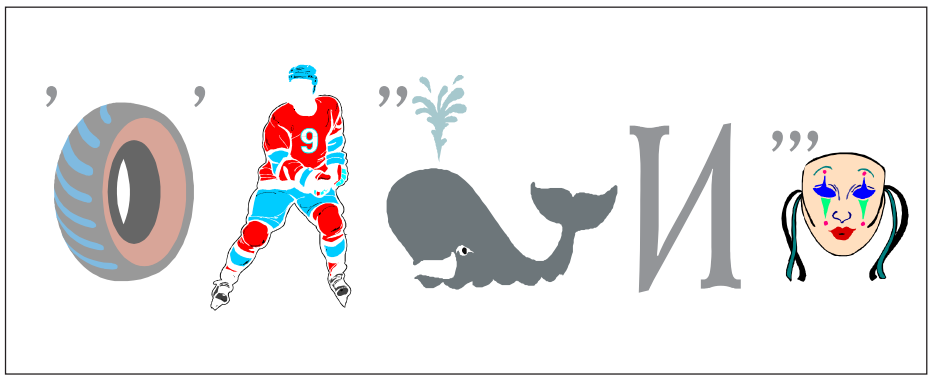
Окончание на с. 14

Решение задачи X международной олимпиады по информатике “Камелот”

