№ 1 (194) январь 1999

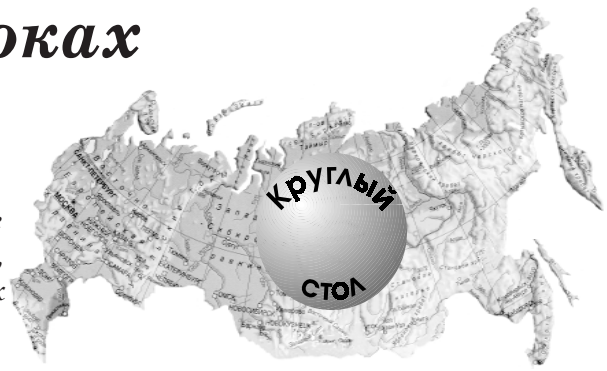
еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИК

Электронная версия газеты Информатик http://www.glasnet.ru/~infosef

Чему учить на уроках информатики

А.Ю. УВАРОВ
Научный совет по комплексной проблеме "Кибернетика", Российская академия наук



Недавно на одном из заседаний комиссии по информатике Федерального экспертного совета председательствующий, завершая обсуждение очередного учебника, с грустью заметил: можно предъявлять разные претензии к учебнику информатики под редакцией академика А.П. Ершова, однако, в отличие от многих появившихся ныне учебников, уровень научной достоверности материала в книге, написанной более десяти лет назад, несравненно выше.

С этим трудно не согласиться. Авторы первого учебника по информатике — ведущие отечественные специалисты. За их плечами огромный практический опыт:

— научные разработки в исследовательских институтах и вычислительных центрах Академии,
— обучение школьников программированию,
— проведение детских компьютерных лагерей.

Они хорошо знали, что, овладевая приемами создания законченных компьютерных программ, дети овладевают новыми мыслительными операциями, новым взглядом на окружающий их мир. У них формируются навыки планирования работы исполнителей, привычка к точному и полному описанию этих действий, представление о способах анализа систем и навыки такого анализа. Все это было условно названо "процедурным", или "алгоритмическим", мышлением. Для сторонников введения информатики как отдельного предмета было важно, что традиционный курс математики, который (и это — одно из бесспорных достижений советской школы) ответствен за формирование у учащихся приемов "абстрактного мышления", никогда не ориентировался (да и теперь не ориентируется) на формирование "алгоритмического мышления". И это естественно: "математики" и "вычислители" — носители различных "взглядов на мир". В то же время есть все основания полагать, что формирование алгоритмического мышления способствует повышению уровня современной математической подготовки школьников, а "математическая культура" — необходимое условие для эффективного обучения информатике.

В качестве основного "управленческого" аргумента для введения в школу информатики авторы первого учебника ссылались на практический (народно-хозяйственный) аспект: обучение программированию актуально, как и обучение всякому вновь возникающему мастерству. В середине 80-х планировалось широкое распространение ЭВМ, и "мастера по компьютерам" обещали стать не менее нужными и распространенными, чем "мастера по автомобилям" или электрики. Для лиц, принимающих решение, эта ситуация напоминала ту, что имела место в стране несколько десятилетий назад, когда промышленность (и "весь советский народ") осваивала электротехнику и электромеханику. Введенный во всех школах страны общеобразовательный курс электротехники, бесспорно, помог делу. Соответствующий компонент технической культуры стал "достоянием масс".

И, наконец, "просвещенческий" аргумент: общество во всех его слоях должно было узнать о начавшейся "информационной революции", без страха и излишних иллюзий отнестись к "новому чуду современной техники" — компьютеру. А для этого нет более эффективного инструмента, чем общеобразовательная школа.

Романтиков, которые стремились ввести компьютер в школу, подогревала уверенность в том, что вслед за введением информатики и появлением компьютеров в школе

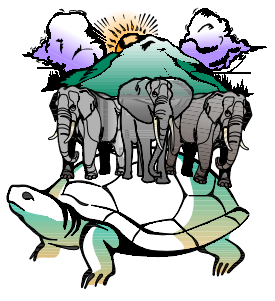
Продолжение на с. 2

НАШИ ДЕТИ БУДУТ ЖИТЬ В XXI ВЕКЕ

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

Лекция 1

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 и 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.

ЗДРАВСТВУЙТЕ, ДОРОГИЕ НАШИ ЧИТАТЕЛИ!

С Новым годом!

Мы благодарны нашим старым подписчикам, которые даже в такое непростое время остались с нами, и очень рады новым подписчикам, ожидания которых постараемся не обмануть.

Нам всегда было важно ваше мнение – результаты обработки анкет, присланных вами в конце прошлого года, мы уже учитываем при планировании номеров. Но в анкетах были конкретные вопросы, а сейчас мы просто хотим попросить вас высказать свои пожелания. Любые – по подбору материалов, по оформлению газеты, по составу спецвыпусков и т.д. Ведь "мнение читателей" – это не абстрактное понятие, это точка зрения каждого из вас. Нам это важно.

_____

_____

_____

_____

Вырежьте этот купон и пришлите его нам. Обязательно укажите свой полный почтовый адрес, фамилию, имя и отчество. Все, отправившие заполненные купоны до 31 января, станут участниками розыгрыша книг по информатике и антивирусных программ, предоставленных ведущим российским разработчиком программного обеспечения – фирмой "ДиалогНаука".

— С 5 ПО 8 ЯНВАРЯ 1999 г. В МОСКВЕ —

VIII Открытая Российская научно-практическая конференция школьников по физике, математике, информатике, экологии

Министерство образования Российской Федерации
Комитет образования г. Москвы
Московский энергетический институт (технический университет)
Московский государственный университет им. М.В. Ломоносова
Журнал "Квант"
Энергофизический лицей № 1502 при МЭИ

Приглашаем вас принять участие в работе VIII Открытой Российской научно-практической конференции школьников по физике, математике, информатике, экологии, которая состоится с 5 по 8 января 1999 г. в Энергофизическом лицее № 1502 при МЭИ.

Открытие конференции — 5 января в 10^{00} в актовом зале лицея. Предметные секции будут работать 5 января с 10^{00} до 17^{00} (с перерывом на обед) и 6 января с 9^{30} до 14^{00}.

Закрытие конференции — 8 января в 12^{00}.

Адрес лицея: г. Москва, ул. Молостовых, д.10а
Тел. 307-11-61
Факс 300-15-31
Проезд: ст. метро "Новогиреево" (от центра первый вагон), автобусы № 237, 247, 659; троллейбус № 77 до остановки "Напольный проезд".

Чему учить на уроках информатики

Продолжение. Начало на с. 1

начнутся изменения и *в целях*, и *в содержании*, и *в методах* обучения. Еще в конце семидесятых годов С.Пейперт (см.: "Переворот в сознании: дети, компьютер и плодотворные идеи") продемонстрировал, что компьютер — эффективный инструмент обучения, открывающий двери для новой педагогической практики (конструктивизм). Становилось ясно, что "традиционные" составляющие реформы школы, направленные на экстенсивный рост системы образования (увеличение продолжительности школьного обучения, введение новых учебных предметов, снижение наполняемости классов и т.п.), с неизбежностью придется дополнять факторами, направленными на "интенсификацию учебного процесса", качественными преобразованиями, способствующими повышению эффективности образования. Предложение использовать компьютеры для повышения эффективности педагогического труда ("техническое перевооружение отрасли") воспринималось так же естественно, как предложение механизировать и автоматизировать производственные процессы в машиностроении. В этой обертке, под флагом "компьютерной грамотности и введения компьютеров в учебный процесс", говоря об "электронных учебниках" будущего, авторы программы "компьютеризации школы" пытались стимулировать процесс внесения принципиальных изменений в содержание, методы и организационные формы обучения: обучение техникам умственной работы ("процедурное мышление"), распространение практики проведения "учебных проектов" (развитие самостоятельности школьников и практическая направленность обучения), изменение взаимоотношений между учителем и учеником (опыт компьютерных лагерей начала 80-х), новый уровень доступа учителей и учащихся к информации (на машинных носителях информации) и т.п.

К сожалению, в середине 80-х и политическая ситуация в стране, и полное отсутствие компьютеров в школе, и крайняя нехватка специалистов, понимающих происходящие процессы, не позволяли не только развернуть работы в этом направлении, но даже их анонсировать. Внедрение ЭВМ в учебный процесс долго оставалось пустыми словами, частью названия соответствующего постановления ЦК КПСС. Понадобилось несколько лет лишь для того, чтобы вслух заявить о неизбежности процесса "информатизации образования"[1]. Однако на практике дело ограничивалось (да и сегодня часто ограничивается) обучением школьников по курсу информатики. Темы, связанные с изменением содержания, методов и организационных форм обучения в связи с построением школы, отвечающей нуждам "информационного общества", практически так и не рассматриваются.

Итак, цель введения общеобразовательного курса "Информатика" включала одновременно несколько составляющих:

1. "общекультурная" составляющая — познакомить школьников с компьютерами, новой распространенной частью "культурного ландшафта", среды обитания современного человека, дать представление о "процессах информатизации" в современном обществе;
2. "технологическая" составляющая — научить каждого пользоваться новыми массовыми "информационными технологиями" (клавиатура, текстовый редактор, электронные таблицы и т.п.);
3. "предпрофессиональная" составляющая — подготовка будущих "работников информационной сферы" — обучение программированию, устройству компьютеров, электронике;
4. "общеобразовательная" составляющая — обучение "процедурному мышлению";
5. "общепедагогическая" составляющая — кабинет информатики в школе, учитель информатики, новая "педагогическая культура", обновление содержания, а главное — методов и организационных форм учебной работы (закрепившихся в информатике) во всех учебных предметах.

Значительный "синергетический" запас, заложенный в основу начинающегося процесса, должен был обеспечить "выживание" курса информатики, помочь ему устоять против пассивного (и активного) сопротивления, которое оказывала (и оказывает) ему каждодневная практика работы массовой школы. Прошедшее десятилетие показало, что, несмотря на пророчества критиков, которых было так много в 80-х, курс информатики устоял. Сегодня он — факт жизни школы.

* * *

Помню, как в 1987 году, когда введение информатики в школу шло полным ходом, в Москву проездом из Бостона в Токио заглянул Сеймур Пейперт. Это было настоящим культурным событием. Пейперт рассказывал о новых педагогических экспериментах, интересовался "компьютеризацией" школы в СССР. Узнав, что основным направлением на данный период является введение общеобразовательного курса информатики во всех школах страны, он ненадолго задумался. А вас не пугает, спросил он, что, успешно вводя информатику в школу как новый учебный предмет, вы создаете барьер на пути действительных широкомасштабных преобразований учебного процесса? Ведь новый учебный предмет — это десятки тысяч учителей, которые ему учат, педагоги и кафедры в педагогических вузах, которые этих учителей готовят, методисты и методические кабинеты в институтах усовершенствования, ученые советы, научно-исследовательские институты, методические журналы, уважаемые и влиятельные директора, профессора, заведующие — все то, что называют "макросистемой учебного предмета". Это живой организм, влиятельная общественная сила. Возникнув, она будет стремиться к самосохранению, стараться поддержать статус-кво. Десятки тысяч достойных людей, специалистов своего дела, будут ассоциировать себя с новым учебным предметом. Сложится инфраструктура, аналогичная инфраструктурам других учебных предметов. Эта инфраструктура будет жить своей собственной жизнью, ратовать за увеличение отводимых для ее деятельности учебных часов, распространять информатику на другие возрастные группы учащихся (начальную и неполную среднюю школу), устанавливать "линии раздела" с математикой и другими смежными учебными предметами (как это уже давно делают все предметники). Информатика станет одним из традиционных учебных предметов, перестанет быть катализатором действительных перемен.

Это был "хороший вопрос", на который трудно ответить. Он звучал из уст специалиста и убежденного сторонника информатизации школы. Получалось, что чем глубже мы введем курс информатики в школу, тем дальше окажемся от главной цели — качественного совершенствования ее работы. Механизмы "самозащиты социальных образований" хорошо известны, и примеров их множество (среди наиболее известных в СССР — система водоканала). Получалось, что развитие учебного предмета "информатика" вело к созданию системы, объективно работающей против цели, ради которых она была введена, — подготовки школы к вступлению в информационное общество. Конечно, не все столь мрачно смотрели на "истинную природу человека и человеческих сообществ". Ратуя за введение в школу общеобразовательного курса информатики, академик А.П. Ершов надеялся, что возникнет корпус учителей-новаторов, преподавателей учебного предмета нового типа, что учителя информатики станут в авангарде сил, изменяющих само содержание и методы учебной работы, начнут практическую перестройку школы снизу. Ему виделось, как учитель информатики — универсал, в совершенстве владеющий всеми новыми информационными технологиями и современным системным взглядом на мир, способный к разработке и введению в практику человеко-машинных организационных систем — распахнет двери кабинета информатики для всех педагогов школы, поможет им освоить новые эффективные методы работы с информацией, будет стимулировать внесение изменений в содержание и методы учебной работы независимо от (а подчас и вопреки) неуклюжих действий органов управления образованием. Он полагал, что верх возьмет не работа "естественных механизмов самосохранения большой системы", а разум, воля и энтузиазм составляющих ее людей. Что верх возьмут педагоги, осознающие направление грядущих изменений и сознательно работающие на совершенствование деятельности школы (сознательно преследующие долгосрочную, а не ситуационную выгоду).

* * *

Сегодня никого не надо убеждать, что компьютер — полезный массовый инструмент обработки информации, что завтрашний день невозможно представить себе без компьютеров. При всех известных трудностях школы сейчас оснащены вычислительной техникой несравненно лучше, чем десять лет назад. Информатизация школы пошла вглубь, а курс информатики уже потерял монополию на использование компьютера в обучении. Появились успешные образцы использования вычислительной техники при обучении другим предметам. Созданы первые интегрированные курсы, где информатика естественно вплетается в изучение традиционных дисциплин[2]. В Московском департаменте образования даже обсуждается идея разделить обучение информатике между курсами "Математика" и "Технология". Преподаватель информатики постепенно теряет статус единственного полномочного представителя грядущего информационного общества в школе. С другой стороны, развивается и курс информатики: сегодня его преподают не только в старших классах. Во многих регионах страны она с успехом изучается и в начальной, и в неполной средней школе. Готовятся кадры учителей, множится число выпущенных учебников. В отведенные программой часы уже не вместить и знакомство с компьютером, и овладение алгоритмическим мышлением, и освоение информационных технологий, и изучение социальных последствий массового распространения ЭВМ. Учителю информатики в дополнение к своим "основным обязанностям" — проведению уроков по информатике — приходится заниматься и развитием внутришкольной информационной среды (локальная сеть школы, Интернет), и поддержкой других учителей, пытающихся использовать компьютер в своих предметах, и вести оригинальные методические разработки (например, учебные телекоммуникационные проекты). Естественно, возникает желание включить все это в свой собственный предмет. Сложилось сообщество преподавателей информатики, которое ищет новое место для своего учебного предмета в системе учебных дисциплин. Школьная информатика вступила в фазу зрелости. Теперь она чувствует себя так же устойчиво, как и другие учебные предметы. Серьезно и со знанием дела обсуждается главный вопрос: чем действительно должен заниматься преподаватель информатики со своими учениками на уроках сегодня? Где начинаются и где кончаются его обязанности как "преподавателя своего особого учебного предмета", сколько классов и сколько недельных часов должен занимать и какое содержание должен охватывать этот предмет?

С одной стороны, это не слишком сложные вопросы. Ответы на них были даны еще на этапе введения информатики как отдельного предмета и определяются пятью названными выше составляющими. Сегодня преподаватели информатики разных школ в зависимости от своей подготовки, интересов и имеющихся условий фактически сами выбирают, чем они будут заниматься на уроках: первичным знакомством с компьютером, программированием, изучением моделей процессов и явлений, освоением новых информационных технологий, развитием специфических навыков умственной работы? (Последнее направление сейчас самое "модное". Однако серьезных данных за то, что это удается делать кому-то еще, кроме тех, кто учит детей, имеющих серьезную математическую подготовку, теоретическому или практическому программированию, пока нет.) Сегодня каждая школа получила известную автономию в определении, чему и как учить своих воспитанников. Мудрые учителя информатики делают свой выбор с учетом своих человеческих и профессиональных возможностей, интересов своих учеников, пожеланий коллег, технического оснащения и традиций школы.

