№ 38 (327) 8–15 октября 2001

ИНФОРМАТИК

Еженедельная газета Объединения педагогических изданий «ПЕРВОЕ СЕНТЯБРЯ»
ПОДПИСКА: (095) 249-47-58

От ARPAnet к Internet

35 лет назад, в 1966 году, появился прототип глобальной компьютерной сети Интернет

См. с. 32

Читайте в номере

Информация 2–3

Что "Информатика" предложит подписчикам в 2001/2002 учебном году

Страницы повышения квалификации 4–7

Е.В. Андреева. Олимпиады по информатике. Пути к вершине

"...как школьные уроки физкультуры не могут помочь в побитии спортивных рекордов, так и уроки информатики вовсе не должны давать абсолютно все знания и умения, необходимые для учеников официальных олимпиад по информатике". Приглашаем в лекторий. Сегодня мы попытаемся узнать секрет успешных выступлений школьников на олимпиадах.

История информатики 8

Д.М. Златопольский. Последователи Джона Непера

"Некий шотландский барон, имени которого я не запомнил, выступил с блестящим достижением: он каждую задачу на умножение и деление превращает в чистое сложение и вычитание..." — так писал один из создателей небесной механики Иоганн Кеплер об изобретателе логарифмов Джоне Непере, известном, однако, не только этим изобретением.

Внеклассная работа 25–26

Д.М. Златопольский. Кроссворд с фрагментами

Предложите ребятам решить кроссворд, в котором слова зашифрованы рисунками.

Материалы к уроку 27–30

В.П. Гладков, А.П. Шестаков. Вопросы, задания и контрольные работы для начинающих программистов

Из каких частей состоит программа на Паскале? Какие значения имеют переменные в начале ее выполнения? Для чего описывают процедуры и функции? Каким образом можно вводить значения логических переменных? Продолжение публикации, цель которой — помочь начинающему программисту освоить Паскаль. По мнению ее авторов, главным условием успеха тут является большое желание изучить этот язык программирования.

На стенд в кабинете информатики 31

Почти первый

Короткий рассказ о том, как создавался "первый успешно функционирующий электронный цифровой компьютер".

Тематический выпуск 9–24

А.Г. Гейн. Обязательный минимум содержания образования по информатике: и в нем нам хочется дойти до самой сути

Продолжаем публикацию работы, где, как вы уже, наверное, знаете, главы соответствуют разделам обязательного минимума, а параграфы — отдельным вопросам в этих разделах.

Сегодня речь идет в основном об алгоритмах и исполнителях. Кстати, "алгоритм — базовое понятие информатики и, как практически всякое базовое понятие, не имеет строгого определения". А какие свойства алгоритмов и какие способы их записи вы можете назвать?

Постоянные рубрики

"Уроки"
"Материалы к уроку"
"Задачи"
"Олимпиады"
"Уроки для учителя"
"Круглый стол"
"Семинар"
"Мнения"
"Методика"
"Книжный шкаф"
"История науки"
"Информатика в лицах"
"Внеклассная работа"
"История информатики"
"Информатика в начальной школе"
"Предлагаю коллегам"
"Не только информатика"
"Теоретические основы информатики"
"Как это делаю я"
"Проекты"
"На стенд"
"Калейдоскоп"
"Беседы"
"Узелки на память"

ЧТО *"Информатика" предложит подписчикам в 2001/2002 учебном году*

Во-первых, революций не будет ☺. За шесть лет у газеты появились традиции, постоянные рубрики, свой стиль и свое лицо. Мы дорожим всем, что было сделано в эти годы, и не считаем, что пришло время что-либо кардинально менять. Но жизнь не стоит на месте, у редакции и, что очень существенно, у наших читателей появляются новые интересные идеи, которые мы будем реализовывать.

Одна новая/старая идея — подготовка тематических выпусков. Как и в прошлом году, мы предложим подписчикам **две серии специальных тематических выпусков — зимнюю, январскую, и летний трехмесячный марафон, состоящий из 12 номеров**. Летняя серия получит традиционное название "Жаркое лето-2002", а серия январских номеров будет называться "Снег идет". (Напомним, что серия январских номеров 2001 года называлась "Информатика нового века".) В следующем номере мы начнем знакомить вас с материалами, которые будут опубликованы в зимних тематических выпусках. Это — новая книга А.А. Дуванова "Азы информатики", специальный номер из серии "На стенд в кабинете информатики" "Информатика в лицах",

Ф. СП-1

Министерство связи Российской Федерации "Роспечать"

АБОНЕМЕНТ на газету **32291** (индекс издания)

ИНФОРМАТИКА

наименование издания	Количество комплектов	

на 20__ год по месяцам

1	2	3	4	5	6	7	8	9	10	11	12

Куда _____ (почтовый индекс) _____ (адрес)

Кому _____ (фамилия, инициалы)

ДОСТАВОЧНАЯ КАРТОЧКА

ПВ	место	литер

на газету **32291** (индекс издания)

ИНФОРМАТИКА (наименование издания)

Стоимость	подписки	_____ руб.	Количество комплектов	
	переадресовки	_____ руб.		