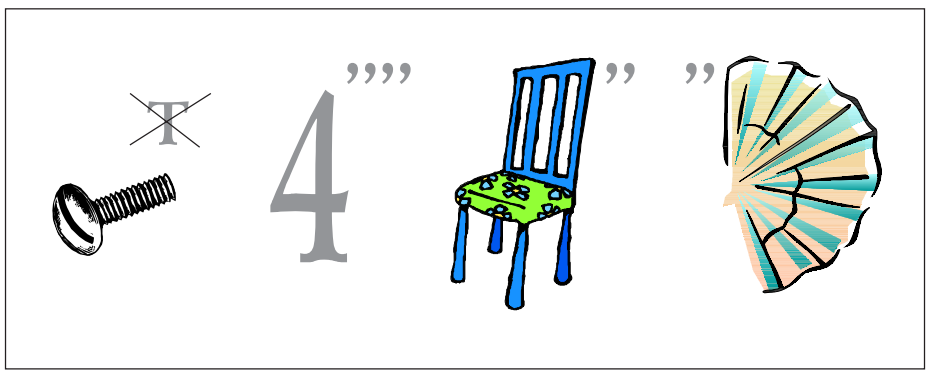
Окончание. Начало на с. 13

```
type
 aa=array['A'..'H','1'..'8','A'..'H','1'..'8'] of byte;
var
  knight:array[0..63] of array[1..2]of char;
  king,k1:aa;
  res,r:array['A'..'H','1'..'8']of integer;
  n,min,h,rr:integer;
  c1,c2,c3,i,j,m:char;
procedure ReadData;
begin
  assign(input,'camelot.in');
  reset(input);
  n:=-1;{n - количество рыцарей, n=0 - одинокий король}
  while not eoln do {пока не достигнут конец строки}
  begin
    n:=n+1;
   {считывание расположения одной фигуры}
   read(knight[n][1],knight[n][2]);
    knight[n][1]:=UpCase(knight[n][1])
  end;
end;
procedure Distance;
begin
  {заполнение матрицы расстояний для короля и матрицы смежности для рыцарей}
  for c1:='A' to 'H' do
  for  i:='1' to '8' do
  for c2:='A' to 'H' do
  for  j:='1' to '8' do
    begin
      if abs(ord(i)-ord(j))>abs(ord(c1)-ord(c2)) then
        king[c1,i,c2,j]:=abs(ord(i)-ord(j))      else
        king[c1,i,c2,j]:=abs(ord(c1)-ord(c2));
      if abs(ord(i)-ord(j))*abs(ord(c1)-ord(c2))=2 then
          k1[c1,i,c2,j]:=1 else k1[c1,i,c2,j]:=100;
    end;
  {заполнение "диагональных" элементов матрицы смежности  для рыцарей}
    for c1:='A' to 'H' do
  for  i:='1' to '8' do
    k1[c1,i,c1,i]:=0;
  {алгоритм Флойда}
  for c3:='A' to 'H' do
  for  m:='1' to '8' do
  for c1:='A' to 'H' do
  for  i:='1' to '8' do
  for c2:='A' to 'H' do
  for  j:='1' to '8' do
    if k1[c1,i,c3,m]+k1[c3,m,c2,j]<k1[c1,i,c2,j] then
       k1[c1,i,c2,j]:=k1[c1,i,c3,m]+k1[c3,m,c2,j];
end;
procedure Solve;
begin
  {цикл по возможным конечным клеткам (c1,i)}
  for c1:='A' to 'H' do
  for  i:='1' to '8' do
    begin
      r[c1,i]:=king[c1,i,knight[0][1],knight[0][2]];
      for h:=1 to n do  {суммирование путей рыцарей с путем короля}
      inc(r[c1,i],k1[c1,i,knight[h][1],knight[h][2]]);
      res[c1,i]:=r[c1,i]; {начальное значение результата}
      for h:=1 to n do
        begin
          if k1[knight[0][1],knight[0][2], knight[h][1],knight[h][2]]+
                k1[knight[0][1],knight[0][2],c1,i]
                =k1[knight[h][1],knight[h][2],c1,i]
          then {путь рыцаря проходит через местоположение короля}
          begin
           res[c1,i]:=r[c1,i]-king[c1,i,knight[0][1],knight[0][2]];
           break
            {других рыцарей можно не рассматривать}
          end
          else
           {ищем лучшую точку встречи рыцаря и короля (c2,j)}
            for c2:='A' to 'H' do
            for  j:='1' to '8' do
             begin
              rr:=king[c2,j,knight[0][1],knight[0][2]]+
              r[c1,i]+k1[c2,j,knight[h][1],knight[h][2]]+
              k1[c2,j,c1,i]-
              king[c1,i,knight[0][1],knight[0][2]]-
                 k1[c1,i,knight[h][1],knight[h][2]];
                if rr<res[c1,i] then res[c1,i]:=rr
             end
        end
    end;
  {определение наилучшей конечной клетки}
  min:=res['A','1'];
  for c1:='A' to 'H' do
  for i:='1' to '8' do
    if res[c1,i]<min then min:=res[c1,i];
  assign(output,'camelot.out');
  rewrite(output);
  writeln(min)
end;
       begin {основная программа}
          ReadData;
          Distance;
          Solve
       end.
```