С другой стороны, эти вопросы неразрешимы. По своей сути они ничем не отличаются от аналогичных вопросов, которые задают себе представители каждого из современных учебных предметов. Как "разъять живую ткань действительности" по учебным дисциплинам — ведь в природе физические процессы не отделены от химических, биологических или информационных. Кто возьмется решить, что тему алгоритмизации должны обязательно изучать на уроках математики или что "информатика должна вобрать в себя курс машинописи"? Похоже, что время, когда жизнь школы беспрекословно определял устойчивый и неизменный учебный предмет, постепенно уходит в прошлое. Пришла пора серьезного обсуждения проблем содержания обучения, его интеграции, изменения методов и организационных форм обучения. Информатизация школы продолжается.

* * *

Возникает вопрос: какова сегодня действительная роль сообщества учителей информатики, методистов, преподавателей вузов и исследователей, составляющих наиболее влиятельную часть этого сообщества? Каково возможное будущее информатики как учебного предмета? В каком направлении предпочтительнее двигаться? Обсуждая положение информатики среди других общеобразовательных учебных предметов и ее взаимодействие с этими областями, можно выделить три базовых сценария — взаимо-

Окончание на с. 16

[1] Первая концепция информатизации образования была опубликована в конце 1988 года.

[2] Известный пример — интегрированный курс естествознания и информационной технологии (взамен природоведения и информатики) для учащихся пятого класса "Наблюдай и исследуй".

Новая старая задача

Д.М. ЗЛАТОПОЛЬСКИЙ

"Рассказывают, что в Ханое стоит храм и рядом с ним — три башни (стержни). На первый стержень надеты 64 диска различного диаметра: самый большой диск — внизу, а самый маленький — вверху, больший диск не лежит на меньшем. Монахи этого храма должны были переместить все диски с первого стержня на третий, соблюдая следующие правила:

1) можно перемещать лишь по одному диску;

2) больший диск нельзя класть на меньший;

3) снятый диск нельзя отложить — его необходимо сразу же надеть на другой стержень".

Задачу и легенду придумал математик Э.Люка в 1883 г.

Когда дисков много (64), процесс оказывается необычайно долгим. (Легенда утверждает: когда монахи закончат свою работу — а они перемещают диск за 1 секунду! — наступит конец света.) Поэтому обычно эту задачу решают с меньшим количеством дисков.

Задача "Ханойские башни" достаточно широко известна. Напомним алгоритм ее решения, использующий рекурсию [1, 2].

Пусть число дисков на первом стержне — n. Будем рассуждать так. Предположим, что мы умеем переносить $n-1$ дисков. В этом случае n дисков перенесем посредством следующих шагов:

1) верхние $n-1$ дисков перенесем с первого стержня на второй, пользуясь свободным третьим стержнем;

2) последний диск первого стержня наденем на третий стержень;

3) $n-1$ дисков перенесем со второго стержня на третий стержень, пользуясь свободным первым стержнем.

Аналогичным образом можно перенести $n-1$, $n-2$ и т.д. дисков. Когда $n=1$, осуществить перенос очень просто: непосредственно на другой.

Соответствующая программа на ***школьном алгоритмическом языке*** [1, 3] имеет вид:

```
алг Задача Ханойские башни
нач цел n, сим c1, c2, c3
| c1, c2, c3 - обозначения стержней ("A", "B", "C")
| c1:="A"
| c2:="B"
| c3:="C"
| вывод нс, "Введите число дисков"; ввод n
| MoveDisks(n, c1, c2, c3)
кон
```

— где `MoveDisks` — рекурсивная процедура, которая переносит `n` дисков со стержня `c1` на стержень `c3`, используя вспомогательный стержень `c2`:

```
алг MoveDisks(арг цел n, сим c1, c2, c3)
нач
| если n>0
| | то
| | MoveDisks(n-1, c1, c3, c2)
| | |Рекурсивный вызов
| | PutOneDisk(c1, c3)   |Вспомогательная процедура
| | MoveDisks(n-1, c2, c1, c3)
| | |Рекурсивный вызов
| все
кон
```

А процедура `PutOneDisk` иллюстрирует перенос одного диска со стержня `from` на стержень `to`:

```
алг PutOneDisk(арг сим from, to)
нач
| вывод from, "->", to, ";"
кон
```

Заметим, что имеются и нерекурсивные алгоритмы решения данной задачи (см., например, [4]).

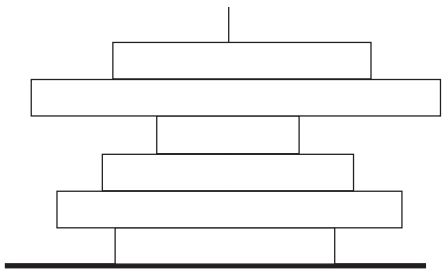


Рис. 1

Эта задача имеет несколько разновидностей. Одна из них состоит в том, что диски на первом стержне находятся не в порядке убывания (при просмотре их снизу вверх), а в некотором произвольном порядке (*рис.* 1). Необходимо собрать все диски на третьем, соблюдая правила перекладывания основной задачи.

Если обозначить первый, второй и третий стержни соответственно **A**, **B** и **C**, то такую задачу можно решить следующим образом:

1. Размещаем по одному диску со стержня **A** на стержне **B** или **C**, соблюдая правила перекладывания.
2. Если не все диски размещены на стержне **C**, то объединяем диски со стержней **B** и **C** на одном из них.
3. Если все диски собраны на стержне **B**, то перекладываем все диски со стержня **B** на стержень **C**.

При выполнении первого этапа возможны 3 ситуации:

1.1. Диск со стержня **A** можно надеть на стержень **C** (*рис.* 2).

1.2. Диск со стержня **A** нельзя надеть на стержень **C**, но можно надеть на стержень **B** (*рис.* 3).

1.3. Остальные случаи (*рис.* 4).

В последнем случае необходимо подготовить место для размещения на стержне **B** или **C** путем перекладывания (возможно, неоднократного) дисков со стержня **B** на стержень **C** и наоборот (естественно, используя стержень **A** как вспомогательный), а затем надеть размещаемый диск со стержня **A** на соответствующий стержень.

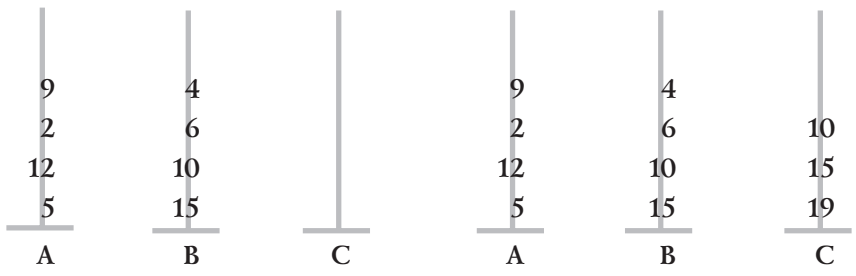


Рис. 2

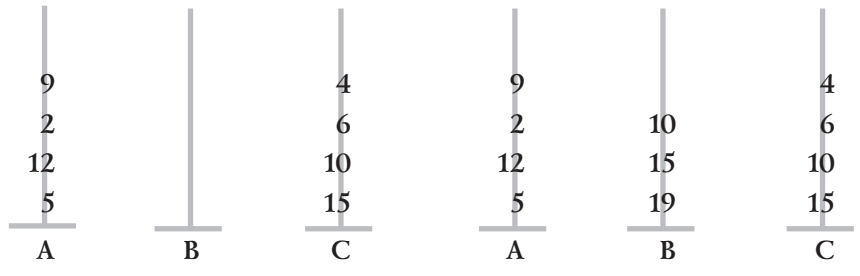


Рис. 3

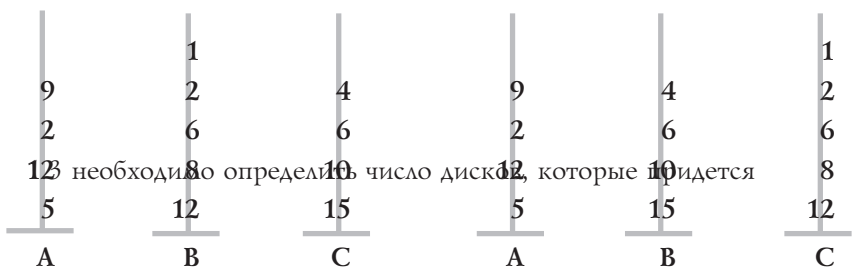


Рис. 4

На рисунках 2, 3, 4 числа соответствуют диаметру дисков.

1.1 и 1.2 можно выполнить, используя описанную выше процедуру `PutOneDisk`.

При выполнении 3 необходимо определить число дисков, которые придется перекладывать со стержня **B** на стержень **C**, и переложить их. Затем выполнить то же, но со стержня **C** на стержень **B**, затем вновь со стержня **B** и т.д. до тех пор, когда диск со стержня **A** можно будет разместить на одном из других стержней (см. *рис.* 4). Например: для ситуации, представленной на *рис.* 4 (слева), следует переложить два диска со стержня **B** на стержень **C**, затем 4 диска — с **C** на **B**, после чего диск со стержня **A** можно разместить на стержне **C**.

Приведем процедуру `Place` (на школьном алгоритмическом языке), которая выполняет описанные действия.

В ней использованы следующие основные величины:

`na`, `nb` и `nc` — число дисков на стержнях **A**, **B** и **C**;

`count` — количество дисков, перекладываемых со стержня на стержень в ходе подготовки места для размещения диска со стержня **A** (см. выше);

`poin` — номер диска, используемый при определении значения величины `count`;

`placed` — величина логического типа, фиксирующая факт подготовки места для размещения диска со стержня **A**.

Стержни с дисками будем моделировать в виде массивов с именами соответственно `a`, `b` `c`, причем диаметр самого нижнего диска будем хранить в первом элементе массива. Размеры дисков на стержне **A** будем задавать случайным образом. С целью упрощения программы примем, что диаметры некоторых дисков могут быть одинаковыми. С учетом этого при перекладывании допустим, что диски одного размера можно класть один на другой.

Заметим также, что после перекладывания дисков со стержня на стержень (по процедуре `MoveDisks`) необходимо изменить размер и состав этих массивов. Процедуру, с помощью которой происходит изменение значений элементов массивов `from` и `to`, а также их размеров `k1` и `k2` при перекладывании последних дисков первого массива в количестве `count` в конец второго массива, назовем `AlterArrays`. Она может быть оформлена так:

```
алг AlterArrays(арг цел count, арг рез цел k1,
    арг рез цел таб from [1:k1], арг рез цел k2,
    арг рез цел таб to[1:k2])
нач цел i, k2old
  k2old:= k2
| k2:=k2old+count
| нц для i от 1 до count
| |  to[k2old+i]:=from[k1-count+i]
| кц
| k1:=k1 - count
кон
```

Продолжение на с. 4

Новая старая задача

Продолжение. Начало на с. 3

Сама процедура Place имеет вид:

```
алг Place (арг рез цел na, арг рез цел таб a[1:na], арг рез цел nb,
           арг рез цел таб b[1:nb], арг рез цел nc,
           арг рез цел таб c[1:nc])
нач цел count, poin, лог placed
| выбор
| | при nc=0 или a[na]<=c[nc]:
| | |диск со стержня A можно надеть на стержень C
| | PutOneDisk("A", "C")
| | |делаем это,одновременно уточняя массивы a и c
| | AlterArrays(1, na, a, nc, c)
| | при nb=0 или a[na]<=b[nb]:
| | |диск со стержня A можно надеть на стержень B
| | PutOneDisk("A", "B")
| | AlterArrays(1, na, a, nb, b)
| | иначе
| | |Готовим место для него на стержне B или C
| | |путем перекладывания дисков на этих стержнях
| | placed:=нет
| | нц
| | | |Перекладываем диски со стержня B на стержень C
| | | |a) определяем количество перекладываемых дисков count
| | | count:=0
| | | poin:=nb
| | | нц пока poin>0 и b[poin]<=c[nc]
| | | | count:=count+1
| | | | poin:=poin-1
| | | кц
| | | если count>0
| | | | то
| | | | |б) и перекладываем их,
| | | | MoveDisks(count, "B", "A", "C")
| | | | |одновременно уточняя массивы b и c
| | | | AlterArrays(count, nb, b, nc, c)
| | | все
| | | если poin=0 или b[poin]>=a[na]
| | | | то |для диска со стержня A подготовлено место на стержне B
| | | | PutOneDisk("A", "B") |размещаем его
| | | | AlterArrays(1, na, a, nb, b)
| | | | placed:=да
| | | | иначе
| | | | |Перекладываем диски со стержня C на стержень B
| | | | |a) определяем количество перекладываемых дисков count
| | | | count:=0
| | | | poin:=nc
| | | | нц пока poin>0 и c[poin]<=b[nb]
| | | | | count:=count+1
| | | | | poin:=poin-1
| | | | кц
| | | | если count>0
| | | | | то
| | | | | |б) и перекладываем их
| | | | | MoveDisks(count, "C", "A", "B")
| | | | | AlterArrays(count, nc, c, nb, b)
| | | | все
| | | | если poin=0 или c[poin]>=a[na]
| | | | | то |Для диска со стержня A подготовлено место
| | | | | | |на стержне C
| | | | | PutOneDisk("A", "C") |размещаем его
| | | | | AlterArrays(1, na, a, nc, c)
| | | | | placed:=да
| | | | все
| | | все
| | кц при placed
| все
кон
```

Примечание. В процедуре используются сложные условия типа `poin=0 или c[poin]>=a[na]`

В то же время при `poin=0` значения `c[poin]` и `b[poin]` не определены, поскольку нижней границей индекса этих массивов является единица. Кажется, что такие условия некорректны. Однако они успешно обрабатываются компилятором школьного алгоритмического языка. Почему? При вычислении сложных условий типа *или* сначала находится значение первого условия. Если оно истинно, то второе условие не вычисляется, а сложное условие получает значение “истина”. Аналогично вычисляются сложные условия типа *и*. Если значение первого условия “ложь”, то второе условие не вычисляется, а сложное условие получает то же значение, т.е. “ложь”. Точно так же вычисляют сложные условия компиляторы языков Паскаль и Си. Однако интерпретатор Бейсика действует по-другому: сначала находятся значения обоих простых условий, а уже затем определяется значение сложного условия. К этому вопросу мы еще вернемся ниже.

На этапе 2 решения задачи (см. выше) объединение дисков со стержней **B** и **C** на одном из них также может быть проведено путем перекладывания дисков. Процедура `Join`, выполняющая объединение, во многом аналогична процедуре `Place`. Для номеров дисков на стержнях **B** и **C** используются обозначения `poin1` и `poin2`.

```
алг Join(арг рез цел nb, арг рез цел таб b[1:nb], арг рез цел nc,
         арг рез цел таб c[1:nc])
нач цел count, poin1, poin2
| poin1:=nb
| poin2:=nc
| нц пока poin1>0 и poin2>0
| | |Перекладываем диски со стержня B на стержень C
| | |a) определяем количество перекладываемых дисков count
| | count:=0
| | poin1:=nb
| | нц пока poin1>0 и b[poin1]<=c[nc]
| | | count:=count+1
| | | poin1:=poin1-1
| | кц
| | если count>0
| | | то
| | | |б) и перекладываем их,
| | | MoveDisks(count, "B", "A", "C")
| | | |одновременно уточняя массивы b и c
| | | AlterArrays(count, nb, b, nc, c)
| | все
| | если poin1>0
| | | то
| | | |Перекладываем диски со стержня C на стержень B
| | | |a) определяем количество перекладываемых дисков count
| | | count:=0
| | | poin2:=nc
| | | нц пока poin2>0 и c[poin2]<=b[nb]
| | | | count:=count+1
| | | | poin2:=poin2-1
| | | кц
| | | если count>0
| | | | то
| | | | |б) и перекладываем их
| | | | MoveDisks(count, "C", "A", "B")
| | | | AlterArrays(count, nc, c, nb, b)
| | | все
| | все
| кц
кон
```

На этапе 3 перекладывание всех дисков со стержня **B** на стержень **C** проводится с помощью процедуры `MoveDisks`.