на 20__ по месяцам

1	2	3	4	5	6	7	8	9	10	11	12

Куда _____ (почтовый индекс) _____ (адрес)

Кому _____ (фамилия, инициалы)

продолжение книги "Зимние вечера. Информатика для начинающих". Первый анонс летних номеров по традиции появится в № 8/2002.

Новый проект, который стартовал в октябре (№ 37/2001), связан с нашей новой рубрикой **"Страницы повышения квалификации"**. Учителю информатики как никому другому необходимо постоянно поддерживать свою "форму". Но далеко не всегда есть возможность пройти *хорошие* курсы повышения квалификации. Мы не случайно построили последнее предложение именно так. Во-первых, у некоторых подписчиков нашей газеты, и их не так мало, возможность пройти курсы повышения квалификации отсутствует в принципе вследствие отсутствия таковых в радиусе нескольких сотен километров вокруг. Во-вторых, не все, у кого такие курсы находятся в пределах досягаемости, имеют на них время, ведь нагрузка учителя порой очень велика. В-третьих, не всех и устраивает уровень курсов, которые им предлагают институты усовершенствования. Некоторые (и иногда справедливо) полагают, что тратить время на такие курсы не стоит. В новом учебном году "Информатика" предложит подписчикам несколько лекционных курсов для повышения квалификации, которые будут прочитаны на страницах газеты. Мы гарантируем высочайший уровень этих материалов и надеемся, что они будут не просто интересны, но очень полезны нашим читателям.

План публикаций лекций курса "Современные педагогические технологии и частные методики обучения информатике" на "страницах повышения квалификации".

Лекции читает И.Н. Фалина

Номер лекции	Номер газеты
1	37/2001
2	39/2001
3	41/2001
4	43/2001
5	45/2001
6	47/2001
7	5/2002
8	7/2002
9	9/2002
10	11/2002
11	13/2002
12	15/2002

Уважаемые читатели, пожалуйста, обратите внимание на то, что лекции 7—12 будут опубликованы в первом полугодии 2002 г.

План публикаций лекций курса "Олимпиады по информатике. Пути к вершине" на "страницах повышения квалификации".

Лекции читает Е.В. Андреева

Номер лекции	Номер газеты
1	38/2001
2	40/2001
3	42/2001
4	44/2001
5	46/2001
6	48/2001
7	6/2002
8	8/2002
9	10/2002
10	12/2002
11	14/2002
12	16/2002

Уважаемые читатели, пожалуйста, обратите внимание на то, что лекции 8—12 будут опубликованы в первом полугодии 2002 г.

План публикаций лекций курса "Введение в специальность "Учитель информатики" на "страницах повышения квалификации".

Лекции читает А.Г. Гейн

Номер лекции	Номер газеты
1	6/2002
2	8/2002
3	10/2002
4	12/2002
5	14/2002
6	16/2002

Уважаемые читатели, пожалуйста, обратите внимание на то, что лекции 1—6 будут опубликованы в первом полугодии 2002 г.

Уважаемые читатели! Газета "Информатика" распространяется только по подписке, поэтому не забудьте подписаться на нашу газету.

И это не все!

В первом полугодии 2002 года начнется публикация материалов, которые все мы давно ждем. Это — **задачник по Excel**, который готовится специально для "Информатики".

Но и это не все!

Еще одна **новая рубрика**, которая появится во втором полугодии, — **"Начните с простого"**. Статьи этой рубрики, написанные простым языком, ориентированы на тех, кто начинает знакомство с вопросом "с нуля". В них вы познакомитесь с множеством интересных и якобы сложных (ведь дело в том, как объяснить!) вопросов: Perl, PHP, Visual Basic, Delphi, SQL и многими другими.

Но и это еще не все!

Мы все равно не сможем рассказать обо всех полезных и интересных материалах, которые будут опубликованы в "Информатике" в 2002 году.

Напоминаем, что наша газета распространяется только по подписке. Розничный тираж, который можно приобрести в редакции, пренебрежимо мал и "разлетается" через месяц (максимум — два) после выхода номера ☹.

Олимпиады по информатике. Пути к вершине

Лекции читает Е.В. Андреева

Что представляют собой современные (официальные) олимпиады по информатике? Каковы правила их проведения, что собой представляет набор задач и как осуществляется их проверка?

Как правильно произвести отбор участников для Всероссийской олимпиады по информатике (данные рекомендации можно применить и для областных, городских и т.д. олимпиад)?