Ребусы по информатике

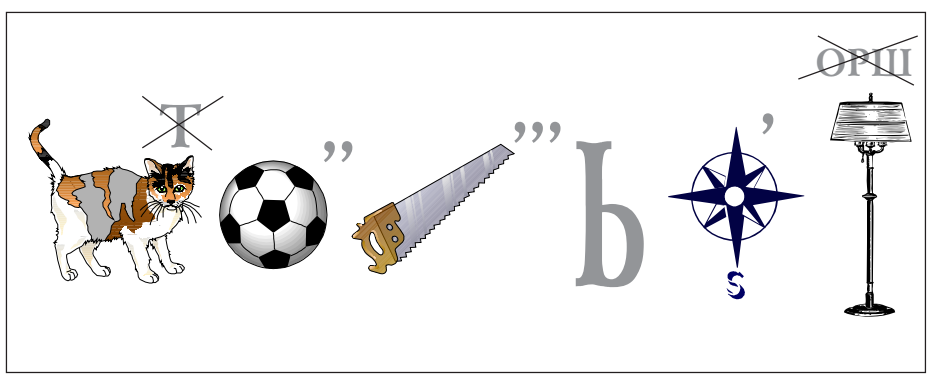


Информатика



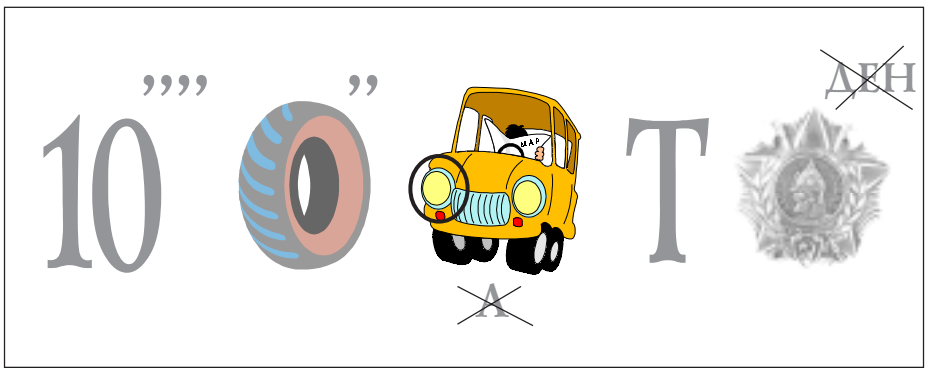
Винчестер

Накопитель



Компьютер

Сумматор



Дешифратор

Ребусы подготовил В.Г. Федоринов

ВНЕКЛАССНАЯ РАБОТА ПО ИНФОРМАТИКЕ

Интернет для начинающих

Продолжение. Начало на с. 1

Папа. Точно так же: поле **Тема** надо оставить пустым, а команды записываются в тексте письма по одной в каждой строке.

В СКИ робота **archie@archie.hensa.ac.uk** есть еще одна очень полезная команда — **Servers**. В ответ на нее робот пришлет список всех archie-серверов в сети, которые работают в настоящее время.

Вася. Зачем нам нужен этот список, если один archie нам уже известен? Будем работать только с ним.

Папа. Всегда полезно иметь под рукой несколько адресов. Если один сервер отключится от сети или будет загружен работой, то можно обратиться к другим. Кроме того, базы данных archie-серверов не всегда совпадают.

Вася. Об этом я не подумал. А как записывать команду для поиска нужного файла?

Папа. Об этом рассказано в инструкции, которая приходит по почте в ответ на команду **Help**. Команда поиска ftp-архива, содержащего файл с именем **имя_файла** для робота **archie.hensa.ac.uk**, выглядит так:

find имя_файла

(*Find* (файнд) переводится с английского на русский словом "найти".)

Вася. Помнишь, на нашем компьютере была игра арканоид? Мне она нравилась, но потом она куда-то пропала. Наверно, брат удалил с диска. Мы можем поискать в сети эту программу?

Папа. Конечно! Какое имя было у этой программы на диске?

Вася. Я точно не помню... Кажется, **arkanoid.exe**. А может быть, **arcanoid.exe**. Как же быть? Надо посылать запросы с разными вариантами имен?

Папа. В этом нет необходимости. В качестве предмета поиска в команде **find** можно указывать только часть имени файла, и archie-робот будет искать все файлы, ее содержащие. Нужный тебе поиск можно задать командами:

set search sub	— искать как составную часть имени
set sortby size	— сортировать результат поиска по размеру
set maxhits 1000	— искать не более 1000 файлов
find anoid	— искать по заданному образцу

Первая команда, **search sub**, устанавливает режим поиска, в котором заданный образец рассматривается как составная часть имени искомого файла. Если этой команды не будет, некоторые archie-роботы будут искать файлы, имя которых точно совпадает с заданным образцом.

Вторая команда, **sortby size**, задает сортировку списка найденных файлов по их размеру. Сортировка полезна, так как нас не интересуют слишком большие файлы: их прием не по силам для нашего модема. А когда файлы упорядочены по размеру, можно сразу рассматривать только нужную часть списка.