Основная часть программы на ***школьном алгоритмическом языке*** может быть оформлена следующим образом:

```
алг Задача Ханойские башни
нач цел n, na, nb, nc, i, лит s
| |n - число дисков на стержне A в исходном положении
| |na, nb, nc - число дисков соответственно на стержнях A, B, C
| |во время перекладывания
| нц
| | вывод нс, "Введите исходное количество дисков на стержне A"
| | вывод нс, "Оно должно быть больше нуля"
| | ввод n
| кц при n>0
| цел таб a[1:n], b[1:n], c[1:n]
| |Заполняем массив a случайным образом
| нц для i от 1 до n
| | a[i]:=rnd(25)+1
| кц
| вывод нс, "Массив, моделирующий стержень A, имеет вид:"
| вывод нс, a
| вывод нс, "Чтобы получить решение - нажмите любую клавишу"
| s:=getkey()
| na:=n; nb:=0; nc:=0
| вывод нс, "Последовательность перекладывания дисков:"
| нц
| | |Размещаем по одному диску со стержня A на стержне B или C
| | Place(na, a, nb, b, nc, c)
| кц при na=0
| если nc<>n |Если не все диски размещены на стержне C
| | то
| | |Объединяем диски со стержней B и C на одном из них
| | Join(nb, b, nc, c)
| все
| если nc=0 |Если все диски собраны не на стержне C
| | то
| | |Перекладываем все диски со стержня B на стержень C,
| | MoveDisks(nb, "B", "A", "C")
| | |одновременно уточняя массивы b и c
| | AlterArrays(nb, nb, b, nc, c)
| все
| |Печатаем результат
| вывод нс, "Теперь массив, моделирующий стержень C, имеет вид:"
| вывод нс, c
кон
```


Продолжение на с. 13

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

121165, Киевская, 24
тел. 249 4896
Отдел рекламы
(095) 249 3886

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков 32291
комплекта приложений 32744

Internet: infosef@glasnet.ru
Fidonet: 2:5020/69.32
WWW: http://www.glasnet.ru/~infosef
FTP: ftp://ftp.glasnet.ru/users/infosef

Объединение педагогических изданий "ПЕРВОЕ СЕНТЯБРЯ"

Отпечатано в типографии "Пресса"

125865, Москва, ул. Правды, 24
Тираж 9000 экз.
Заказ №

Windows 95

Start | тел. | факс (095) | тел. | 24.12

№ 1 (194) январь 1999

еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

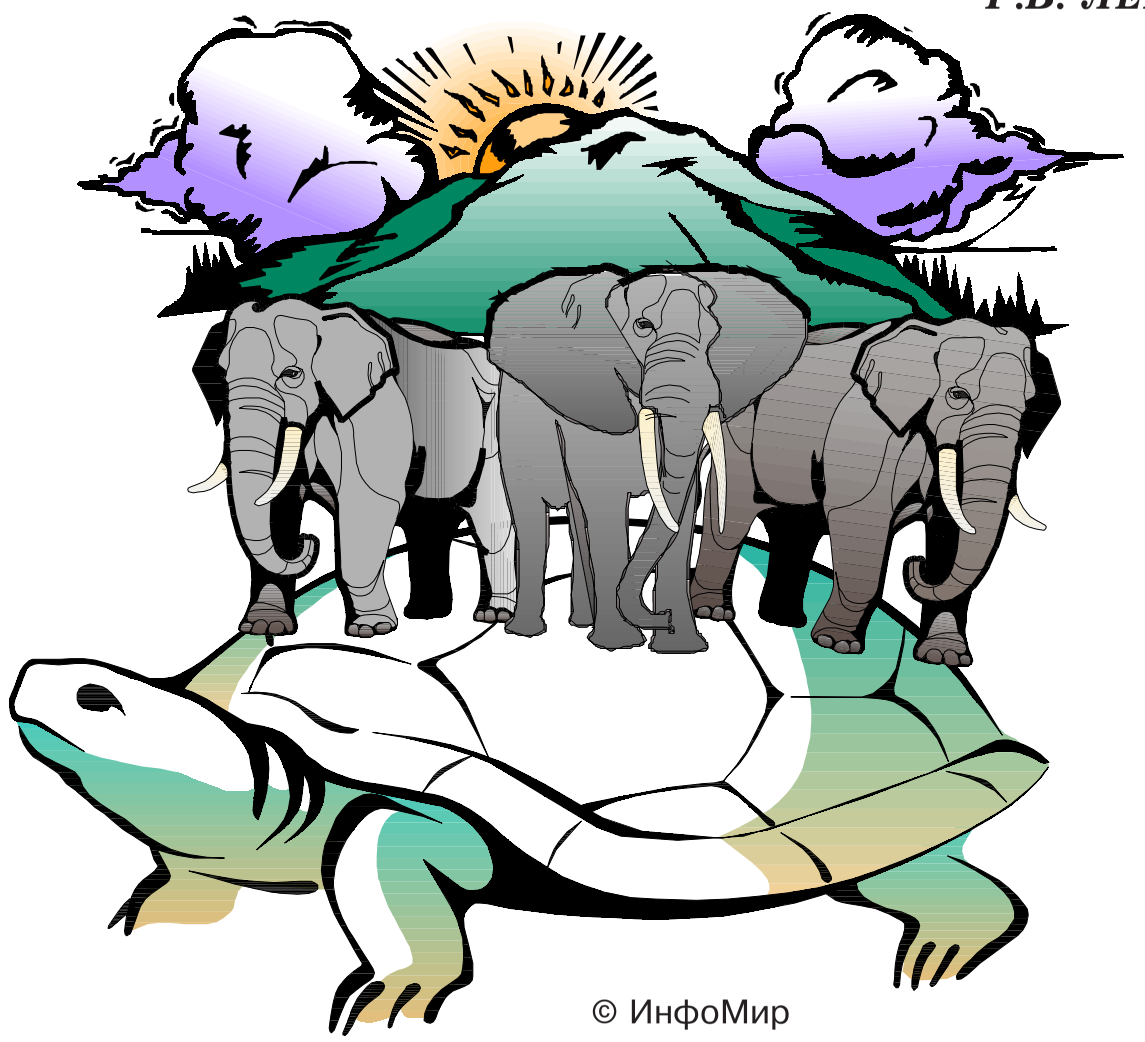
ИНФОРМАТИКА

12 лекций

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

Лекция 1

А.Г. КУШНИРЕНКО,
Г.В. ЛЕБЕДЕВ



Книга* содержит описание основных понятий, идей и целей 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), а также ряд практических советов и методических приемов по организации и проведению курса.

Особое внимание уделено формированию представлений о структуре и взаимосвязях отдельных частей курса, об их относительной значимости, о месте курса в школьном образовании, о том, что такое информатика вообще и школьная информатика в частности. Обсуждается, какие понятия "настоящей информатики" как направления науки должны войти в общеобразовательный школьный курс и почему.

Материал пособия будет интересен учителям и методистам, использующим указанный учебник, а также всем тем, кто захочет сравнить разные подходы к преподаванию школьного курса информатики или разработать свой собственный курс, независимо от того, на какой возраст этот курс будет рассчитан.

---

* Книга готовится к изданию в издательстве "Дрофа".

СОДЕРЖАНИЕ

Примеры неадекватности и адекватности.

Я здесь люблю приводить два примера. Один я заимствовал у Сергея Попова — это игротехник, организатор деловых игр, в том числе по Байкалу. Он однажды объяснял, как в России начали вводить рынок. Мы, сказал он, очень похожи на папуасов, которые увидели, что из прилетевшего самолета выгружают продукты. "Ага, — сказали папуасы, — продукты выгружают из самолета", — сделали самолет из коры и листьев, сели вокруг него и стали ждать появления продуктов. Это пример неадекватного представления об окружающей реальности.

Другая любимая мною аналогия состоит в следующем. После школьного курса физики все мы имеем некоторую физическую картину мира в голове. Мы, может быть, уже не помним ни формул, ни чему нас учили, но каждый знает, что между выключателем и лампочкой есть провода, даже если никаких проводов не видно. Всем известно, что когда щелкаешь выключателем и зажигаются лампочки — это не отдельно выключатель и отдельно лампочки; что есть провода, электростанции, есть провода к электростанциям и т.д. И каждый знает, что солнце таким образом включить-выключить нельзя. Также более или менее всем понятно, что если из выключателя торчат провода, то лучше их не трогать, потому что мы знаем что-то о токе и напряжении. Мы знаем не только о том, что видно, но кое-что о содержании, о физических величинах, процессах и пр. У нас есть — обычно достаточно адекватная — физическая картина мира.

Адекватную информационную картину мира подобного рода и должна сформировать информатика. Школьники должны представлять себе, какие примерно процессы протекают в ЭВМ, когда она что-то делает. Должны знать, что существует программное обеспечение, в котором могут быть ошибки, и т.д. Как и в примере с выключателем и лампочкой, школьник должен — пусть примерно — представлять себе информационную картину мира. Вот мы пришли в класс и включили компьютер — что при этом происходит, почему появляется Бейсик или алгоритмический язык, почему мы можем набрать алгоритм и как он выполняется, что такое компьютерный "вирус" и как он работает, что вообще в этой машине происходит? Только после формирования такого рода представления, на мой взгляд, возможны демистификация ЭВМ, сознательное использование ЭВМ как обычного средства производства, отношение к ЭВМ как к нормальному орудию труда, а не как к таинственному и загадочному устройству.

Завершая эту часть, хочу еще раз повторить, что ЭВМ — самое обычное орудие труда, как ручка, утюг, молоток и пр. И в задачу третьего "кита" входит формирование общей картины мира с учетом наличия в этом мире ЭВМ.

В школьном курсе информатики должны быть представлены основные понятия и методы информатизации и алгоритмизации.

Ну и поскольку я затронул провода, лампочки и электричество, то, чтобы потом уже не повторяться, скажу еще одну вещь. Совсем недавно бушевали страсти: чему же мы должны учить школьников в курсе информатики? Одна из точек зрения звучала так: "Зачем учить программированию, давайте учить пользоваться готовыми системами, они же не будут профессиональными программистами" — и т.д.

Здесь, как мне кажется, очень полезными являются аналогии. Давайте взглянем с этой точки зрения на такой традиционный школьный предмет, как физика. В школьном курсе физики изучают электрическое сопротивление, ток, законы Ома и другое, в общем-то не имеющее отношения к включению и выключению лампочек. И именно они формируют общую физическую картину мира. Конечно, многое потом забывается — формулы, законы, даже понятия. Конечно, лишь единицы становятся электриками, а большинство только включает и выключает свет. Тем не менее никому и в голову не приходит предложить исключить это содержание из курса физики. Даже в страшном сне не приснится курс физики, в котором на первом занятии школьники учатся включать свет, на втором — выключать, на третьем — выворачивать лампочки и т.д. (хотя преподавать такой курс сможет кто угодно и без всякой подготовки).

Так же и школьная информатика должна давать школьникам нечто большее, чем просто навыки, которые к тому же устареют еще до начала их реального применения. Использовать готовые программные системы, базы данных, электронные таблицы и системы коммуникаций каждый, если понадобится, научится столь же быстро, как включать и выключать свет.

На наш взгляд, школьные курсы физики, информатики и других дисциплин призваны закладывать основы общей культуры. В этом смысле информатика должна учить основным методам информатизации и алгоритмизации, тому, что принято в современной "большой" информатике, в науке. Именно это должно составлять базу школьного курса, а вовсе не умение нажимать на кнопки.

Литература

1. *Авербух А.В., Гисин В.Б., Зайдельман Я.Н., Лебедев Г.В.* Изучение основ информатики и вычислительной техники. М.: Просвещение, 1992, 302 с.

2. *Каймин В.А., Щеголев А.Г., Ерохина Е.А., Федюшин Д.П.* Основы информатики и вычислительной техники / Пробн. учебн. пособие для 10—11-х кл. средн. шк. М.: Просвещение, 1989, 272 с.

3. *Гейн А.Г., Житомирский В.Г., Линецкий Е.В., Сапир М.В., Шолохович В.Ф.* Основы информатики и вычислительной техники. Учебное пособие для 10—11-х классов общеобразовательных школ. М.: Просвещение, 1991, 254 с.

4. Основы информатики и вычислительной техники. Пробн. учебн. пособие для средн. учебн. заведений // *Ершов А.П., Кушниренко А.Г., Лебедев Г.В. и др.* М.: Просвещение, 1988, 207 с.

5. *Майерс Г.* Надежность программного обеспечения. (Пер. с англ. Под ред. Кауфмана В.Ш. М.: Мир, 1980.

Продолжение следует

Подробнее ко всему этому мы подойдем в конце, когда я буду рассказывать про третью главу. Сейчас я хочу вас вернуть к четырем фундаментальным понятиям информатики:

1 Команды (Циклы) → **3** Вспомогательные алгоритмы
2 Величины (Таблицы) → **4** Исполнители

Давайте посмотрим на этот список еще раз. Первые два понятия отражают методы записи действий и объектов, и в особенности записи большого количества действий и большого количества объектов. Вторые два понятия отражают фундаментальные приемы структуризации, которые человечество выработало за последние три десятка, а отчасти и за последние три тысячи лет. Все четыре понятия просты, конкретны, доступны школьникам, могут быть поняты и освоены в процессе решения задач и все вместе образуют замкнутое целое, фундамент. На нем можно развивать и внутренние способности человека к алгоритмическому мышлению, и понимание реальностей окружающего нас мира.

Итак, эти четыре понятия выписаны не случайно. Не просто потому, что я решил, что они важнее других. За этим выбором стоят фундаментальные свойства действий и величин и фундаментальные способы их записи — их организации и структуризации, которые к настоящему моменту выработало человечество.

Нужно ли выходить за пределы этих понятий?

Если эти четыре понятия освоены, то, образно говоря, мы забрались на некое "плато" современной информационной культуры. Карабкались, карабкались вверх по горным склонам, наконец добрались до плато, а дальше можно идти во многих разных направлениях (если есть потребность, желание, время). На этом "плато информатики" протоптано много тропинок, проложено много дорог. Одна дорога ведет к развитию способов конструирования структур данных: стеки, деки, графы, деревья и т.д. Другая — к конструированию новых языков программирования. Третья — к решению классов прикладных задач, например, задач трехмерной графики. Этого всего в школьном курсе может и не быть, а вот базовые понятия в нем быть должны. Именно поэтому я говорю, что эти четыре понятия обязаны быть в школьном *общеобразовательном* курсе.

Тут я должен сделать одну оговорку — я говорил про "плато информатики" в так называемой процедурной традиции информатики, процедурной традиции программирования, где важны действия и объекты и наше умение их организовывать и записывать, которая и является выражением понятия алгоритмический стиль мышления. Есть и другие традиции — функциональная и логическая, которые связаны скорее с математическим (логическим) стилем мышления и про которые я расскажу позже.

Но в процедурной традиции, там, где алгоритмический стиль мышления является ключевым, именно четыре указанных выше понятия образуют то культурное "плато", начиная с которого можно решить любую задачу, изучить любой язык программирования и т.д. Общее плато, которое обязано быть в базовой общеобразовательной подготовке любого школьника.

Одним развитием алгоритмического стиля мышления не обойтись — нужно сформировать представление о том, как ЭВМ, потоки информации, алгоритмы вписаны в жизнь современного общества.

"Адекватное представление", конечно, включает в себя еще и представление о месте информации и ЭВМ в нашем обществе, а также о том, что можно и чего нельзя, как можно и как нельзя делать с помощью компьютеров, где их разумнее применять, а где — обойтись без них и т.д.

Другими словами, мы должны сформировать у школьника некоторое представление обо всей "компьютерной" стороне жизни. Пусть упрощенное, неполное, но отвечающее реальности, адекватное представление об информационной реальности. Здесь уже другая задача, отличная от развития алгоритмического стиля мышления.

Практически это означает, что, объясняя какие-то применения ЭВМ, нужно описать их не только внешне, но и изнутри. Всякий раз, когда мы объясняем, что компьютеры используются в информационных системах, для продажи билетов или еще где-то, мы должны не только сказать, что, нажав на клавиши, можно увидеть на экране наличие свободных мест (это школьники смогут узнать и при покупке билетов), но и объяснить, как компьютер используется, какие методы обработки, представления информации при этом применяются, почему компьютер что-то может делать, а чего-то не может, какова здесь информационная модель, какие стороны жизни она отражает, а какие нет, какие алгоритмы используются для переработки информации в этой модели и т.д.