Нужна ли подготовка к олимпиаде по информатике и в чем она может заключаться? Какие алгоритмы желательно знать участнику олимпиады и какую литературу полезно использовать при подготовке к ней? Какой уровень программирования необходим для успешного выступления на Всероссийской олимпиаде? И, наконец, как правильно вести себя школьнику во время практического тура олимпиады по информатике, для того чтобы достичь максимально возможного для данного школьника результата?

Здесь приведены лишь вопросы. Ответы будут даны в лекционном курсе Е.В. Андреевой.

Лекция 1*

В настоящее время в России проводится несколько различных видов олимпиад, в названии которых так или иначе упоминается информатика. Поэтому прежде всего давайте определимся, о каких именно олимпиадах по информатике будет идти речь в данном курсе.

В марте 2001 года в г. Екатеринбурге состоялся заключительный этап XIII Всероссийской олимпиады по информатике (см.: "Информатика", № 18, 19/2001). Данная олимпиада проводится по приказу Министерства образования РФ, согласно "Положению о проведении заключительного этапа Всероссийской олимпиады школьников". Проведению заключительного этапа предшествуют олимпиады по информатике в различных регионах России: республиках, краях, областях, автономных округах и т.д., организованные соответствующими местными органами образования. В региональных олимпиадах участвуют, в свою очередь, победители городских и районных олимпиад по информатике. Участники районных олимпиад отбираются непосредственно в школах. Таким образом, в олимпиаде принимает участие большое число школьников из всех регионов России.

С 1996 года в Санкт-Петербурге на базе Дворца творчества юных при непосредственном организационном участии Института точной механики и оптики проводятся полуфинальные соревнования чемпионата мира по программированию среди студенческих команд вузов. А с 1998 года эти соревнования сопровождаются так называемым ACM junior contest для школьных команд (см.: "Информатика", № 12/2000). В 2000 году эти увлекательные соревнования для школьников получили согласно приказу Министерства образования РФ статус Всероссийской командной олимпиады по программированию (см.: "Информатика", № 12/2001, и сайт **http://neerc.ifmo.ru/school**).

Кроме того, на региональном уровне или в сети Интернет проводятся различные олимпиады, фестивали и конкурсы, связанные с информатикой. Некоторые из этих соревнований имеют довольно широкое представительство и приглашают участников не только из различных регионов России, но и из-за рубежа. Правила подобных соревнований определяются их организаторами и могут быть совершенно различными.

В рамках данного курса мы будем в основном рассматривать вопросы, связанные с первой из упомянутых олимпиад, а также отчасти с командной олимпиадой, так как задачи последней во многом пересекаются с задачами личных олимпиад по информатике, хотя разница в подготовке к олимпиаде именно команды школьников, несомненно, существует. В последнем случае приходится учитывать организационные и психологические особенности участия в таких соревнованиях. Объясняется такой выбор тем, что именно Всероссийская олимпиада по информатике является официальной олимпиадой, по результатам которой после проведения учебно-тренировочных сборов производится формирование сборной России для участия в Международной олимпиаде по информатике. То есть именно эти соревнования представляют собой вершину айсберга олимпиадной информатики и правила их проведения, по-видимому, в ближайшие годы кардинальным образом меняться не будут. Хотя изменения в некоторых условиях Международной олимпиады, в частности, связанные с использованием современных сред программирования (см.: "Информатика", № 37/2001), могут в недалеком будущем коснуться и участников российских олимпиад.

Однако при встречах с учителями информатики членам жюри Всероссийской олимпиады и представителям органов образования неоднократно приходилось выслушивать критические замечания по поводу правил проведения именно этих олимпиад, и в первую очередь по поводу предлагаемых на этих олимпиадах задач, которые в значительной степени определяют уровень задач республиканских, краевых, областных, городских и рай-

* Вторая лекция будет прочитана на страницах № 40.

онных олимпиад. По результатам последних органы образования нередко оценивают работу конкретного учителя информатики, что, конечно же, не является корректным, так как программа школьного курса информатики не может охватить все темы, изучение которых могло бы улучшить результаты выступления школьников на олимпиадах. Тем не менее ответ на подобные замечания очевиден: выбор задач для Всероссийской олимпиады по информатике опирается на опыт и традиции Международной олимпиады, а, в свою очередь, подготовка задач для региональных и местных олимпиад осуществляется в большинстве случаев с оглядкой на Всероссийскую олимпиаду. То есть, как и в спорте, национальные соревнования проводятся по правилам соответствующих международных.