Третья команда, **maxhits 1000**, определяет предельное число файлов в найденном списке. Если такую команду не поместить в программу, робот будет выдавать списки, ограничиваясь каким-то своим размером списка, установленным для него по умолчанию. Для нашего робота этот "умалчиваемый" размер равен 100 и может оказаться недостаточным для поиска нужного файла.

Ну а четвертая команда, **anoid**, задает образец поиска.

Вася. Пошлю письмо не откладывая.

Готово!

Папа. Сервер **archie.hensa.ac.uk** очень быстрый. Мы не успеем выпить и по стакану чая, как он пришлет нам ответ.

Действительно, когда через пять минут Вася запустил почтовую программу, выяснилось, что письмо от archie-робота уже пришло.

Вася. Теперь надо разобраться с тем, что мы получили:

```
Host ftp.gns.getronics.nl       (145.13.42.4)
Last updated 14:20 11 Aug 1998

.....

Location: /pub/os/sinclair/games-fullnames/a
FILE -rw-r--r-- 30380  04:11 23 Apr 1998  Arkanoid1.z80.zip

Location: /pub/os/sinclair/games-fullnames/a
FILE -rwxr--xr--x 25332  17:38  3 Jun 1998  Arkanoid2_48Trainer.z80.zip
.....

Host homer.cc.utexas.edu        (128.83.40.2)
Last updated 03:15 29 Oct 1998

Location: /microlib/dos/games
FILE    -rw-rw-r--    34718  20:00 15 Oct 1991  bananoid.zip
```

Папа. В строке **Host** указан сетевой адрес ftp-архива, в котором обнаружены файлы, содержащие в имени шаблон поиска.

Во второй строке, **Last updated**, приводятся дата и время последних обновлений ftp-архива.

Строка **Location:** содержит имя каталога архива, в котором обнаружены файлы, а далее следуют строки с информацией об этих файлах.

Вася. Эти строки очень похожи на те записи, которые выдает ftp-сервер на команду **Dir**.

Папа. Совершенно верно! Например, строка

```
FILE  -rw-rw-r--  34718  20:00 15 Oct 1991  bananoid.zip
```

означает, что в каталоге расположен файл **bananoid.zip**. Этот файл имеет размер 34718 байт, и он был записан в архив вечером, в 8 часов, 15 октября 1991 года. Файл доступен для чтения — об этом говорит буква **r** (третий символ с конца в записи **-rw-rw-r--**).

Вася. Меня удивляет странный вид записи имен файлов в списке. Например, **Arkanoid2_48Trainer.z80.zip**. Основа этого имени **Arkanoid2_48Trainer** содержит не 8 символов, а девятнадцать (!) и, что самое удивительное, имя содержит два расширения — **z80** и **zip**!

Папа. Ты просто привык к обозначению имен в DOS. Большинство серверов в сети работает под операционной системой UNIX, а в ней нет жестких ограничений на форму и длину имени для файла. Кроме того, на нашем компьютере установлена система Windows 95, и она тоже понимает такие имена.

Но файл **Arkanoid2_48Trainer.z80.zip** нас не интересует. Скорее всего этот файл предназначен для процессоров Z80. Таким процессором снабжены, например, компьютеры Ямаха и Синклер. Обрати внимание на название каталога, в котором расположен файл: **/pub/os/sinclair/games-fullnames/a**. Запись **os/sinclair** (ос/синклер) говорит с большой вероятностью о том, что файл предназначен для операционной системы компьютера Синклер. На нашей PC он работать не будет. Мне кажется, надо заказать файл **bananoid.zip**.

Вася. Но это какой-то "банаоид", а нам нужен арканоид!

Папа. У игры арканоид очень много модификаций, или, как говорят, клонов. Давай еще раз посмотрим внимательно на запись:

```
Host homer.cc.utexas.edu        (128.83.40.2)
Location: /microlib/dos/games
FILE   -rw-rw-r—     34718  20:00 15 Oct 1991  bananoid.zip
```

Из нее можно извлечь такую информацию:

— Файл расположен в ftp-архиве в Америке (скорее всего в штате Техас).