После окончания курса школьники, как мне кажется, должны себе представлять, что сделать систему для продажи билетов — хотя и сложная, но, в общем, понятная задача, а отличить по фотографии кошку от собаки с помощью компьютера — задача не только крайне сложная, но и не очень понятная. Они должны представлять себе, что и как происходит внутри компьютера и каковы сложности этих процессов.

Наконец, и совсем простыми вещами пренебрегать нельзя. Всякий сегодня знает, что 20—30 килограммов груза легко перевезти на легковом автомобиле, для 2—3 тонн понадобится уже грузовик, а для 200—300 тонн придется придумывать что-то особенное. Точно так же и в информатике нужно воспитать интуитивное ощущение "тяжести" тех или иных задач и "грузоподъемности" тех или иных моделей компьютеров.

Предисловие

Основная задача этой книги — систематизация материала учебника, демонстрация основных целей курса и методики его преподавания, демонстрация относительной значимости и сложности тех или иных понятий, формирование у учителя представления об общей структуре и взаимосвязях отдельных частей курса, о месте курса в школьном образовании, а также о том, какую часть "настоящей информатики" как науки представляет курс.

В отличие от методического пособия для учителей [1], материал настоящего пособия весьма неоднороден — одним темам уделено много внимания, другие почти не затронуты, кое-где обсуждаются вопросы "вокруг да около" курса, некоторые темы и тезисы повторяются или рассматриваются несколько раз, на разных уровнях и с разных точек зрения.

История этой книги такова. Много лет мы писали учебники информатики для школы и создавали программное обеспечение для этих учебников на механико-математическом факультете МГУ, в Академии наук и в объединении "ИнфоМир". Параллельно мы вели курсы переподготовки учителей, регулярно встречались с учителями и методистами, отвечали на их вопросы.

В результате этих встреч мы поняли: учителю и методисту негде найти ответы на важные общие вопросы о том, что такое информатика вообще и школьная информатика в частности, что мы считаем важным в школьном курсе, почему мы отобрали для учебника именно этот материал, почему он излагается в такой последовательности и т.д. Эти вопросы повторялись вновь и вновь на каждой встрече.

В 1991 году в Архангельском институте усовершенствования учителей Г.В. Лебедев прочел цикл лекций, посвященный этим вопросам. Лекции были засняты любительской видеокамерой, и комплект из 6 видеокассет разошелся по России в небольшом числе экземпляров, а Л.Е. Самовольнова изготовила компьютерную стенограмму видеозаписи.

Мы подправили полученный текст, исключили повторы, добавили новый материал, но сохранили стиль стенограммы лекций и изложение от первого лица. Мы добавили также фрагменты лекций А.Г. Кушниренко для учителей информатики, которые он читал в 1993—1995 годах.

Учитывая пожелания учителей, в приложении к этой книге мы поместили описание системы программирования на школьном алгоритмическом языке "КуМир" для IBM PC и Apple Macintosh.

Авторы глубоко благодарны Л.Е. Самовольновой, сотрудникам Архангельского ИУУ А.С. Личутину, Ю.Н. Коноплеву, И.А. Захаровой, Н.Н. Нечай и Л.Н. Нестеровой, сотрудникам объединения "ИнфоМир" А.Г. Леонову и Р.Д. Перышковой, а также многим другим не названным здесь людям за помощь и поддержку при создании этой книги.

Введение

Напомним, что изложение идет от первого лица, как в стенограмме лекций.

Построение лекций будет следующим:

— сначала я изложу основные идеи, исходя из которых учебник был написан, его главные цели и задачи;

— потом расскажу методику построения курса и постараюсь дать общее представление об учебнике в целом, об относительной важности, значимости его частей и о месте и роли понятия "исполнитель";

— далее я просто изложу учебник — параграф за параграфом, делая основной упор на методические приемы: как подавать материал на уроках, на чем акцентировать внимание, как устроить небольшое представление, трехминутную контрольную и т.д. (нумерация разделов в этой части пособия будет соответствовать номерам параграфов в учебнике);

— затем, думаю, это будет интересно, поговорю "вокруг да около" учебника: организация курса, методическое пособие, разные полезные книги, программное обеспечение, сравнение с другими учебниками, ответы на вопросы;

— далее еще раз затрону методику, но уже считая, что вы получили достаточно полное представление о курсе. А именно — расскажу про "машинную" и "безмашинную" методики и т.д.;

— потом немного поговорю о месте курса в "большой информатике": какие области "большой информатики" курс затрагивает, какие не затрагивает, как можно развивать его дальше на факультативе или при углубленном изучении;

— наконец, изложу нашу точку зрения на место курса информатики в школьном цикле (в частности, поделюсь нашими планами переноса курса в 7—8-е классы), а также (в послесловии) расскажу о том, как возникли Робот и алгоритмический язык.

Лекция 1

А. Основные цели, или Три "кита" курса

Я, естественно, рассказываю о нашем учебнике, о том понимании информатики, которое заложено в этот учебник. А связь с другими учебниками и его место в "большой информатике" мы обсудим позже.

Курс преследует три основные цели, или, образно говоря, базируется на трех "китах":

1) первый и основной "кит" называется "алгоритмический стиль мышления": **главная цель курса — развитие алгоритмического стиля мышления** как самостоятельной культурной ценности, независимо в каком-то смысле от компьютеров и всего прочего;
2) второй "кит": **курс должен быть "настоящим"**. Слово "настоящий" означает, что в процессе упрощения понятий информатики мы не должны

не в состоянии записать, что вот эти объекты будут предназначены для того-то и того-то и связаны с такими-то действиями; а другие объекты предназначены для другого и связаны с другими действиями.

Уровень Паскаля — это уровень информатики 70-х годов: вспомогательными алгоритмами (подпрограммами) уже пользовались, а аналогичные средства для работы с объектами еще не появились.

> Структурировать в информатике нужно не только действия, алгоритмы, но и саму информацию, объекты.

Я попробую пояснить структуризацию объектов на житейском примере. Допустим, вы решили сделать ремонт, дома или в школе, неважно. Нужны материалы, инструменты, рабочие. Первый подход: нанять рабочих, самому закупить материалы и инструменты и отвечать за них. Отдавая любой приказ любому рабочему, выдавать ему инструменты и материалы, а по окончании работы забирать инструменты и остатки материалов. К чему приводит такой подход? Все материалы и инструменты — ваша забота, вы за них отвечаете, вы ими постоянно манипулируете, у вас полон рот хлопот. А рабочий выполнил работу, все сдал — и без хлопот пошел домой. В каком-то смысле это похоже на использование вспомогательных алгоритмов: при вызове алгоритма ему передаются аргументы, по завершении работы от него забираются результаты, и после окончания выполнения все, что было связано с алгоритмом, стирается из памяти ЭВМ ("рабочий пошел домой"). Второй подход — нанять сразу целую бригаду со всеми инструментами и материалами, а самому с ними дела не иметь. Рабочие внутри бригады сами по мере надобности будут передавать друг другу инструменты и материалы, сами за них отвечать и разбираться с тем, что им необходимо. Бригада оказывается коллективным Исполнителем ремонта. И если вы наймете две бригады для разных целей (например, маляров и сантехников), то инструменты и материалы каждой из них будет хранить она сама и между собой они не перепутаются.

В информатике второй подход обеспечивает понятие исполнителя — хранителя какого-то набора объектов. Исполнитель и есть недостающее в классическом Паскале средство структуризации.

> Все вновь создаваемые языки программирования имеют средства структуризации объектов, а все классические языки такими средствами дополняются.

Насколько мне известно, первый язык, в котором появились способы такой "структуризации" объектов, — это язык Simula-67. Как следует из названия языка, предназначен он был для информационного моделирования ("симулирования") различных процессов. Примерами процессов могут служить ремонт школы, воздушный бой или работа компании по продаже авиабилетов. Позже подобные конструкции для структуризации объектов появились практически во всех других языках программирования.

Сейчас средства структуризации объектов есть во всех современных языках программирования, но называются они везде по-разному. Здесь пока нет единой терминологии. Наиболее распространенный термин, используемый в научных журналах и рекламных объявлениях, — объектно-ориентированное программирование. Термин подчеркивает, что первичны, вообще говоря, не действия, не алгоритмы, а объекты, с которыми алгоритмы работают. А на практике важны и алгоритмы, и объекты: надо уметь структурировать вместе и информацию, и обрабатывающие ее алгоритмы. Для этого языку нужны средства выделения некоторого набора объектов (величин в нашей терминологии) и действий, которые над ними производятся. И эти средства должны позволить все это назвать одним именем и впоследствии использовать целиком, подобно бригаде маляров при ремонте.

В нашем учебнике соответствующее понятие называется информационной моделью исполнителя, или просто исполнителем, и вводится в третьей главе. В других языках это понятие называется по-разному, скажем, в языке Ада — пакетом (*package*), в языке Модула — модулем, в C++ и Simula-67 — объектом и экземпляром класса; классический Паскаль расширен конструкцией Unit, в классическом C для этих целей используется свойство локализации объектов в файлах. Достаточно широко распространено слово "модуль", хотя это слово применяется и для других целей. Например, в книге Майерса [5] дается классификация порядка 15 типов разных модулей и обсуждаемое нами понятие называется "информационно-прочным" модулем.

Здесь пока нет единой, общепринятой терминологии. Важно то, что во всех современных языках теперь такая конструкция есть, ибо она фундаментальная и без нее не обойтись. Поскольку, однако, единой терминологии до сих пор нет, мы с вами, следуя учебнику, будем говорить "информационная модель исполнителя", или просто "исполнитель".

> Фундаментальное понятие информационной модели в школьных курсах информатики, в том числе и в нашем курсе, отражено недостаточно.

Я тут столько языков программирования назвал, что у вас могло создаться впечатление, что понятие информационной модели, быть может, важно и нужно, но вовсе не в общеобразовательном курсе информатики, а в профессиональном программировании. Это не так. Умение строить информационные модели — одна из базисных компонент алгоритмического стиля мышления. Задачи по придумыванию информационных моделей увлекательны и поучительны не меньше, а может, даже и больше, чем задачи на составление алгоритмов. Такие задачи должны занять подобающее место в школьном курсе.

других, их нужно снабдить именами. В памяти современной ЭВМ могут храниться значения миллионов числовых величин. Снова жизни не хватит, чтобы придумать для каждой из них имя. Таким путем задействовать большой объем памяти ЭВМ не удастся. На выручку приходят табличные величины (массивы). Такая величина имеет только одно имя и много элементов, занумерованных целыми числами. Поскольку элементы памяти ЭВМ также имеют свою внутреннюю нумерацию, понятие табличной величины (массива) хорошо соответствует структуре памяти современных ЭВМ и есть во всех современных языках программирования.

Замечу, что сами методы и понятия компактной (короткой) записи большого количества действий или компактного обозначения больших объемов информации носят фундаментальный характер и могут изучаться безотносительно ЭВМ, хотя развитие этих методов было вызвано появлением ЭВМ и их указанными выше особенностями.

Итак, мы разобрали, почему в наш список вошли первые два понятия: 1) команды, и особенно циклы, и 2) величины, и особенно табличные величины. В принципе этими двумя понятиями можно было бы ограничиться, и получился бы вполне замкнутый курс, я бы сказал, нулевого уровня. Кстати, Бейсик идейно близок именно этому уровню знаний.

Можно ли ограничиться понятиями “цикл” и “табличная величина”?

А нужны ли остальные понятия? Ответ на этот вопрос не так прост. Набор из первых двух понятий, простой и замкнутый, хорошо отвечает структуре ЭВМ. Есть язык программирования — Бейсик, прекрасно поддерживающий эти два понятия и отвечающий требованиям 60-х годов. Но с точки зрения теории и практики информатики сегодняшнего дня этот набор из двух понятий мал, алгоритмический стиль мышления отражает весьма бедно и не охватывает фундаментального для этого стиля понятия “структуризации”.

Способность структуризации — фундаментальное свойство человеческого мышления.

Лучше всего мне, наверное, снова прибегнуть к аналогиям. Представьте себе математику, где каждое утверждение доказывается исходя из аксиом от начала до конца. Любое доказательство превращается в огромный текст, и понятно, что так работать нельзя. Точнее, существует естественный порог сложности задач, которые можно решить таким образом.

То же произойдет, если не вводить в русский язык новые слова. Попробуйте представить себе человека, ничего никогда не слышавшего о метро, и попробуйте описать все то знание (поезда, эскалаторы, под землей, но не всегда — и т.д.), которое в вашей голове связано с этим словом. А теперь представьте себе, что вы в обычном разговоре всякий раз вместо слова “метро” используете это описание из сотни или тысячи слов!

Для всех случаев, когда встречается проблема такого разрастания описаний, человечество придумало способы сбора, накопления и компактификации знаний. Если в русском языке появляется интенсивно используемое понятие, то его называют новым словом. Если в математических формулах начинает повторяться какой-то громоздкий фрагмент, то для него вводится новое короткое обозначение. Если в математике начинает повторяться одно и то же рассуждение, то оно формулируется в виде теоремы. Теорема получает имя, на которое отныне достаточно просто сослаться, не повторяя доказательство.

Все это называется структуризацией. Структуризация в языке позволяет коротко назвать, обозначить то, что раньше требовало длинного описания. Структуризация в математике дает возможность доказывать утверждения исходя не только из аксиом, но и из ранее доказанных утверждений, т.е. задействовать знания, полученные ранее.

Способность задействовать сделанное раньше (может быть, даже предшествующими поколениями) — одно из фундаментальных свойств человеческой культуры, и оно проявляется почти во всех областях человеческой деятельности.

Поэтому для решения современных задач, для развития алгоритмического стиля мышления в его современном понимании необходимо овладеть понятиями и методами структуризации в информатике.

Способы структуризации в информатике опираются на понятие вспомогательного алгоритма.

Если в информатике мы ограничимся циклами и величинами, то будем вынуждены любую задачу доводить до базовых конструкций языка программирования и писать любой алгоритм, любую программу одним куском. Как и в математике без теорем, в информатике без каких-то способов структуризации серьезно работать не удастся. И в информатике придуманы такие способы, позволяющие использовать ранее сделанные вещи как целые, как готовые.

Для действий таким структурирующим понятием является понятие “вспомогательного алгоритма”, или “подпрограммы” с параметрами. Мы можем некоторую последовательность действий записать отдельно, сказать, что это вспомогательный алгоритм (подпрограмма), и потом им пользоваться. Здесь полная аналогия с леммами и теоремами в математике.

Поскольку вспомогательный алгоритм, или подпрограмма, — понятие к настоящему моменту широко известное, я думаю, здесь у вас никаких особенных вопросов возникать не должно.

Действия → Команды (Циклы) → Всп. алгоритмы
Объекты → Величины (Таблицы) → ?

Кстати, язык Паскаль, который в свое время придумал Н.Вирт, содержит именно эту тройку понятий, т.е. действия можно разбить на вспомогательные алгоритмы (подпрограммы, процедуры). С объектами в Паскале хуже — их трудно разделять по назначению. Мы

ны вместе с водой выплеснуть и ребенка, т.е. упрощать можно лишь до тех пор, пока не теряется содержание, суть дела;

3) третий “кит”: **курс должен формировать “адекватное представление о современной информационной реальности”**. Это означает некоторую замкнутость, законченность, достаточность набора понятий курса. Другими словами, если второй “кит” запрещает в процессе упрощения переходить к чему-то, удобному для изложения, но не имеющему отношения к “настоящей информатике”, то третий “кит” требует, чтобы адекватное представление об информатике было тем не менее сформировано, чтобы материала было достаточно и курс содержал необходимый для этого набор понятий, покрывающий сегодняшние реальности.

Забегая вперед, скажу: эти три “кита”, как водится, базируются на “черепахе”. Наш подход предполагает большую работу и учителей, и учеников — и тогда что-либо из курса будет воспринято. Ведь это не “беллетристический” курс: слушать учителя, читать учебник и отвечать у доски недостаточно. Предполагается, что ученик будет много работать, решать задачи. Мы уверены: только так он сможет что-то усвоить. Можно сказать, что “черепахой” курса является русская пословица: “Без труда не вытащишь и рыбку из пруда”.

Для школы это не новость — в математике и в физике все тоже рассчитано на регулярную работу ученика. Посмотрите на коллег-математиков: вечно они с тетрадками, с контрольными — и иначе никак нельзя.

Сейчас я раскрою смысл слов и понятий “развитие алгоритмического стиля мышления”, “информатика должна быть “настоящей” и др., а потом, когда буду излагать учебник параграф за параграфом, покажу, как это все в нем воплотилось.