Достигнуть вершин, то есть получить медаль на Международной олимпиаде по информатике, могут лишь единицы. Причем как школьные уроки физкультуры не могут помочь в побитии спортивных рекордов, так и уроки информатики вовсе не должны давать абсолютно все знания и умения, необходимые для участников официальных олимпиад по информатике. Впрочем, подобное замечание можно было бы сделать и по поводу остальных предметных олимпиад школьников, в частности, всероссийских олимпиад по математике. Решение предлагаемых там задач зачастую просто непонятно рядовому учителю математики, хотя темы задач вроде бы имеют отношение к школьной программе. Темы задач по информатике действительно не всегда соответствуют "Обязательному минимуму изучения информатики". Более того, в качестве решения этих задач на олимпиаде требуется предъявить отлаженные программы, написанные на языке программирования высокого уровня, а не описания алгоритмов. Но и это требование полностью соответствует международным правилам. Изучение же программирования в российских школах постепенно вытесняется изучением информационных технологий, и, как ни странно, связано это в том числе и с улучшением качества парка компьютеров в школах.

Однако в нашей стране существует достаточное количество физико-математических школ, а также гимназий и лицеев, изучение информатики в которых ведется углубленно, а программирование является неотъемлемой частью курса. Следовательно, ученики именно таких школ могут быть потенциальными призерами региональных и Всероссийской олимпиад по информатике. Оценивать же работу учителей остальных школ согласно результатам их учеников в этих олимпиадах, как уже было сказано выше, неправильно. Этим целям могли бы служить другие конкурсы и соревнования, проводимые по другим правилам и с другим кругом задач. Ведь спортивные игры с мячом проводят по разным правилам, то же самое может относиться и к современным компьютерным технологиям. Таким образом, Всероссийская олимпиада по информатике сама по себе не может и не должна отслеживать все изменения в программе изучения информатики в российских школах.

Целью проведения Всероссийской олимпиады служит, конечно же, не дальнейшая победа российских школьников в олимпиаде международной. Олимпиады предназначены выявлять наиболее одаренных в области информатики школьников, развивать их способности, повышать интерес к предмету. Они дают возможность школьникам получить раннюю профориентацию, что способствует становлению в дальнейшем российских специалистов в области информатики, вычислительной техники и программирования. Рассмотрим теперь с этой точки зрения правила отбора участников на Всероссийскую олимпиаду по информатике, а также различные возможные подходы к этому отбору в регионах.

Согласно Положению о проведении заключительного этапа Всероссийской олимпиады по информатике, каждый регион (республика, край, область или автономный округ) может выставить всего одного участника. Исключение составляют лишь Москва, Санкт-Петербург и Московская область. Такое ограничение объясняется тем, что в противном случае число компьютеров, необходимых для проведения олимпиады, превысит 200. Кроме того, следует учесть, что по своим техническим характеристикам предоставляемые участникам компьютеры должны различаться незначительно. К сожалению, пока ни один из оргкомитетов проведения олимпиады подобных технических условий предоставить не мог. И как следствие такого правила представительства — итоги олимпиады подводятся в общем зачете, вне зависимости от класса, в котором обучаются участники в школе, хотя среди них встречаются ученики 9-, 8-х, а иногда и более младших классов. Однако квота региона может быть увеличена по результатам выступления представителей этого региона на олимпиаде предшествующего учебного года. Условия привлечения дополнительных участников не всегда одинаковы и зависят в основном опять же от возможностей оргкомитета. Например, в 2001 году на олимпиаду приглашались все обладатели дипломов олимпиады 2000 года, если в 2000 году они обучались в невыпускном классе (это так называемые персональные приглашения), а за каждого ученика выпускного класса, получившего в 2000 году 1-й диплом, регион получал дополнительное место. В последнем случае жюри рекомендует региону использовать дополнительные места в квоте для привлечения к участию в олимпиаде учеников 9—10-х классов.

Казалось бы, успехи или неудачи в выступлениях на Всероссийской олимпиаде представителей различных регионов объясняются уровнем преподавания информатики в регионе, оснащенностью школ современными компьютерами, а также компьютерной грамотностью и доступом к компьютерной технике населения вообще. Однако это далеко не всегда так. Например, московские школьники в целом выступают на олимпиадах очень неровно, несмотря на выдающиеся успехи отдельных учеников. Поэтому попробуем разобраться в "секрете" успешных выступлений на олимпиадах, ключ к которым лежит не только в формах работы с одаренными школьниками, но и в принципах поиска на региональном уровне способных ребят. **Для этого сначала рассмотрим несколько возможных форм дополнительных занятий со школьниками.**

1. Предположим, что в регионе существует школа, преподавание точных наук в которой осуществляется на очень высоком уровне, а набор учеников производится