— Файл запакован архиватором Zip. Такой архиватор у нас есть, и, значит, мы сможем файл распаковать.

— Файл небольшой по размеру — всего около 35 Кб. Значит, у нас с нашим модемом не будет сложностей с его приемом.

— В названии каталога есть слово **dos**, а это, вероятно, говорит о том, что файл — программа для операционной системы DOS нашего компьютера PC.

— В названии каталога есть слово **games**, значит, в этом каталоге располагаются игровые программы.

— Наконец, запись **-rw-rw-r--** говорит о том, что файл BANANOID.ZIP в этом каталоге доступен для чтения, то есть мы сможем получить его по сети.

Вася. Твой рассказ напомнил мне диалоги Холмса с Ватсоном. Из такой небольшой записи, оказывается, можно извлечь так много полезных сведений. Хорошо, давай закажем этот банаоид нашему ftp-роботу! Вот мое письмо-программа:

```
open homer.cc.utexas.edu
cd /microlib/dos/games
get bananoid.zip
quit
```

Папа. Последняя очень важная рекомендация. Когда файл окажется на нашем компьютере и ты его распакуешь, не забудь проверить его на вирусы!

Вася. Это я хорошо усвоил. В этом деле главное — сначала запустить антивирусную программу, а потом игрушку, и никак не наоборот!

Для учителя
(комментарии для серьезных читателей)

В 1990 году Питер Дейч, Билл Хилан и Элан Эмтидж (студенты университета Мак-Гилла в Монреале (*McGill University*) создали базу данных, которую назвали Archie, и программу, которая периодически обращается к ftp-серверам и обновляет базу данных. Теперь каждый может обратиться к archie-серверу и спросить, где в сети можно взять нужный файл.

Следует отметить, однако, что поиск файла при помощи archie-сервиса эффективен только тогда, когда известно точное имя файла или хотя бы точная часть имени. Часто такой информации нет. Например, нужно найти тексты песен Beatles. Как могут называться эти файлы? Вообще говоря, как угодно, хотя, конечно, разумно начать поиск по шаблону "beatle". В полученном списке оказывается множество строк совершенно ненужных и ни одной, про которую можно стопроцентно сказать: да, в ней указана та самая информация! Вот примеры результатов такого поиска:

```
Host ftp.unina.it       (192.132.34.17)
Location: /pub/Other/music/midifile2/e
FILE -rw-r--r-- 46260 23:00 8 Jun 1997 Beatles-Beautiful_Boy.mid.gz
Location: /pub/Other/music/midifile2/e
FILE -rw-r--r-- 12357 23:00 4 Jan 1995 Beatles-909.mid.gz
```

Окончание на с. 16

Интернет для начинающих

Окончание. См. с. 1, 15

```
Host ftp.jussieu.fr (132.227.77.2)
Location: /pub/idgames/levels/doom2/deathmatch/a-c
FILE -r--r--r-- 35702 00:00 28 Jun 1995 beatle.txt
```

Последняя строка — ссылка на текстовый файл. Может быть, в нем содержится то, что нужно? Правда, сильно смущает французское происхождение ftp-сервера и запись "doom2"в названии каталога. Посылка запроса на получение файла **beatle.txt** привела к сообщению "файла нет в каталоге"! Почему? Вероятно, у archie-сервера устаревшая информация об этом файле.

Можно, отчаявшись, послать запрос по шаблону "song". Но и это не слишком хорошо. Из строчек типа:

```
Location: /pub/mirror/win95/winsite/games
FILE   -rw-r--r--      1804961  05:38  9 Feb 1998  lsongs.zip
Location: /ftp/disk5/pc-demos/music/songs/1996/o
FILE   -r--r--r--       89864  22:00 23 Dec 1996  oz96song.zip
```

ничего нельзя предположить о содержимом архивив.