А1. Главная цель курса – развитие алгоритмического стиля мышления

Существует особый, алгоритмический, стиль мышления, и его развитие есть важная общеобразовательная задача.

Итак, первое и основное: алгоритмический стиль мышления.

Мы исходим из того постулата, что человек может мыслить по-разному, что существуют разные стили мышления. Математика и логика развивают математический (или логический) стиль мышления, т.е. умение рационально рассуждать, пользоваться математическими формулами в рассуждениях, из одних утверждений логически выводить другие (теоремы из аксиом и уже известных теорем). Литература и история, наоборот, связаны с какими-то менее рациональными аспектами сопереживания, этики и морали (этот стиль мышления условно можно назвать “гуманитарным”). Ряд школьных предметов — таких, как астрономия или география — вообще не нацелен на развитие какого-либо стиля мышления (цель этих предметов — изложение знаний в конкретной области, расширение кругозора учащихся).

Первое мое утверждение состоит в том, что есть особый, алгоритмический, стиль мышления. Как всегда, когда мы затрагиваем такие фундаментальные понятия, я не могу вам дать точного определения, что такое алгоритмический или что такое логический стиль мышления. Я просто чуть позже на наглядно-интуитивном уровне покажу, что я под этим понимаю.

А сейчас — следствие из этого утверждения. Если существует (а мы считаем, что существует) такой особый, алгоритмический, стиль мышления человека, то его развитие представляет самостоятельную ценность, так же как и развитие мышления человека вообще. Мы должны развивать все стороны мышления, какие только сможем выделить. И если в некоторой области удается выделить характерный для нее стиль мышления человека (умение думать), по моему мнению, развитие должно объявляться самоцелью и внедряться в общее образование как необходимый элемент общей культуры.

И поэтому учебник весь нацелен на то, что надо развивать алгоритмический стиль мышления, умение думать алгоритмически, а тем самым и умение думать вообще.

Математика как курс, в рамках которого преимущественно формируется логический стиль мышления, и информатика как курс, специально нацеленный на развитие алгоритмического стиля мышления, должны в обязательном порядке входить в общеобразовательные базовые курсы средней школы. Более того, основы алгоритмизации, как и основы логики, в идеале должны закладываться на ранних стадиях обучения, до 12 лет.

Цель курса — развить алгоритмический стиль мышления. Умение обращаться с компьютером или знание конкретных программных систем не входит в непосредственные цели курса.

Обратите внимание, я не говорю, что цель курса — научить школьников работать за компьютером, обучить их языкам программирования, современным базам данных или электронным таблицам. Все это полезные и важные вещи, но наш курс построен совсем по-другому и нацелен совсем на другое — на развитие в каком-то смысле интеллекта ученика, его умения думать. Вот это — первое и основное.

У нас сейчас много разных учебников и, я думаю, будут появляться все новые и новые, и большинство из них — хорошие. Но когда вы выбираете тот или иной учебник, вы должны, на мой взгляд, исходить в первую очередь из целей учебника, сопоставляя их с теми целями, которые перед курсом информатики ставите вы.

Если, например, вы считаете, что вам надо, чтобы ученики знали Бейсик и умели работать за компьютерами, пользоваться базами данных или электронными таблицами, то наш учебник вам почти бесполезен.

Наш курс в этом смысле больше похож на предметы физико-математического цикла: на первом плане боль-

шое количество задач, умение решать эти задачи, умение думать, владение соответствующими методами. А компьютер стоит как-то немного в стороне и играет вспомогательную роль.

Я еще и еще раз повторяю: главная и основная цель учебника — развитие алгоритмического стиля мышления как общей культуры ученика. Учебник не нацелен ни на знакомство с языками программирования, ни на знакомство с системами программирования, ни на знакомство с компьютерами как таковыми. Изучение работы на компьютерах, языков и систем программирования не входит в основные цели нашего учебника.

Что такое алгоритмический стиль мышления?

Я часто повторяю слова "алгоритмический стиль мышления", а что это значит — каждый понимает по-своему. И хотя обычно имеется некоторое общее представление о том, что это такое, я хочу привести пример, демонстрирующий, что алгоритмический стиль мышления вообще существует. Не давая точного определения, я попытаюсь показать, что стоит за словами "алгоритмический стиль мышления", сформулировать проблемы, задачи, для решения которых человек должен мыслить алгоритмически, рассмотреть какие-то рассуждения, характерные для данного стиля мышления, привести методы решений "чисто алгоритмических" задач. В учебнике для этого используется специально сконструированная среда — Робот на клетчатом поле. И я тоже буду использовать ее.

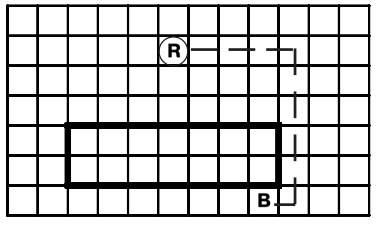


Рис. 1. Робот (R) и пульт управления им

Пусть мы имеем вертикальную металлическую клетчатую стену (в учебнике — "клетчатое поле") с выступающим прямоугольным "препятствием", а на стене — несколько выше препятствия — в одной из клеток находится Робот (R). Робот — это машинка с антенной, батарейками, моторами, магнитными присосками и т.п. И пусть у нас в руках имеется пульт радиоуправления Роботом с кнопками: "→", "←", "↑", "↓" (см. *рис.* 1).

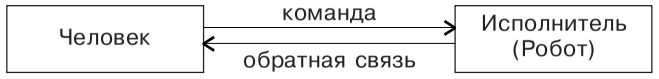
Нажимаем на кнопку "→" — Робот смещается на клетку вправо, нажимаем на кнопку "←" — смещается влево, на "↑" — вверх, на "↓" — вниз.

Простейшую задачу управления без каких-либо затруднений решает каждый школьник.

Если такую машинку — радиоуправляемый Робот — принести в класс, прикрепить к клетчатой доске и дать ученику пульт управления, то любой ученик в состоянии [illegible] так, чтобы Робот, объезжая препятствие, оказался под ним. Тут, я повторяю, нет вообще ничего сложного. Даже ребенок 5—7 лет в состоянии это проделать.

Такой стиль "общения" (работы) с электронными устройствами называется "непосредственным управлением": я нажимаю на кнопку, смотрю на результат. Нажимаю на другую кнопку, смотрю, что получилось, принимаю решение, на какую следующую кнопку нажать, и т.д. —

СХЕМА НЕПОСРЕДСТВЕННОГО УПРАВЛЕНИЯ



Итак, в этой схеме человек нажимает на кнопки, а "исполнитель" выполняет действия. Глобальный план действий (как и куда двигаться) человек может придумывать по ходу, по мере выполнения команд и прояснения ситуации.

> **МЕТОДИЧЕСКОЕ ОТСТУПЛЕНИЕ.** Поскольку вы все здесь учителя и методисты, я сразу скажу, как все это может быть организовано на уроке. Пока у нас нет Робота "в металле", учитель может играть его роль — рисуя Робота и клетчатое поле на доске. Удобно также использовать магнитные доски — на них легко перемещать Робота, и не надо постоянно стирать и рисовать его положение.
>
> Итак, на уроке — даже в младших классах — можно поднять самого слабого ученика и сказать: "Иванов, командуй!". И он без труда, глядя, как вы выполняете его команды (перемещаете Робота по доске), продиктует вам последовательность команд "вниз", "вправо" и т.д., по которым Робот обойдет препятствие.

Более сложная задача управления с использованием команд "обратной связи" также не вызывает затруднений.

Теперь немного усложним задачу. Будем считать, что Робот — в соседней комнате или вообще далеко от нас (т.е. мы его не видим), а у нас на пульте есть специальные кнопки — "?←","?→", "?↑", "?↓" — и лампочка. Нажимаем на кнопку "?↓" — Робот анализирует, можно ли сделать шаг вниз, и если вниз шаг сделать можно, то лампочка на пульте загорается зеленым цветом (если нельзя — то красным). Итак, нажали на кнопку "?↓", если зажегся зеленый свет — значит, снизу свободно, а если красный — значит, снизу препятствие.

1) команда (оператор, управляющая конструкция), и в первую очередь — команда повторения (цикл);
2) величина (переменная, объект), и в первую очередь — табличная величина (массив);
3) вспомогательный алгоритм (подпрограмма, процедура) с параметрами;
4) "информационная модель исполнителя" (как это изложено в 3-й главе нашего учебника), или просто "исполнитель" (пакет, модуль, объект, экземпляр класса).

Конечно, когда очередной лектор (сейчас — я) выписывает такой набор и говорит, что вот это — 4 основных понятия информатики, у вас немедленно должна появиться мысль, что другой лектор написал бы 7 других "основных понятий", третий добавил бы еще пару и т.д. Вообще возникает вопрос о роли личности лектора в выборе этих понятий. Так вот, понятия, которые я выписал, не случайны и выбраны отнюдь не по принципу "нравятся они мне или не нравятся". Они являются не исходной точкой, а результатом некоторого рассуждения, доказательства. И я сейчас приведу набросок этого рассуждения — обоснования, что именно эти четыре понятия являются фундаментом современного алгоритмического стиля мышления.

От действий над объектами к их записи — командам и величинам.

Мы можем начать с устройства ЭВМ. Как вы знаете, машина состоит в основном из процессора и памяти. Процессор выполняет действия, память хранит информацию об объектах и величинах, с которыми работает процессор. Все остальное (принтер, экран, дисковод и пр.) отличается от Робота только системой команд. Устройству дается команда, оно ее выполняет. А вот процессор с памятью — это особые, фундаментальные компоненты ЭВМ. Им соответствуют два фундаментальных свойства ЭВМ — умение выполнять действия и умение хранить информацию в памяти машины.

Можно, впрочем, не апеллируя к устройству ЭВМ, начать сразу с содержательного уровня действий и объектов. Я думаю, любой специалист по информатике согласится с тем, что объекты и совершаемые над ними действия — две самые фундаментальные сущности управления.

Мы, однако, занимаемся не просто непосредственным управлением, а информатикой. Поэтому объекты и совершаемые над ними действия нам надо как-то записывать в виде программ (алгоритмов). В школьном языке для записи действий используются команды (в других языках они называются операторами, или управляющими конструкциями), а для записи объектов — величины (в других языках называемые переменными, или объектами):

Действия → Команды
Объекты → Величины

Запись большого количества действий и работа с большими объемами информации.

Возникновение информатики как науки тесно связано с появлением ЭВМ по той причине, что ЭВМ умеет быстро выполнять действия и хранить много информации. Соответствующие характеристики ЭВМ называются "быстродействием" и "объемом памяти". Чтобы задействовать эти возможности ЭВМ, необходимо научиться записывать выполнение большого количества действий над большими объемами информации. Соответствующие методы компактной (короткой) записи длинных последовательностей действий и компактной записи больших объемов информации и стали зачатками информатики. Запись простого действия в виде команды или запись объекта в виде простой величины обычно не вызывает труда. Информатика как наука начинает работать, когда нужно записать длинную последовательность действий или обработать большой объем информации. Для этого в языках программирования используются конструкции циклов (короткая запись большого количества действий) и величины, состоящие из многих элементарных объектов, прежде всего массивы (в школьном языке названные "табличными величинами"):

Действия → Команды → Циклы
Объекты → Величины → Табличные величины

Цикл — это команда, позволяющая коротко записать длинную последовательность действий. Табличная величина — это кратко записываемая величина, которая дает возможность хранить много информации.

"Циклы" и табличные величины соответствуют устройству современных ЭВМ.

> Отвлекаясь в сторону, давайте рассмотрим еще раз взаимосвязь этих понятий с устройством современных ЭВМ.
>
> Чтобы задействовать "быстродействие", надо уметь приказать ЭВМ выполнить громадное количество действий, скажем, миллион или миллиард. Если мы начнем эти действия выписывать: первое, второе, третье, ... — жизни не хватит. Какой прок в быстродействии миллион операций в секунду, если приказ на каждую операцию мы будем записывать минуту! Другое дело, если мы напишем небольшую последовательность действий, скажем, десять или сто команд, но прикажем компьютеру выполнить ее много раз — миллион или миллиард. Тут мы сможем задействовать быстродействие в полной мере. Такая запись многократного выполнения заданной последовательности действий и называется циклом.
>
> Теперь давайте поговорим о памяти ЭВМ. Память состоит из отдельных элементов, имеющих свою внутреннюю нумерацию. Но записываем мы алгоритм на каком-то языке, а в языках имеем дело с отдельными порциями информации. Эти порции называются переменными, объектами или с легкой руки А.П. Ершова — величинами (по аналогии с математикой и физикой). Чтобы работать с величинами и отличать одни от

Еще один пример “ненастоящести” — неформальные алгоритмы из первых школьных учебников [4] типа алгоритма перехода улицы или алгоритма заваривания кофе. Беда в том, что в этих задачах заранее четко не задавался набор базовых команд, которыми можно пользоваться и которые дальше уже детализировать не надо. Это сразу переводит задачу из информатики в художественно-психологическую область — никаких формальных критериев для проверки и оценки ответа не существует, ответ просто должен совпадать с мнением учителя. Вообще изложение понятия исполнителя на неформально-художественном уровне — одна из основных ошибок первых учебников. О системе команд исполнителей в этих учебниках упоминалось, но ни одна система команд не приводилась. Алгоритмы были либо изложением чего-то из математики, либо чисто художественными и потому легко заменимыми на любые другие. (Вспомните мучения учителей, пытавшихся объяснить “неправильность”, например, “алгоритма” заварки кофе “пойти в бар и заказать чашечку” или “алгоритма” выхода из кинотеатра “идти за всеми”).

Возвращаясь к “настоящести”, хочу еще раз подчеркнуть, что “настоящесть” состоит в том, что мы не должны преподавать (как бы приятно или легко это ни было) понятия, не имеющие отношения к реальности, придуманные ради легкости их изложения; мы не должны решать придуманные задачи, которые не имеют отношения к информатике. Мы не должны “вместе с водой выплескивать и ребенка”. Можно упрощать под школьный возраст только до тех пор, пока не теряется суть. То, что нельзя преподать адекватно, не нужно преподавать вообще. Лучше не преподавать ничего, чем преподавать ерунду, не имеющую отношения к действительности.

А3. Курс должен формировать адекватное представление о современной информационной реальности

Третий “кит” — это другая сторона медали. Но если второй “кит” — курс должен быть “настоящим” — запрещает придумывать новые понятия ради легкости изложения, то третий требует, чтобы некоторое минимальное множество понятий и методов, достаточное для формирования представления о современной информационной реальности и современной информатике, было изложено и доведено до учеников, как бы сложно это ни оказалось.

> Развивать тот или иной стиль мышления не обязательно только научными методами.

Математический (логический) стиль мышления можно развивать не с помощью современных курсов математики, а на кубиках, головоломках, занимательных задачках и т.д. Точно так же алгоритмический стиль мышления можно начинать развивать с задачек типа “Волк, коза и капуста”, “Ханойская башня” или с игры в “Черный ящик”, где, наблюдая, как информация поступает на “вход” Черного ящика, и видя, что получается на “выходе”, нужно угадать правило (алгоритм) преобразования информации.

Примечание. Более подробно о Черном ящике вы прочитаете в § 1 лекции 3.

Но отдельными занимательными задачками, развивающими алгоритмический стиль мышления, ограничиться нельзя. Курс информатики должен охватить некоторый замкнутый, самосогласованный набор понятий, взятый из “настоящей информатики”. Чтобы лучше объяснить, что я имею в виду, снова сошлюсь на устоявшийся школьный курс — курс математики.

> Школьный курс математики — часть настоящей математики. Школьный курс информатики должен быть частью настоящей информатики.

Современный школьный курс математики, формируя представление о современной математической культуре, не может ограничиться только головоломками и развитием математической сообразительности, а вынужден охватывать формулы алгебры, аксиомы геометрии, теоремы, леммы, доказательства и др., составляющие вместе замкнутый курс элементарной математики. Этот курс, конечно, намного проще, чем все здание “большой” математики или курс высшей математики в вузе, но тем не менее он “настоящий” — и академик-математик, и инженер-конструктор, и восьмиклассник решают квадратное уравнение по одной и той же формуле и используют одну и ту же математическую символику.