на конкурсной основе. Обычно это физико-математическая школа, лицей или гимназия, расположенная в областном центре, а обучение в ней ведется при участии преподавателей из университета или ведущего вуза того же города. Если такая школа в регионе одна, то именно в рамках такой школы разумно организовать занятия со школьниками — потенциальными участниками олимпиад по информатике. Хорошо, если соответствующие занятия будет проводить преподаватель вуза или студент, как правило, бывший участник олимпиад, так как при решении олимпиадных задач, да и задач по программированию и информатике вообще, полезно знание основ дискретной математики и других разделов математики, которые входят в программу обучения студентов математических специальностей вузов. О методах и содержании подобной формы работы со школьниками можно будет прочитать в следующих лекциях. Такие занятия позволяют ребятам узнать много нового и интересного о любимом предмете, поэтому вне зависимости от результатов учеников данной школы в региональных соревнованиях потраченные усилия оказываются не напрасными. Положительный опыт организации таких занятий имеется в физико-математическом лицее города Кирова. Аналогичная форма работы с учащимися ведется в Нижнем Новгороде, Саратове, СУНЦ МГУ и в ряде других школ России. Школа, преподавание информатики в которой уже многие годы осуществляется на высоком уровне, существует и в небольшом городе Белорецке (Башкортостан), что также отразилось на результатах региона на последней олимпиаде. Представители ведущих физико-математических школ Казани, Ижевска, Мурманска, Оренбурга, Перми, Самары, Чебоксар, Челябинска и ряда других городов часто неплохо выступают на Всероссийской олимпиаде по информатике в силу общего высокого образовательного уровня этих школ. Проведение систематических дополнительных занятий по информатике в этих школах, включая решение олимпиадных задач, может только развить и закрепить такие успехи.

2. Предположим теперь, что одну лучшую школу в регионе выделить невозможно, то есть либо существует сразу несколько подобных школ, либо ни одна из школ региона особенно не выделяется в лучшую сторону среди других, по крайней мере в области школьной информатики. В этом случае подход к поиску талантливых ребят и организационные формы работы с ними должны быть иными. В данном случае дополнительные занятия имеет смысл организовать непосредственно при вузе или другом организационном центре для всех одаренных в области информатики школьников того или иного города. Примечателен здесь опыт Санкт-Петербурга. Так, например, в городской олимпиаде по информатике там обычно участвуют по крайней мере 400 школьников, а городской олимпиаде предшествуют еще и районные. Затем для 10—20 проявивших себя ребят проводятся недельные сборы, по результатам которых и формируется состав участников Всероссийской олимпиады от Санкт-Петербурга. Однако на этом работа с одаренными школьниками не заканчивается. Наоборот, в течение всего следующего учебного года многие школьники, проявившие себя на городской олимпиаде, посещают занятия во Дворце творчества юных, в Отделе техники которого существует Центр подготовки сборной Санкт-Петербурга по информатике. Проводят занятия в основном студенты Санкт-Петербургского университета — призеры всероссийских и международных олимпиад школьников прошлых лет. Кроме того, в течение всего учебного года школьники могут тренироваться и вместе со студентами института точной механики и оптики, традиционно успешно выступающими в командных соревнованиях по программированию. Много общего с данной организационной формой имеет работа со школьниками Кабардино-Балкарии. Аналогичным образом можно было бы строить работу и, например, в Москве и Екатеринбурге.

3. Следующая форма работы обычно возникает в условиях, аналогичных описанным в предыдущем случае. Отличия заключаются лишь в том, что в регионе существует сразу несколько городов и других населенных пунктов, школьники из которых способны к достижению высоких результатов. Кроме того, возможно, в регионе отсутствует достаточное количество квалифицированных кадров, способных дополнительно заниматься со школьниками на постоянной основе. Например, такая ситуация может сложиться в силу того, что бывшие победители региональных олимпиад учатся в столичных вузах и лишь летом могут помочь в обучении молодого поколения. В этом случае благодаря усилиям органов образования и ведущих педагогов большинство талантливых школьников региона собираются в летнем лагере (летней школе), сочетающем обучение, отдых и полезное общение со старшими товарищами. К работе в таких лагерях можно привлекать как студентов, так и преподавателей ведущих вузов России. Летние школы традиционно проводятся в Кировской, Московской, Орловской, Ярославской областях, Республике Саха (Якутия) и других регионах России. Очевидно, что подобная форма работы с одаренными школьниками может стать дополнением к любой другой.

4. Наконец, еще одной формой являются учебно-тренировочные сборы. Они уже упоминались при описании опыта Санкт-Петербурга. Проводятся такие сборы обычно для призеров региональных соревнований. Одной из целей подобных сборов является окончательный отбор школьников для участия во Всероссийской олимпиаде, с другой стороны, они могут иметь и образовательные функции. Поэтому при наличии организационных возможностей на сборы имеет смысл приглашать избыточное число школьников, особенно из невыпускных классов, с расчетом на их перспективы и для того, чтобы дать им толчок в развитии и дальнейшем самообразовании. Подобным же образом проводятся традиционные зимние и летние сборы для кандидатов в сборную России. Практика проведения таких сборов существует и в Москве.