Опыты с archie показывают, что, используя этот сервис, мало шансов найти что-нибудь полезное в сети по ключевым словам (а не по известным именам файлов). Прекрасной альтернативой archie-поиску является поиск в WWW-пространстве при помощи специальных поисковых on-line систем типа Alta Vista, Yahoo, HotpBot, Рамблер, Апорт и других.

Вопросы и упражнения

1. Что такое archie-сервис Интернета?

Ответ. Archie-сервис — это механизм поиска файлов в ftp-архивах сети по их именам.

2. Чем отличается archie-сервер от почтового робота archie?

Ответ. Archie-серверы хранят списки файлов в ftp-архивах сети и постоянно эти списки обновляют. Почтовый archie-робот используется для работы с archie-сервером по электронной почте.

3. Каким почтовым роботом нужно воспользоваться для работы с archie-сервером **archie.cs.mcgill.ca**? Где находится этот почтовый робот и какой у него сетевой адрес?

Ответ. Почтовый archie-робот располагается на соответствующем archie-сервере, то есть в данном случае в Канаде (об этом говорит обозначение домена "ca"). Адрес почтового робота строится из адреса сервера добавлением приставки "archie@". Адрес почтового робота в данном случае: a**rchie@archie.cs.mcgill.ca**.

4. Как получить по электронной почте список всех archie-серверов сети, работающих в настоящее время?

Ответ. Надо послать почтовому archie-роботу письмо с пустой темой, содержащее одну команду **servers**.

5. Как познакомиться с системой команд почтового archie-робота и инструкцией по составлению программ для него?

Ответ. Нужно послать роботу письмо с командой **help**.

6. При помощи какой команды можно передать archie-роботу задание на поиск файла в сети? Как управлять поиском?

Ответ. Как правило, командой, задающий поиск, является команда **find шаблон**.

Если не заданы дополнительные команды, уточняющие режимы поиска, то роботы используют заранее определенный для них режим "умолчания".

Одни роботы в этом случае используют шаблон для точного поиска, другие — для нахождения файлов, в имена которых шаблон входит как составная часть.

Лучше всего не полагаться на режим "умолчания" робота, а задавать явное указание, "как искать", командой **set search**.

Возможные варианты этой команды:

set search exact — точный поиск (поиск файлов, имена которых совпадают с шаблоном);

set search sub — расширенный поиск (поиск файлов, в именах которых шаблон содержится как составная часть).

7. Как управлять видом результата поиска?

Ответ. Можно использовать команды **set maxhits** и **set sortby**. Первая команда ограничивает длину списка-результата, вторая позволяет упорядочивать список по разным параметрам.

Возможные варианты команды **set sortby**:

set sortby none — не упорядочивать результат;
set sortby filename — упорядочивать по именам файлов;
set sortby hostname — упорядочивать по именам ftp-архивов;
set sortby size — упорядочивать по размерам файлов;
set sortby time — упорядочивать по времени создания файлов.

8. Играет ли роль регистр буквы в названиях:

а) ftp-архивов;
б) каталогов;
в) файлов?

Ответ:

а) нет;
б) да;
в) да.

9. Какую информацию можно извлечь из следующих строк результатов поиска:

а)

```
Host comserv.urz.uni-magdeburg.de       (141.44.251.1)
Last updated 23:31 11 Jun 1997

Location: /pub/mirror/freebsd/FreeBSD-current/src/
usr.sbin/sup/lib
 FILE ----------   53224 00:00 1 Jan 1970 bitlcomp.c?
```

Ответ.

— Файл **bitlcomp.c** скорее всего является исходным текстом программы на языке программирования Си.
— Этот файл расположен на сервере в Германии.
— Размер файла — 53224 байта.
— Файл создан в каталоге в момент начала 1970 года.
— Файл недоступен для чтения и передачи по сети.