Точно так же и курс школьной информатики должен быть частью информатики “настоящей”. А это значит, что мы не можем в школьном курсе остановиться на уровне задач типа “Волк, коза и капуста”. Такие задачи к “большой информатике” имеют примерно такое же отношение, как головоломки — к “настоящей математике”. Да, есть интересные, занимательные задачки, решение которых развивает алгоритмический стиль мышления, их, конечно же, надо использовать как для повышения интереса учащихся, так и просто для разнообразия. Но тем не менее если мы говорим про курс информатики в школе, то он должен относиться к “настоящей информатике” вне школы примерно так же, как математика — к “математике”, физика — к “физике” и т.д.

Таким образом,
ный курс знакомил с необходимым, пусть минимальным, набором фундаментальных понятий информатики сегодняшнего дня. Как это ни удивительно, но такой набор невелик — всего четыре понятия — и определен вполне однозначно. Сейчас я их сформулирую.

> Четыре основных понятия школьного курса информатики.

В плане развития алгоритмического стиля мышления фундаментальных базовых понятий всего четыре:

Наша задача: не видя ничего, кроме пульта управления, заставить Робота спуститься под препятствие (расстояние от начального положения Робота до препятствия неизвестно). Происходит незначительное усложнение: мы не видим обстановку, мы должны себе ее воображать и принимать решение по миганиям лампочки. Возможно, не с первой попытки, уже не так мгновенно, как в первом случае, но практически все ученики такую задачу решат.

Как? Известно, что Робот стоит где-то выше препятствия, обстановку не видно, размеры препятствия неизвестны. Что надо делать? Надо шагать вниз, пока не дойдем до препятствия, т.е. при каждом шаге проверять (нажимая кнопку “? ↓”), свободно ли еще снизу (зеленый свет) или уже есть препятствие (красный). Как только загорится красный свет (препятствие), надо начать шагать вправо, при каждом шаге проверяя (нажимая кнопку “? ↓”), не кончилось ли препятствие. Потом спускаться вниз, проверяя наличие препятствия слева (кнопка “? ←”). И, наконец, сделать один шаг влево, чтобы оказаться под препятствием. Такие последовательные нажатия на кнопки — даже с анализом невидимой и неизвестной обстановки — доступны любому школьнику.

> **МЕТОДИЧЕСКОЕ ОТСТУПЛЕНИЕ.** На уроке учитель может нарисовать обстановку у себя на листке, никому не показывать и предложить ученикам командовать. В ответ на “снизу свободно” (“? ↓”) лучше сразу говорить “да” или “нет” вместо слов “зеленый” и “красный”.
>
> Другой вариант — вызвать одного ученика, поставить его спиной к доске, на доске нарисовать обстановку и Робота (чтобы видел весь класс, а этот ученик не видел) и предложить ему не глядя командовать, а самому с помощью мела и тряпки исполнять поступающие команды.
>
> Если обход препятствия “втемную” кажется слишком сложным или длинным, то можно рассмотреть только первую часть задачи — “спуститься вниз до препятствия”.
>
> На этом же примере можно показать типичные ошибки. Если ученик начинает с команды “вниз”, а не с вопроса “снизу свободно?”, то в ответ на первую же команду учитель вместо “сделано” может сказать: “отказ — Робот разбился” (и тем самым дать ученикам понять, что проверять обстановку надо перед перемещением, а не после).

Я повторю, что кто-то, быть может, сначала ошибется (например, уведет Робота вбок от препятствия), но тем не менее после 2—3 попыток подавляющее большинство учеников (во всяком случае начиная с 7-го класса) с такой задачей справится. Это по-прежнему “непосредственное управление”: я нажимаю на кнопки, смотрю на ответ (лампочку), нажимаю на другие кнопки и т.д.

> **ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ.** Конечно, дети не родились умными и раньше они тоже не были такими. Просто раньше им приходилось решать большое число подобных задач: когда они гремели погремушкой, потом возились в песочнице, позже — запихивали в портфель книжки, учились звонить по телефону и управлять лифтом или радиоприемником. А вот в ситуации, к рассмотрению которой мы сейчас перейдем, — в задаче “программного управления”, — школьнику не на что опереться, такой
> оказывается трудной.

> Записать или объяснить кому-нибудь алгоритм труднее, чем выполнить работу самому.

А вот если теперь школьника попросить: “**Запиши как-нибудь** всю последовательность нажатий на кнопки для обхода препятствия неизвестных размеров, находящегося где-то ниже Робота”, — то тут-то и выяснится, что значительная часть учеников:

а) прекрасно представляет, на какие кнопки и как надо нажимать, чтобы заставить Робота препятствие обойти, но

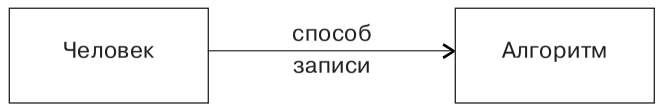
б) не в состоянии четко описать (записать) эту последовательность действий.

> Нет ничего удивительного в том, что алгоритм легче выполнить, чем записать.

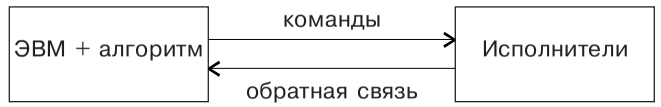
Вы можете либо называть эту запись алгоритмом или программой, либо никак не называть. Вы можете предложить записать это хоть на русском языке, хоть на школьном алгоритмическом, хоть на Бейсике, хоть на языке, придуманном самим школьником, сути дела это не изменит. А суть в том, что если раньше человек прямо нажимал на кнопки (схема “непосредственного управления”), то теперь он пишет программу (алгоритм), которая далее будет выполняться без его участия — обычно на ЭВМ. Другими словами, если раньше я сам нажимал на кнопки, то теперь пишу программу, которую отдаю ЭВМ, и дальше уже ЭВМ “нажимает на кнопки” и командует исполнителем:

СХЕМА ПРОГРАММНОГО УПРАВЛЕНИЯ

Составление алгоритма



Выполнение алгоритма



Итак, сначала человек выбирает какой-то способ записи, какой-то язык и записывает алгоритм. Затем (это на нашей схеме вообще не отражено) алгоритм как-то попадает к ЭВМ. И, наконец, ЭВМ начинает командовать Исполнителем в соответствии с алгоритмом, полученным от человека.

Почему же алгоритм трудно записать?

Это трудно по трем причинам.

Во-первых, алгоритм придется сразу ("наперед") продумать во всех деталях, ничего нельзя отложить на потом — ведь выполнять алгоритм будем уже не мы, а ЭВМ.

Во-вторых, мы должны не только все продумать "наперед" во всех мыслимых вариантах, но и записать это без двусмысленностей и выражений типа "и т.д.".

В-третьих, выполнять алгоритм будет ЭВМ — техническое устройство. Оно не может догадаться, что мы что-то "имели в виду", — все должно быть описано явно, точно, формально и на понятном ЭВМ языке.

Алгоритм — план будущей деятельности, записанный в заранее выбранной формальной системе обозначений (нотации). Составляет алгоритм человек, а выполняет ЭВМ.

Введение **записи** будущих действий с помощью какой-то нотации (в нашем учебнике используется школьный алгоритмический язык, в учебнике [2] — Пролог и Бейсик, в учебнике [3] — Бейсик и некоторая разновидность алгоритмического языка) — это и есть переход от непосредственного управления к программированию. Здесь и возникают сложности. Человек может прекрасно представлять себе, на какие кнопки и как надо нажать для получения результата, но иногда не в состоянии *записать* эту последовательность команд в виде программы. Те, кто уже преподавал информатику, я думаю, неоднократно сталкивались с подобной ситуацией.

Повторю, что трудности не в форме записи, не в языке. Какую бы нотацию мы ни ввели, коль скоро мы от непосредственного управления перешли к программированию, т.е. к записи плана *будущих* действий, нам придется предвидеть эти будущие действия во всем многообразии возникающих вариантов. Именно здесь и возникают сложности.

И вот тот стиль мышления, методы и способы, которые необходимы для перехода от непосредственного управления к программному, от умения *сделать* к умению *записать* алгоритм, я и называю алгоритмическим стилем мышления. Повторю, что это не строгое определение. Я лишь привел пример ситуации, которая показывает, что алгоритмический стиль мышления существует. Его можно и нужно тренировать, развивать, и именно этому посвящен наш учебник.

Исходя из цели развития алгоритмического стиля мышления ясно, что необходимо включить в общеобразовательный курс информатики, а без чего можно обойтись.

Теперь еще одно лирическое отступление, связанное по-прежнему с первым "китом" — алгоритмическим стилем мышления. Чтобы было понятнее, я специально буду несколько категоричен, буду, скажем так, "перегибать палку".

Во-первых, из того, что целью мы считаем развитие алгоритмического стиля мышления, вытекает, что для нас не так важно, есть компьютер или нет. Я хочу это особо подчеркнуть, хотя вовсе не призываю вас преподавать без компьютера. Логика тут простая. Если нет компьютера, то научиться пользоваться им, конечно, нельзя. Но ведь такой цели мы и не ставим. Вы все знаете известный пример про обучение плаванию без воды в бассейне — действительно, нельзя научить учеников пользоваться компьютерами, научить их работать с системами, не имея оных.

Но это не относится к развитию алгоритмического стиля мышления. **Для развития мышления нужна только голова.** Развивать алгоритмический стиль мышления можно без компьютера и даже независимо от того, есть какие-либо компьютеры в мире или нет. Подобно тому как электрическое сопротивление и закон Ома можно изучать независимо от того, придуманы ли уже электромоторы и телевизоры, курс основ информатики можно изучать независимо от того, придуманы ли уже ЭВМ или еще нет. И зачатки информатики — задачки типа "Волка, козы и капусты" или "Ханойской башни" существовали испокон веку. Такие задачки развивали именно алгоритмическую сторону мышления, развивали в меру тех потребностей, которые существовали в обществе. Сейчас, с развитием компьютеров, эти потребности резко возросли, информатика из "Волка, козы и капусты" превратилась в отдельную научную область с массой своих методов, направлений, подразделений и т.д. И, как видите, дело дошло уже до отдельного курса в школе.

Нельзя научиться решать алгоритмические задачи, изучая устройство компьютеров и языки программирования.

Во-вторых, поскольку речь идет о развитии алгоритмического стиля мышления, то компьютеры из предмета, из цели обучения превращаются в средство. Компьютеры нам нужны для того, чтобы развивать алгоритмический стиль мышления. Обращаю ваше внимание на эту очень важную смену ориентиров: **компьютеры из цели обучения превращаются в средство обучения.** С ними учить интереснее, с ними учить эффективнее. Школьники больше успевают усвоить, если у них есть компьютеры с соответствующим программным обеспечением, но тем не менее это только средство.

Проведем аналогию с математикой. Когда мы решаем математические задачи, их необходимо записывать, то ли с помощью мела и доски, то ли с помощью ручки и бумаги. Но мел и доска, ручка и бумага — это средства записи. Они не имеют отношения к математике как таковой. Если у нас не будет ручки, бумаги, мела, доски

и понадобится, к примеру, каменным топором вырубать формулы на деревянных досках, то, ясное дело, задачи придется выбирать попроще и мы сможем решить меньше задач. И вообще нам начнут нравиться задачи, в которых поменьше надо "вырубать формул". И в школьном курсе мы оставим не самое важное из элементарной математики, а самое удобное для "вырубания". Значит, инструмент должен в каком-то смысле быть более или менее адекватным специфике предмета, иначе он начинает деформировать предмет.

Но тем не менее и ручка, и мел, и топор, да и компьютеры в нашем курсе — все же средства. Это означает, что как нельзя научить решать математические задачи, изучая устройство ручки и процесс изготовления бумаги, так же точно не научишь решать алгоритмические задачи, обучая устройству компьютеров и языкам программирования.

Это чрезвычайно важная вещь, и я хочу, чтобы вы ее поняли. Это ключ ко всему построению курса и учебника.

ЭВМ и языки — средства для проведения курса информатики, подобно ручке и бумаге в математике. Если, скажем, ручка плохо пишет и мы с ней много возимся или, прежде чем написать слово, нам надо ее десять минут заряжать, то мы будем больше времени уделять ручке и меньше решать с ее помощью задач. Точно так же, если наши компьютеры, программные средства и т.д. требуют к себе особого внимания, изучения, если у нас не будет возможности просто пользоваться ими для решения задач, а надо заниматься самими ЭВМ, самими языками, то те задачи, которые мы будем решать, естественно, упростятся — мы не сможем уделить им достаточно времени. Я на этом потом еще остановлюсь, потому что именно этим объясняется появление Робота и всей кажущейся игрушечности в начале курса.

И еще раз скажу вам: ЭВМ и языки — только средства.

Итак, повторю: ЭВМ и языки — только средства, их изучение не дает почти ничего для развития алгоритмического стиля мышления. Может быть, расширяется кругозор, так же как при изучении других достижений техники, но в плане развития структур мышления в человеческой голове не изменяется практически ничего.

Но и средства тоже важны.

С другой стороны, совсем без средств обойтись нельзя. Как в математике мы можем писать ручкой, карандашом или фломастером, но чем-то мы писать должны, выбрать какие-то средства (ЭВМ, язык) мы должны.

И, что очень важно, выбор средств оказывает влияние на сложность решаемых нами задач **и на то, что мы изучаем.** Если мы выберем неадекватные средства: плохую ручку, ненадежную ЭВМ, сложный язык программирования, неудобную программную систему, — то волей-неволей мы будем вынуждены потратить часть времени на ремонт и изучение собственно ЭВМ, собственно языков, собственно ручки, собственно бумаги. И тем самым эта часть времени будет изъята из курса, т.е. пропадет время, которое можно было бы потратить на решение задач, на другие содержательные вещи.

Первый "кит" — выводы.

Вот, пожалуй, и все о первом "ките". Подведем итоги:

1) существует особый, "алгоритмический", стиль мышления;
2) главная цель нашего курса — развитие этого стиля мышления как самостоятельной культурной ценности;
3) ЭВМ и системы программирования в нашем курсе не цели, а всего лишь средства обучения, используемые для развития алгоритмического стиля мышления;
4) выбор этих средств очень важен — удобные средства дают нам возможность продуктивно развивать алгоритмический стиль мышления, тратя минимум времени и сил на технические детали.

А2. Курс должен быть "настоящим"

Второй "кит" — курс должен быть "настоящим". Это означает, что нельзя ради простоты обучения подменять понятия.

Теоремы, доказательства и многие понятия школьной математики, возможно, и сложны для преподавания, но из этого не следует, что их можно заменить какими-нибудь легкими придуманными понятиями, отсутствующими в "настоящей математике" (в науке). Если в иностранном языке какие-то конструкции слишком сложны для преподавания в школе, то их лучше не преподавать вообще, а не заменять придуманными простыми, но не существующими в языке суррогатами.

Точно так же в школьной информатике: мы, конечно, должны упростить, адаптировать набор понятий "настоящей информатики" для школьников, но при этом ни в коем случае нельзя производить подмену понятий. Учить надо настоящему либо — если что-то слишком сложно для школьников — не учить вовсе. Но нельзя учить придуманному, тому, чего в жизни нет.

Я могу привести несколько примеров. В одном из пособий по информатике всерьез изучались такие надуманные "свойства" информации, как ее "своевременность" или "ясность". Эти придуманные (точнее, "бытовые") понятия легко объяснить, о них массу всего можно сочинить, о них легко говорить. Но ничего подобного в "настоящем" курсе информатики быть не должно. "Настоящий" означает, что следует взять науку под названием "информатика" (англ. *Computer science*) и выбрать из нее то, что можно и нужно адаптировать для школьников. В школьном курсе можно ограничиться небольшим набором понятий, но нельзя их подменять легкими для изучения, но "ненастоящими" понятиями, не существующими ни в теоретической, ни в практической информатике.