Рассмотрим теперь, как выбор задач для проведения региональной и даже местной олимпиады может способствовать поиску наиболее одаренных школьников. На наш взгляд, ответ на этот вопрос зависит в том числе от того, какие из описанных форм занятий со школьниками получили распространение в том или ином регионе. Так, при наличии в регионе школ с углубленным изучением ряда дисциплин, в том числе и

информатики, набор заданий на олимпиадах, в которых участвуют ученики таких школ, должен иметь достаточный уровень сложности. А очень простые задания, доступные большинству участников, могут немного исказить объективно сложившуюся картину и не способствовать отбору лучших из лучших.

При наличии же возможности заниматься со школьниками дополнительно в течение практически всего учебного года задачи, предлагаемые на региональной олимпиаде, не должны требовать знания каких-либо специальных алгоритмов (в том числе и "традиционных" олимпиадных алгоритмов, например, на графах) или умения в совершенстве владеть всеми возможностями применяемого языка программирования. Ведь в этой ситуации цель олимпиады — обнаружить ребят, потенциально способных к высоким достижениям. А необходимыми знаниями и техникой программирования они смогут овладеть на занятиях. В противном случае легко упустить из виду большую часть талантливых ребят.

Если в регионе проводится только летняя школа, то отбор для участия во Всероссийской олимпиаде по информатике должен проводиться на достаточно сложных задачах, сопоставимых с задачами всероссийских олимпиад. Так как только такая подборка задач сможет выявить, кто из школьников может применять полученные летом знания при решении задач в условиях настоящей олимпиады. А это не в последнюю очередь зависит от самоорганизованности и работоспособности ребят, ведь для успешного выступления на олимпиаде необходимо научиться самостоятельно без ошибок реализовывать порой достаточно сложные, пусть даже знакомые, алгоритмы за очень короткий срок. А без приобретения практических навыков это сделать невозможно. То есть в течение двух первых учебных четвертей ребята должны самостоятельно отработать те знания и умения, которые они приобрели летом.

Если же региональной олимпиаде сопутствуют учебно-тренировочные сборы, то задачи областной (краевой и т.п.) олимпиады должны быть подобраны так, чтобы выявить школьников, техника программирования у которых уже достаточно высока и которые способны решать так называемые задачи "на сообразительность". Знание необходимых, но типичных алгоритмов они смогут получить на сборах. Улучшить же технику программирования за короткий срок практически невозможно, да и такая задача во время сборов обычно не ставится.

Наконец, рассмотрим случай, когда какие-либо систематические занятия со школьниками в регионе по той или иной причине организовать не удается. Иногда на это просто нет средств, иногда отсутствуют квалифицированные специалисты, одновременно являющиеся и энтузиастами олимпиадного движения. В таком случае важную роль играет выбор задач для проведения региональной олимпиады. Эти задачи должны практически полностью соответствовать по сложности задачам Всероссийской олимпиады по информатике. При этом возможно включение в набор задач одной "утешительной" задачи, которая окажется по силам большинству ребят. Конечно, у многих участников, например, областной олимпиады полученные при решении таких задач баллы будут весьма невелики. Однако если в регионе есть школьники, способные успешно конкурировать с представителями других областей на Всероссийской олимпиаде, то выявить их можно только таким образом. Если же набор задач будет достаточно простым, то высокие баллы смогут получить многие школьники и определить, кто же из них способен решать и более сложные задачи, будет уже невозможно. Причем потенциально наиболее сильные ребята могут и не оказаться абсолютными победителями региональной олимпиады. В случае отсутствия в регионе явного лидера наиболее оправданно послать для участия во Всероссийской олимпиаде ученика невыпускного класса. Так как он сможет получить практический опыт участия в олимпиаде высокого уровня, познакомиться с некоторыми новыми для него методами решения задач, получить ссылки на необходимую литературу. Тогда при наличии желания и работоспособности такой ученик сможет в течение следующего учебного года существенно повысить свою квалификацию. Если же представлять регион на Всероссийской олимпиаде каждый год будет новый ученик одиннадцатого класса, то успехи могут быть лишь случайными.

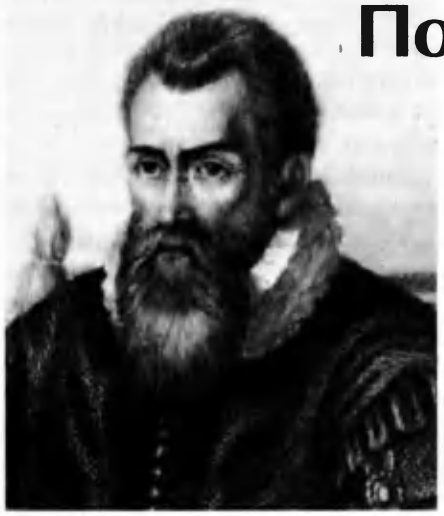
Сказанное выше справедливо и для городских и районных олимпиад, только предлагаемые на них задачи должны соотноситься с заданиями соответствующих республиканских, краевых или областных соревнований.