б)

```
Host nic.switch.ch      (130.59.10.32)
Last updated 13:45  2 Nov 1997

Location: /mirror/psion/EBooks
DIRECTORY  drwxrwxr-x  512  03:25  5 Aug 1996  Carroll
```

Ответ.

— Объект **Carroll** является каталогом.
— Каталог создан в 3.25 утра 5 августа 1996 года.
— Каталог расположен на сервере в Швейцарии.
— Наиболее вероятное содержимое каталога — тексты книг Льюиса Кэрролла.

в)

```
Host ftp.unina.it       (192.132.34.17)
Last updated 13:01 26 Mar 1998

Location: /pub/Amiga/aminet/biz/demo
  FILE    -rw-rw-r--    447   23:00  23  May  1995
kidpixdemo.lha
```

— Архив **kidpixdemo.lha**, вероятно, содержит демонстрационные материалы по известному детскому графическому редактору Kidpix.
— Файл расположен на сервере в Италии и предназначен для компьютера Amiga.

10. Что надо не забыть сделать перед запуском программы, полученной из сети?

Ответ. Программу необходимо проверить антивирусной программой самой последней версии.

11. Повторите опыт Васи. Найдите в сети компьютерную игру и перепишите ее на свой компьютер.

ТЕЛЕ-КОММУНИКАЦИИ

ОБЪЕДИНЕНИЕ ПЕДАГОГИЧЕСКИХ ИЗДАНИЙ "ПЕРВОЕ СЕНТЯБРЯ"

Первое сентября
А.С. Соловейчик
индекс подписки — 32024

Английский язык
Е.В. Громушкина
индекс подписки — 32025

Биология
Н.Г. Иванова
индекс подписки — 32026

Воскресная школа
монах Киприан (Ященко)
индекс подписки — 32742

География
О.Н. Коротова
индекс подписки — 32027

Здоровье детей
А.У. Лекманов
индекс подписки — 32033

Информатика
Е.Б. Докшицкая
индекс подписки — 32291

Искусство
Н.Х. Исмаилова
индекс подписки — 32584

История
А.Ю. Головатенко
индекс подписки — 32028

Литература
Г.Г. Красухин
индекс подписки — 32029

Математика
И.Л. Соловейчик
индекс подписки — 32030

Начальная школа
М.В. Соловейчик
индекс подписки — 32031

Немецкий язык
Gerolf Demmel
индекс подписки — 32292

Русский язык
Л.А. Гончар
индекс подписки — 32383

Спорт в школе
Н.В. Школьникова
индекс подписки — 32384

Управление школой
Н.А. Широкова
индекс подписки — 32652

Физика
Н.Д. Козлова
индекс подписки — 32032

Химия
О.Г. Блохина
индекс подписки — 32034

Школьный психолог
М.Н. Сартан
индекс подписки — 32898

Гл. редактор
Е.Б.Докшицкая
Зам. гл. редактора
С.Л.Островский
Редакция:
Л.Н.Картвелишвили,
Ю.А.Соколинский,
Н.Л.Беленькая,
Н.П.Медведева
Дизайн и компьютерная верстка:
Н.И.Пронская
Корректоры:
Е.Л.Володина,
С.М.Подберезина

©ИНФОРМАТИКА 1999
выходит четыре раза в месяц
При перепечатке ссылка на ИНФОРМАТИКУ обязательна, рукописи не возвращаются.
Регистрационный номер 012868

121165, Москва, Киевская, 24
тел. 249 4896
Отдел рекламы
тел. 249 9987

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков 32291
для предприятий и организаций 32591
комплекта приложений 32744

Internet: inf@1september.ru
Fidonet: 2:5020/69.32
WWW: http://www.1september.ru

Отпечатано с готовых диапозитивов редакции в типографии "ПРЕССА", 125865, Москва, ул. Правды, 24

Тираж 7000 экз.
Заказ №

Start тел./факс (095)249 3138, факс (095)249 3184, тел. 249 3386

10.03