Новая старая задача

Продолжение. См. с. 3, 4

Язык Паскаль

В Паскале в отличие от школьного алгоритмического языка (а также и Бейсика) массивы являются статическими. Это означает, что при декларации массивов нижние и верхние границы индексов могут быть лишь константами. Правда, здесь имеется специальная техника динамического размещения массивов [1]. Однако для упрощения программ мы будем пользоваться обычными статическими массивами. Максимально возможный размер массивов `a`, `b`, `c` (максимально возможное число дисков на стержне A в исходном состоянии) `Nmax` принят равным 20.

```
{Задача "Ханойские башни"}
uses crt;
const Nmax=20;
type ArrType=array[1..Nmax] of word;
var
  n,na,nb,nc,i:byte; a,b,c:ArrType;
  {Процедура, иллюстрирующая перемещение диска с одного стержня
   на другой}
Procedure PutOneDisk(FromB,ToB:char);
begin write(FromB,'->',ToB,';',' ') end;
  {Процедура перекладывания n дисков в традиционной задаче
   "Ханойские башни"}
  procedure MoveDisks(n:byte; c1,c2,c3:char);
  begin
    if n>0 then begin
    {Перекладываем n-1 дисков с c1 на c2}
    MoveDisks(n-1, c1,c3,c2);
    {Перекладываем один диск с c1 на c3}
    PutOneDisk(c1,c3);
    {Перекладываем n-1 дисков с c2 на c3}
    MoveDisks(n-1, c2,c1,c3);
 end;
end;
{Процедура, изменяющая элементы массивов ArrFrom и ArrTo
 и их количество при перекладывании count дисков }
procedure AlterArrays(count:byte; var k1,k2:byte;
var ArrFrom, ArrTo: ArrType);
Var i:byte;
begin
  for i:=1 to count do ArrTo[k2+i]:=ArrFrom[k1-count+i];
  k1:=k1-count; k2:=k2+count;
end;{AlterArrays}
{Процедура Place подготавливает стержни B и C так, чтобы
 диск на стержне A можно было переместить на один из них,
 после чего перемещает диск }
Procedure Place(var na,nb,nc:byte; var a,b,c:ArrType);
Var
  count,poin,switch:byte;
  placed:boolean;
begin
  if (nc=0) or (a[na]<=c[nc]) then switch:=1
  {диск со стержня A можно надеть на стержень C}
  else
    if (nb=0) or (a[na]<=b[nb]) then switch:=2
    {диск со стержня A можно надеть на стержень B}
    else
      switch:=3; {надо готовить место для него на стержне B или C}
      Case switch of
      1:begin
        PutOneDisk('A','C'); {перекладываем диск со стержня A на C}
        AlterArrays(1,na,nc,a,c) {уточняем массивы a и c}
        end;
     2:begin
        PutOneDisk('A','B'); {перекладываем диск с A на B}
        AlterArrays(1,na,nb,a,b) {уточняем массивы a и b}
       end;
    3:begin{switch=3}
      {Готовим место для него на стержне B или C путем
       перекладывания дисков с B на C и наоборот }
      placed:=false;
  repeat {цикл подготовки места}
   {Перекладываем диски с B на C. Сначала определяем количество
    перекладываемых дисков count}
   count:=0; poin:=nb;
   while (poin>0) and (b[poin]<=c[nc]) do
   begin count:=count+1; poin:=poin-1 end;
      if count>0 then begin
     MoveDisks(count,'B','A','C'); {перекладываем их}
     AlterArrays(count,nb,nc,b,c) {уточняем массивы b и c}
   end;
   if (poin=0) or (a[na]<=b[nb]) then begin
   {для диска на A подготовлено место на B}
   PutOneDisk('A','B'); {перекладываем диск с A на B}
   AlterArrays(1,na,nb,a,b); {уточняем массивы a и b}
   placed:=true;
  end
 else begin {Перекладываем диски с C на B}
  {Сначала определяем количество перекладываемых дисков count}
   count:=0; poin:=nc;
   while (poin>0) and (c[poin]<=b[nb]) do
   begin count:=count+1; poin:=poin-1 end;
      if count>0 then begin
     MoveDisks(count,'C','A','B'); {перекладываем их}
     AlterArrays(count,nc,nb,c,b) {уточняем массивы c и b}
   end;
   if (poin=0) or (a[na]<=c[nc]) then
   begin
     {для диска на A подготовлено место на C}
     PutOneDisk('A','C'); {перекладываем диск с A на C}
     AlterArrays(1,na,nc,a,c); {уточняем массивы a и c}
     placed:=true;
  end
 end; {Перекладываем диски с B на C}
until placed; {цикл подготовки места}
end; {switch=3}
end;{Case switch}
end;{Place}
{Процедура Join объединяет диски стержней B и C на одном из них}
Procedure Join(var nb,nc:byte; var b,c:ArrType);
var
  count,poin1,poin2:byte;
begin
  poin1:=nb; poin2:=nc;
  while (poin1>0) and (poin2>0) do begin {Цикл объединения}
 {Перекладываем диски с B на C}
 {Сначала определяем количество перекладываемых дисков count}
 count:=0; poin1:=nb;
 while (poin1>0) and (b[poin1]<=c[nc]) do
 begin count:=count+1; poin1:=poin1-1 end;
 if count>0 then
 begin
   MoveDisks(count,'B','A','C'); {перекладываем их}
   AlterArrays(count,nb,nc,b,c) {уточняем массивы b и c}
 end;
 if poin1>0 then begin{Перекладываем диски с C на B}
   {Сначала определяем количество перекладываемых дисков count}
   count:=0; poin2:=nc;
   while (poin2>0) and (c[poin2]<=b[nb]) do
   begin count:=count+1; poin2:=poin2-1 end;
 if count>0 then begin
   MoveDisks(count,'C','A','B'); {перекладываем их}
   AlterArrays(count,nc,nb,c,b) {уточняем массивы c и b}
                end;
 end;{Перекладываем диски с C на B}
 end; {Цикл объединения}
end;{Join}
BEGIN
  repeat {Цикл ввода}
  writeln('Введите исходное количество дисков на стержне A ');
  writeln('оно должно быть больше нуля и не превышать ',Nmax);
  readln(n);
  until (n>0) and (n<=Nmax); {Цикл ввода}
  {Заполняем массив случайным образом}
  randomize;
  for i:=1 to n do a[i]:=random(25)+1;
    writeln('Массив, моделирующий стержень A, имеет вид:');
    for i:=1 to n do write(a[i],' ');
      writeln;
      writeln('Чтобы получить решение, нажмите любую клавишу');
    repeat until keypressed;
    na:=n; nb:=0; nc:=0;
    {na, nb, nc - число дисков на стержнях A, B, C}
    writeln('Последовательность перекладывания дисков :');
  repeat
{Перекладываем по одному диску с A на B или C}
Place(na,nb,nc,a,b,c);
until na=0;
{Если не все диски размещены на C, то
 объединяем диски стержней B и C на одном из них}
if nc<>n then Join(nb,nc,b,c);
{Если диски собраны на B, то переносим их на C}
if nc=0 then begin
 MoveDisks(nb,'B','A','C');
 AlterArrays(nb,nb,nc,b,c);
            end;
writeln;
writeln('Теперь массив, моделирующий стержень C, имеет вид:');
for i:=1 to n do write(c[i],' ');
END.
```

Язык Бейсик (вариант QuickBasic [2])

Прежде чем представлять программу, заметим следующее. В примечании к процедуре `Place` на школьном алгоритмическом языке говорилось об использовании в процедуре сложных условий и упоминалось, что в Бейсике всегда происходит полная проверка всех простых условий, входящих в сложное условие. Поэтому на этом языке сложное условие вида:

```
nb% = 0 OR a%(na%) <= b%(nb%)
```

Продолжение на с. 14

Новая старая задача

Продолжение. См. с. 3, 4, 13

— является некорректным, если нижней границей индекса массивов a, b и c является единица. Выход заключается в использовании массивов a, b и c с нижней границей индексов 0.

Хотя в Бейсике можно использовать динамические массивы, но для простоты организуем массивы a, b и c, так же как в Паскале: верхней границей индекса этих массивов будет константа Nmax=20. Элементами массивов a, b и c с индексом 0 пользоваться не будем

```
'Задача "Ханойские башни"
'Объявляем используемые процедуры и величины
DECLARE SUB AlterArrays (count%, k1%, k2%, aFrom%(), aTo%())
DECLARE SUB Join (nb%, nc%, b%(), c%())
DECLARE SUB MoveDisks (n%, c1$, c2$, c3$)
DECLARE SUB Place (na%, nb%, nc%, a%(), b%(), c%())
DECLARE SUB PutOneDisk (fromB$, ToB$)
DIM Nmax AS INTEGER: Nmax = 20
DEFINT I, N
DIM a(0 TO Nmax) AS INTEGER, b(0 TO Nmax) AS INTEGER
DIM c(0 TO Nmax) AS INTEGER
DO
 PRINT "Введите исходное количество дисков на стержне A"
 PRINT "оно должно быть больше нуля и не превышать"; Nmax
 INPUT n
LOOP UNTIL n > 0 AND n<=Nmax
'Заполняем массив a случайным образом
RANDOMIZE (TIMER)
FOR i = 1 TO n: a(i) = INT(25 * RND(1)) + 1: NEXT i
PRINT "Массив, моделирующий стержень A, имеет вид:"
FOR i = 1 TO n: PRINT a(i); : NEXT i
PRINT
PRINT "Чтобы получить решение, нажмите любую клавишу"
DO: LOOP UNTIL INKEY$ <> ""
  na = n: nb = 0: nc = 0
  'na, nb, nc - число дисков на стержнях A, B, C
  PRINT "Последовательность перекладывания дисков :"
  DO
   'Перекладываем по одному диску с A на B или C
   CALL Place(na, nb, nc, a(), b(), c())
   LOOP UNTIL na = 0
  'Если не все диски размещены на стержне C, то
  'объединяем диски стержней B и C на одном из них
  IF nc <> n THEN CALL Join(nb, nc, b(), c())
    'Если диски собраны на B, то переносим их на C
    IF nc = 0 THEN
      CALL MoveDisks(nb, "B", "A", "C")
      CALL AlterArrays(nb, nb, nc, b(), c())
  END IF
  'Печатаем результат
  PRINT
  PRINT "Теперь массив, моделирующий стержень C, имеет вид:"
  FOR i = 1 TO n: PRINT c(i); : NEXT i
END

SUB AlterArrays (count%, k1%, k2%, aFrom%(), aTo%())
'Процедура, изменяющая элементы массивов aFrom и aTo
'и их количество при перекладывании count дисков
DIM i AS INTEGER
FOR i = 1 TO count%
  aTo%(k2% + i) = aFrom%(k1% - count% + i)
NEXT i
k1% = k1% - count%: k2% = k2% + count%
END SUB

SUB Join (nb%, nc%, b%(), c%())
'Процедура Join объединяет диски
'стержней B и C на одном из них
DIM count AS INTEGER, poin1 AS INTEGER, poin2 AS INTEGER
 poin1 = nb%: poin2 = nc%
DO WHILE poin1 > 0 AND poin2 > 0 'Цикл объединения
 'Перекладываем диски с B на C
 'сначала определяем количество
 'перекладываемых дисков count
 count = 0: poin1 = nb%
 DO WHILE poin1 > 0 AND b%(poin1) <= c%(nc%)
   count = count + 1: poin1 = poin1 - 1
   LOOP
     IF count > 0 THEN
      'перекладываем их,
      CALL MoveDisks(count, "B", "A", "C")
      'уточняем массивы b и c
      CALL AlterArrays(count, nb%, nc%, b%(), c%())
     END IF
    'Если на B есть диски, то перекладываем диски с C на B
       IF poin1 > 0 THEN
         'сначала определяем количество
         'перекладываемых дисков count
          count = 0: poin2 = nc%
   DO WHILE poin2 > 0 AND c%(poin2) <= b%(nb%)
   count = count + 1: poin2 = poin2 - 1
   LOOP
    IF count > 0 THEN
     'перекладываем их
     CALL MoveDisks(count, "C", "A", "B")
     'уточняем массивы c и b
     CALL AlterArrays(count, nc%, nb%, c%(), b%())
    END IF
 END IF 'poin1 > 0
LOOP 'Цикл объединения
END SUB

SUB MoveDisks (n%, c1$, c2$, c3$)
'Процедура перекладывания n дисков в
'традиционной задаче "Ханойские башни"
IF n% > 0 THEN
 'Перекладываем n-1 дисков с c1 на c2
 CALL MoveDisks(n% - 1, c1$, c3$, c2$)
 'Перекладываем один диск с c1 на c3
 CALL PutOneDisk(c1$, c3$)
 'Перекладываем n-1 дисков с c2 на c3
 CALL MoveDisks(n% - 1, c2$, c1$, c3$)
END IF
END SUB

SUB Place (na%, nb%, nc%, a%(), b%(), c%())
'Процедура Place подготавливает стержни B и C так, чтобы
'диск на стержне A можно было переместить на один из них,
'и делает это
DIM count AS INTEGER, poin AS INTEGER
DIM switch AS INTEGER, placed AS INTEGER
IF nc% = 0 OR a%(na%) <= c%(nc%) THEN
 switch = 1 'диск со стержня A можно надеть на стержень C
ELSE
 IF nb% = 0 OR a%(na%) <= b%(nb%) THEN
  switch = 2 'диск со стержня A можно надеть на стержень B
 ELSE
  switch = 3 'надо готовить место для него на B или C
 END IF
END IF
SELECT CASE switch
 CASE 1
 CALL PutOneDisk("A", "C") 'перекладываем диск с A на C
 'уточняем массивы a и c
 CALL AlterArrays(1, na%, nc%, a%(), c%())
 CASE 2
 CALL PutOneDisk("A", "B") 'перекладываем диск с A на B
 'уточняем массивы a и b
 CALL AlterArrays(1, na%, nb%, a%(), b%())
 CASE 3
 'Готовим место для него на стержне B или C
 'путем перекладывания дисков с B на C и наоборот
 placed = 0
 DO 'цикл подготовки места
 'Перекладываем диски с B на C
 'сначала определяем количество
 'перекладываемых дисков count
 count = 0: poin = nb%
 DO WHILE poin > 0 AND b%(poin) <= c%(nc%)
 count = count + 1: poin = poin - 1
 LOOP
 IF count > 0 THEN
   CALL MoveDisks(count, "B", "A", "C") 'перекладываем их
   'уточняем массивы b и c
   CALL AlterArrays(count, nb%, nc%, b%(), c%())
 END IF
 IF poin = 0 OR a%(na%) <= b%(nb%) THEN
   'для диска на A подготовлено место на B
   CALL PutOneDisk("A", "B") 'перекладываем диск с A на B
   'уточняем массивы a и b
   CALL AlterArrays(1, na%, nb%, a%(), b%())
   placed = 1
 ELSE 'Перекладываем диски с C на B
   'сначала определяем количество
   'перекладываемых дисков count
   count = 0: poin = nc%
   DO WHILE poin > 0 AND c%(poin) <= b%(nb%)
   count = count + 1: poin = poin - 1
   LOOP
   IF count > 0 THEN
    CALL MoveDisks(count, "C", "A", "B") 'перекладываем их
    'уточняем массивы c и b
    CALL AlterArrays(count, nc%, nb%, c%(), b%())
   END IF
   IF poin = 0 OR a%(na%) <= c%(nc%) THEN
    'Для диска на A подготовлено место на C
    CALL PutOneDisk("A", "C") 'перекладываем диск с A на C
    'уточняем массивы a и c
    CALL AlterArrays(1, na%, nc%, a%(), c%())
    placed = 1
   END IF
 END IF
 LOOP UNTIL placed = 1 'цикл подготовки места
END SELECT
END SUB

SUB PutOneDisk (fromB$, ToB$)
'Процедура, иллюстрирующая перемещение
'диска с одного стержня на другой
 PRINT fromB$; "->"; ToB$; ";"; " ";
END SUB
```