В заключение данной лекции отметим еще одну особенность представительства региона на Всероссийской олимпиаде, на наш взгляд, способствующую стабильным результатам. Положительным фактором может оказаться то, что в течение нескольких лет школьников на олимпиаду будет сопровождать один и тот же человек. Причем это может быть как педагог, непосредственно работающий со школьниками, так и представитель органов образования. Такой руководитель будет знаком со всеми нюансами правил проведения олимпиады и технологией проверки решений задач, которые подвержены со временем некоторым изменениям. Так олимпиады начала 90-х годов значительно отличаются от олимпиад нового тысячелетия. Даже простое знакомство с правилами позволит такому руководителю дать квалифицированные советы своим школьникам и правильно их психологически настроить. А присутствие руководителя на таких этапах олимпиады, как тестирование решений и разбор задач, позволит приобрести много новых знаний, которые пригодятся в его дальнейшей работе с учениками. При ежегодной же смене руководителей такой опыт оказывается невостребованным. Хотя справедливости ради стоит заметить, что иногда именно смена руководителя приводит к позитивным переменам в выступлениях школьников того или иного региона.

Чтобы оказать методическую помощь тем педагогам, которые непосредственно в олимпиадах по информатике высокого уровня еще не участвовали, следующую лекцию мы посвятим разбору правил олимпиады и общим рекомендациям для школьников — будущих участников олимпиад.

Последователи Джона Непера

Д.М. Златопольский,
Москва



В статье, посвященной Джону Неперу ("Информатика", № 15/2001), отмечается, что идея, положенная в основу предложенного им в начале XVII века приспособления для умножения (так называемых палочек Непера), породила большое количество идей по его усовершенствованию. Опишем некоторые из них.

В 1668 г. Каспар Шот в книге "Organum mathematicum" предложил заменить палочки Непера цилиндрами, на поверхности которых нанесены ленты с написанными на них числами, как и на палочках Непера. Цилиндры помещались в ящике параллельно друг другу и могли вращаться. Повернув каждый цилиндр так, чтобы их верхние цифры составляли первый сомножитель, мы можем прочитать искомое произведение.

В 1673 г. Пти изготовил арифметический барабан (барабан Пти). В XVIII в. усовершенствования делали Леопольд в 1727 г., М.Форпус в 1728 г. (пифагорова монзула), Праль в 1789 г. (переносная арифметика), Брюсон в 1790 г. и др.

Так, в 1892 г. Прюво-Ле-Гюэ заменил счетные палочки счетными брусками, аккуратно помещенными в ящике, имелся также и ящик для выкладывания результатов. Умножение на многозначные числа сводилось к умножению на однозначные и складыванию на бумаге со сдвигом. Несмотря на примитивность устройства, действие умножения с этим прибором совершалось просто и достаточно быстро.

В этом же году Эггис создал прибор для умножения, в котором вместо палочек применил узкие полосы, закрепленные в одном футляре в виде записной книжки. Полосы передвигались заостренной палочкой, и умножение многозначного числа на однозначное после соответствующего передвижения полос сводилось к правильной считке. Следует иметь в виду, что во всех приборах, основанных на палочках Непера, необходимо внимательно следить за считкой, так как десятки одного произведения нужно складывать с единицами произведения следующего разряда. Для избежания ошибок при считке Эггис выкрасил в красный цвет правую половину одной полосы и левую — смежной (и так далее через полосу), так что те числа, которые нужно складывать, будут стоять на частях полос, окрашенных в один цвет. При умножении многозначных чисел частные произведения складываются на бумаге.

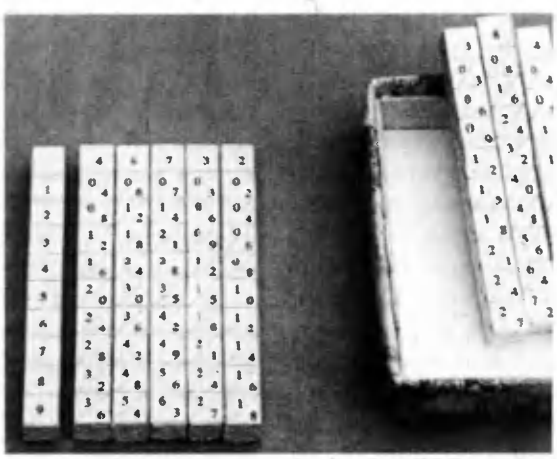
Бруски для умножения предложил также в 1891 г. И.Блятер из Вены, назвав их "Упрощенные таблицы умножения и деления". Однако для деления бруски Блятера неудобны. При умножении сбор брусков, выбор частных произведений и их сложение требуют довольно много времени. При умножении двух пятизначных чисел экономия во времени еще не ощущается по сравнению с умножением на бумаге. При шестизначных числах выигрывается около 20% времени. При еще больших числах выгода во времени увеличивается.