Язык Си

В Си (так же как и в Паскале) массивы являются статическими. Это означает, что при декларации массива его размер может быть лишь константой. Разумеется, и здесь есть специальная техника динамического размещения массивов [3], но для упрощения программ мы будем пользоваться обычными статическими массивами. Максимально возможный размер массивов `a, b, c` `Nmax` принят равным 20. Напомним, что в Си индексация массивов всегда начинается с нуля. Чтобы сохранить единый подход, элементами массивов `a, b` и `c` с индексом `0` пользоваться не будем. Во избежание недоразумений подчеркнем, что, в отличие от Бейсика, это никак не связано с проблемой вычисления сложных условий, которая в Си решается оптимально. Заметим, что максимальное число дисков на стержне **A** при отказе от нулевых элементов составляет `Nmax-1`.

```
/* Задача "Ханойские башни" */
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<conio.h>
#define Nmax 20
int a[Nmax],b[Nmax],c[Nmax];
void PutOneDisk(char FromB,char ToB)
/* Процедура, иллюстрирующая перемещение
 диска с одного стержня на другой */
{printf("%c->%c; ",FromB,ToB);}
void MoveDisks(int n,char c1,char c2,char c3)
/* Процедура перекладывания n дисков в
 традиционной задаче "Ханойские башни" */
{if (n>0)
 {/*Перекладываем n-1 дисков с c1 на c2*/
 MoveDisks(n-1, c1,c3,c2);
 /*Перекладываем один диск с c1 на c3*/
 PutOneDisk(c1,c3);
 /*Перекладываем n-1 дисков с c2 на c3*/
 MoveDisks(n-1, c2,c1,c3);
 }
}
void AlterArrays(int count, int *k1,int *k2,
 int ArrFrom[],int ArrTo[])
/* Процедура, изменяющая элементы массивов ArrFrom и ArrTo
 и их количество при перекладывании count дисков */
{int i;
 for (i=1; i<=count; i++) ArrTo[*k2+i]=ArrFrom[*k1-count+i];
 *k1-=count; *k2+=count;
}
void Place(int *na,int *nb,int *nc,int a[],int b[],int c[])
/* Процедура Place подготавливает стержни B и C так, чтобы диск на
  стержне A можно было переместить на один из них, и делает это */
{int count,poin,var,placed=0;
 if (*nc == 0 || a[*na]<=c[*nc])
 /*диск с A можно надеть на C*/ var=1; else
 if (*nb == 0 || a[*na]<=b[*nb])
 /*диск с A можно надеть на B*/ var=2;
 else /* надо готовить место для него на B или C */ var=3;
 switch (var)
 {case 1:
 PutOneDisk('A','C');/*перекладываем диск с A на C*/
 AlterArrays(1,na,nc,a,c);/*уточняем массивы a и c*/
 break;
 case 2:
 PutOneDisk('A','B');/*перекладываем диск с A на B*/
 AlterArrays(1,na,nb,a,b);/*уточняем массивы a и b*/
 break;
 case 3:
 /* Готовим место для него на стержне B или C путем
 перекладывания дисков с B на C и наоборот */
 do /* цикл подготовки места*/
 /*Перекладываем диски с B на C*/
 /*сначала определяем количество перекладываемых дисков count*/
 {count=0; poin=*nb;
 while (poin>0 && b[poin]<=c[*nc])
 {count++; poin--;}
 if (count>0)
 {MoveDisks(count,'B','A','C');/*перекладываем их*/
 /*уточняем массивы b и c*/
     AlterArrays(count,nb,nc,b,c);
 }
 if ( poin == 0 || a[*na]<=b[*nb] )
 /*для диска на A подготовлено место на B*/
 {PutOneDisk('A','B');/*перекладываем диск с A на B*/
     AlterArrays(1,na,nb,a,b);/*уточняем массивы a и b*/
 placed=1;
 }
 else /*Перекладываем диски с C на B*/
 /*сначала определяем количество перекладываемых дисков count*/
 {count=0; poin=*nc;
 while (poin>0 && c[poin]<=b[*nb])
 {count++; poin--;}
 if (count>0)
 {MoveDisks(count,'C','A','B');/*перекладываем их*/
 /*уточняем массивы c и b*/
      AlterArrays(count,nc,nb,c,b);
 }
 if ( poin == 0 || a[*na]<=c[*nc] )
 /*для диска на A подготовлено место на C*/
 {PutOneDisk('A','C');/*перекладываем диск с A на C*/
      AlterArrays(1,na,nc,a,c);/*уточняем массивы a и c*/
 placed=1;
 }
 } /*Перекладываем диски с C на B*/
 }
 while (!placed);/* цикл подготовки места*/
 } /*switch*/
}
void Join(int *nb, int *nc,int b[],int c[])
/* Процедура Join объединяет диски
 стержней B и C на одном из них */
{int count,poin1,poin2;
 poin1=*nb; poin2=*nc;
 while (poin1>0 && poin2>0)/*Цикл объединения*/
 {/*Перекладываем диски с B на C*/
 /*сначала определяем количество перекладываемых дисков count*/
 count=0; poin1=*nb;
 while (poin1>0 && b[poin1]<=c[*nc])
 {count++; poin1--;}
 if (count>0)
 {MoveDisks(count,'B','A','C');/*перекладываем их*/
 AlterArrays(count,nb,nc,b,c);/*уточняем массивы b и c*/
 }
 if (poin1>0) /*Перекладываем диски с C на B*/
 {/*сначала определяем количество перекладываемых дисков count*/
 count=0; poin2=*nc;
 while (poin2>0 && c[poin2]<=b[*nb])
 {count++; poin2--;}
 if (count>0)
 {MoveDisks(count,'C','A','B');/*перекладываем их*/
 AlterArrays(count,nc,nb,c,b);/*уточняем массивы c и b*/
 }
 } /*Перекладываем диски с C на B*/
 } /*Цикл объединения*/
}
void main()
{int n,na,nb,nc,i;
 do
 {printf
 ("\n Введите исходное количество дисков на стержне A ");
 printf("\n Оно должно быть больше нуля и меньше Nmax ");
 scanf("%d",&n);
   }
 while (n<=0 || n>=Nmax);
/*Заполняем массив случайным образом*/
 randomize();
 for (i=1; i<=n; i++)
 a[i]=random(25)+1;
 printf("\n Массив, моделирующий стержень A, имеет вид:\n");
 for (i=1; i<=n; i++) printf("%d ",a[i]);
 printf("\n Чтобы получить решение, нажмите любую клавишу");
 getch();
 na=n; nb=0; nc=0;
 /*na, nb, nc - число дисков на стержнях A, B, C*/
 printf("\n Последовательность перекладывания дисков :\n");
 do
 {/*Перекладываем по одному диску с A на B или C*/
 Place(&na,&nb,&nc,a,b,c);
 }
 while (na>0);
 /*Если не все диски размещены на C, то
 объединяем диски стержней B и C на одном из них*/
 if (nc != n) Join(&nb,&nc,b,c);
 /*Если диски собраны на B, то переносим их на C*/
 if (nc == 0)
 {MoveDisks(nb,'B','A','C');
 AlterArrays(nb,&nb,&nc,b,c);
 }
 printf
 ("\n Теперь массив, моделирующий стержень C, имеет вид:\n");
 for (i=1; i<=n; i++) printf("%d ",c[i]);
}
```

Приведем пример вывода на экран результатов выполнения какой-либо из этих программ:

```
Массив, описывающий стержень A, имеет вид:
            10 21 22 8
Чтобы получить решение, нажмите любую клавишу
Последовательность перекладывания дисков:
A->C; A->B; A->B; A->B; C->B; B->A; B->C; A->C; B->A; C->B;
C->A; B->A; B->C; A->C; A->B; C->B; A->C; B->A; B->C; A->C;

Теперь массив, описывающий стержень C, имеет вид:
            22 21 10 8
```

Интересным является вопрос о том, когда перекладывание дисков происходит быстрее — при произвольном порядке расположения дисков в исходном состоянии или в случае классической задачи “Ханойские башни”. Как показали расчеты, при количестве дисков n, равном 4 и 5, число перекладываний в первом случае больше (в среднем соответственно на 10 и 6%, а при $n > 5$ — меньше, причем с увеличением n экономия времени перекладывания увеличивается. Так, при $n=6$ экономия в среднем составляет 5%, при $n = 8$ — 18%, при $n=10$ — 26%.

ЛИТЕРАТУРА

1. *Кушниренко А.Г., Лебедев Г.В., Сворень Р.А.* Основы информатики и вычислительной техники. М.: Просвещение, 1990.
2. *Златопольский Д.М.* Рекурсия. “Информатика”, 1996, № 7—8.
3. *Эпиктетов М.Г.* Почему школьный алгоритмический? “Информатика”, 1995, № 24.
4. *Вьюкова Н.И., Галатенко В.А., Ходулев А.Б.* Систематический подход к программированию. М.: Наука, 1988.
5. *Фаронов В.В.* Программирование на персональных ЭВМ в среде Турбо Паскаль. М.: Изд-во МГТУ, 1992.
6. *Зельднер Г.А.* QuickBasic 4.5. М.: ABF, 1994.
7. *Болски М.И.* Язык программирования Си. Справочник. М.: Радио и связь, 1988.

Программа на языке Си подготовлена редакцией.

Чему учить на уроках информатики

Круглый стол

Окончание. См. с. 1, 2

действие через экспансию, взаимодействие через уточнение и размежевание предметных областей, взаимодействие через интеграцию.

Экспансия. Суть сценария — превратить курс информатики в сквозной обязательный курс с первого по одиннадцатый класс, включив в него все общезначимые образовательные составляющие и поддерживая с помощью этого курса преподавание других учебных предметов (включая русский и иностранный языки, математику, различные дисциплины естественно-научного цикла). В этом сценарии информатике отводится роль интегрирующей дисциплины.

Размежевание. Освободить курс информатики от несвойственных ему составляющих, передав все специальные вопросы, не относящиеся к "ядру" этой дисциплины, в смежные учебные предметы (например, в технологию). Оставшееся ядро и будет составлять действительное содержание курса в обозримом будущем.

Интеграция. Включиться в движение за интеграцию учебных дисциплин, начать разработку модульных курсов с "информатическими составляющими", стимулировать коллективное педагогическое действие, направленное на постепенное создания "гибких учебных планов" в школе. "Информатическая" составляющая могла бы стать одной из основ такой интеграции (например, интеграция вокруг используемых НИТ).

Каждый из этих сценариев имеет свои притягательные стороны, у каждого из них есть шансы воплотиться в реальность. Но шансы эти различны. Осознанно или неосознанно каждый из нас (преподавателей, методистов и исследователей в области "педагогической информатики") способствует претворению в жизнь одной из этих базовых схем.

По первому сценарию информатика должна взять на себя роль "метадисциплины". На воплощение этого сценария работают, в частности, те, кто пытается всемерно расширить ее мировоззренческий и воспитательный аспекты. Например, авторы одного из учебников информатики для неполной средней школы, который рассматривался на Федеральном экспертном совете осенью этого года, настаивают на необходимости освоения школьниками в курсе информатики не столько техники работы с компьютерами, сколько "базовых, фундаментальных идей, вошедших в общенаучный и инженерный обиход в связи с развитием и распространением идей информатики". При этом главной задачей курса мыслится "формирование у школьников соответствующих структур мышления". Заметим, что речь идет не только (и не столько) об уже знакомом "алгоритмическом мышлении", сколько о развитии у школьников системного аналитического подхода, формировании умственных приемов, необходимых для постановки задач, создании (и исследованиях) моделей процессов и явлений. Авторы другого рекомендованного к использованию учебника информатики ставят задачу обучения школьников методам компьютерного моделирования. Очевидно, что все это важные и благородные задачи. Но ясно и другое: это общие задачи всего курса школьного обучения, а не только предмета "информатика". К этой позиции наверняка присоединятся методисты по математике (в неполной средней школе они вводят понятие "переменная", без которой не может обойтись и курс информатики), и методисты естественно-научных дисциплин (любая научная теория включает в себя как одну из основных частей построение и использование математических моделей). Известно, как сложна эта педагогическая задача, как нелегко добиться переноса соответствующих умственных действий на новые области (даже если эти действия сформированы). Конечно, эту задачу нельзя решать в рамках незначительного по объему (36 учебных часов) курса информатики. И, хотя авторы учебников ничего не говорят о межпредметных связях, следующий шаг очевиден: установить эти связи и перенести изучение соответствующего материала на уроки информатики (забрав, например, часы из предметной области "естествознание"). Ясно, что на этом пути авторов учебников по информатике ждут жаркие дебаты с авторами учебников по другим общеобразовательным дисциплинам. Процедуры и возможные результаты решения задачи "справедливый раздел пирога учебного времени между учебными предметами" хорошо известны из практики развития советской школы.

Первый из выделенных базовых сценариев можно рассматривать как "экспансионистский". В противовес этому второй сценарий можно назвать "консервативным". Сегодня информатика признана как учебный предмет, и повсеместное исчезновение из школы угрожает ей не больше, чем физике или иностранному языку. Вместе с тем объем средств, выделяемых на общее образование, постоянно сокращается. Сокращается и время, отводимое на отдельные учебные дисциплины. Нужно ответить на вопрос: где тот минимум, далее которого информатика исчезает как самостоятельная учебная дисциплина, что должно войти в минимальное "ядро" учебного предмета? Авторы некоторых популярных современных учебников пытаются включить в это ядро "пользовательский" курс, фактически вынося за скобки сложную и трудоемкую часть, связанную с освоением программирования. Другие стараются сохранить ориентацию на развитие "алгоритмического мышления". Действительно, вопрос о "ядре" — традиционный "вечный вопрос" для всякого учебного предмета. Ответ на него фиксирует ценностную ориентацию педагогического сообщества. Мы уже были свидетелями перехода от массового изучения Бейсика к освоению Лексикона. Сможет ли общеобразовательный курс информатики действительно обеспечить формирование у школьников алгоритмического мышления, будут ли российские школьники продолжать одерживать победы на международных олимпиадах — об этом мы скоро узнаем.

Первый и второй сценарии построены по схеме "соперничество за ресурсы" между представителями отдельных учебных предметов. Третий базовый сценарий основан на схеме сотрудничества. Действительно, интеграция учебных дисциплин возможна лишь на "добровольной и взаимовыгодной основе". Здесь все строится исходя из общих для всех интересов отдельного ученика, а не конфликтующих между собой интересов отдельных профессиональных групп педагогов. На претворение этого сценария в жизнь работают, в частности, те, кто пытается выделять и разрабатывать отдельные модули по информатике, опробует формы совместного обучения информатике и другим общеобразовательным дисциплинам. Все работы по установлению и углублению межпредметных связей информатики также способствуют развитию событий по третьему сценарию. Этому способствует и расширяющееся использование компьютеров в ходе преподавания различных общеобразовательных дисциплин, и создание "компьютерной информационной учебной среды" в школе, и распространение Интернет. Развитию по этому сценарию также способствуют мероприятия по демократизации школы, претворение в жизнь принципов педагогики сотрудничества. На этот сценарий работает и "общепедагогическая" составляющая курса информатики, которая упоминалась выше. На этот сценарий работает и создание интегрирующих учебных курсов, наподобие "Компьютерного дела"[3]. В таком курсе содержание учебной работы, взятое из различных предметных областей (естествознание, родной язык), объединяется со средствами работы с информацией, изучаемыми на уроках информатики.

Чему же сегодня учить на уроках информатики? Автор далек от того, чтобы призывать к немедленным переменам. Процессы, разворачивающиеся в системе образования, достаточно инерционны. Каждый учитель с неизбежностью "воспроизводит себя" и, решая текущие задачи, делает то, что лучше умеет. Вопрос в том, на какие изменения, на какое будущее ориентировать преподавателей информатики, куда направлять переподготовку учителей. Сегодня самые распространенные курсы повышения квалификации ориентируются на освоение работы в среде Windows 95 и Microsoft Office. Авторы и преподаватели этих курсов обычно замалчивают, что эту технологическую задачу очень успешно и эффективно решают, например, обучающие программы, входящие в комплект "Самоучителя по Windows 95 и Microsoft Office[4]". Этот учебный программный продукт позволяет любому быстро приобрести "компьютерную компетентность", достаточную для практической работы с компьютером в режиме самообучения. Такой продукт потенциально способен вывести обучение работе с базовыми программными средствами из числа основных задач учителя информатики (или качественно упростить ее решение). Пора менять технологическую ориентацию подобных курсов, помочь преподавателю информатики стать не только и не столько помощником тех, кто осваивает работу с компьютером, а и носителем инновационных процессов в школе.

* * *

На что же ориентироваться преподавателю информатики сегодня? Автор этих заметок пытается внести свой вклад в реализацию третьего из приведенных здесь сценариев. Присоединяйтесь!

---

[3] Подробнее об этом курсе см. в журнале "Информатика и образование", № 4 и 6, 1996.

[4] Подробную информацию об этом пакете можно найти в Интернет (**www.aha.ru/~uniar**).

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

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

Тираж 9000 экз.
Заказ №

121165,
Киевская, 24
тел. 249 4896
Отдел рекламы
тел. 240 1041

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков 32291
для предприятий и организаций 32591

Internet: infosef@glasnet.ru
Fidonet: 2:5020/69.32
WWW: http://www.glasnet.ru/~infosef
FTP: ftp://ftp.glasnet.ru/users/infosef

Москва.