В том же 1891 г. Женайль и Мокас усовершенствовали прибор Непера. Они исходили из того, что произведения, записанные на палочках Непера, могут быть получены с помощью особой цифровой шкалы, помещенной на каждом из десяти брусков (0, 1, 2, ..., 9), если только дать указатель того пути, который должен проделать глаз от одного бруска к другому. Роль таких указателей выполняли черные треугольники, начерченные на каждом бруске.

При помощи этих брусков можно производить умножение однозначного числа на число, в котором одна и та же цифра может повторяться четыре раза. Чтобы умножить любое число на однозначное, Женайль составил прибор из вращающихся цилиндров. Если иметь 20 таких цилиндров, то мы получим прибор, при помощи которого можно получить произведение однозначного числа на любое число до 10^{20}.

Для умножения двух многозначных чисел вначале составляются частные произведения, а затем они складываются на бумаге.

Самый простой прибор для сложения представлял бы собой линейку с делениями, вдоль которой может перемещаться другая такая же линейка. Сложение двух чисел сводилось бы к двум последовательным передвижениям линейки. Аналогично можно осуществить сложение, вращая один круг внутри другого. Наиболее известным прибором, основанным на передвижении линеек, является прибор К.Казе. В 1720 году он предложил использовать несколько пар линеек и расположить их рядом. Первая пара справа служила для складывания единиц, вторая — для десятков и т.д. Переносить десятки из разряда в разряд приходилось самому считающему. Весь прибор был смонтирован на одной пластинке, линейки имели зубчатый край и передвигались при помощи специально заостренной палочки. В Парижском музее искусств и ремесел хранятся три прибора Казе, которые названы "арифметическими машинами".



Литература

Апокин И.А., Майстров Л.Е. Развитие вычислительных машин. М.: Наука, 1974, 397 с.

Обязательный минимум содержания образования по информатике:

и в нем нам хочется дойти до самой сути

А.Г. Гейн

Продолжение. Начало см. в № 24, 30, 35, 36/2001

Глава 5. Алгоритмы и исполнители

В этой главе мы несколько отступим от избранной нами ориентации только на анализ содержания пунктов обязательного минимума и в данном предисловии к очередной главе остановимся на том, как изменялись целевые установки преподавания школьной информатики на протяжении не столь уж долгой истории этого учебного предмета.

Весной 1995 года появилось совместное постановление ЦК КПСС и Совета Министров СССР о введении в общеобразовательной школе курса "Основы информатики и вычислительной техники". Внедрение курса шло за счет часов физики, поэтому учителя информатики рекрутировались в основном из учителей физики и частично математики. Впрочем, иногда это был учитель труда или физкультуры — мужик все-таки, а тут речь идет о технике. Для всех рекрутированных организовывались краткосрочные (как правило, месячные) курсы переподготовки. Как вспоминала потом одна из учительниц: "Кинули мы между собой жребий — выпал он на меня. Ну я поплакала и поехала на курсы переподготовки..."

Впрочем, поначалу черт оказался не так уж и страшен — ведь это была безмашинная информатика. Компьютеры дети видели только на картинках, на уроках же в основном писали алгоритмы на специально придуманном языке, а учитель проверял их правильность. Как и во всех других предметах, учитель оставался единственной инстанцией, решавшей, правильно или неправильно выполнено задание учеником.

Преподавание информатики в машинном варианте — это совсем иная педагогическая обстановка. Окончательный вердикт алгоритму о его правильности выносит не учитель, чье знание учеником должно восприниматься как истина в последней инстанции, а какая-то машинюшка, которую почему-то никак не удается заставить исполнить то, что написано в задании. Эту стрессовую ситуацию выдержали далеко не все учителя первой волны. И примерно два года спустя — по мере появления в школе компьютерной техники — им на смену преподавать информатику приходят профессиональные программисты. Они начинают учить школьников какому-нибудь языку программирования и относиться к компьютеру как большому и чуть более "умному" калькулятору. Впрочем, компьютеры, имевшиеся в то время в школе, и программное обеспечение к ним вполне оправдывали такое к себе отношение.

Остановимся в этой точке. Потому что крен в сторону изучения алгоритмизации, присутствовавший с самого начала введения курса, здесь становится еще больше. Вот как описывает этот период один из руководителей информатизации образования в Пермской области Е.К. Хеннер.

"На начальном этапе введения в школу предмета "Основы информатики и вычислительной техники" определяющее влияние оказала принятая в то время концепция компьютерной грамотности. Превалирующим был тезис: "компьютерная грамотность — это умение программировать". Этот тезис определял содержание предмета (программ и учебников) и одновременно содержание развивающего направления, поскольку развитие алгоритмического мышления считалось главной целью.

В 90-х годах наконец до нас начали доходить современные компьютерные информационные технологии (приблизительно с 10-летним опозданием). Постепенно стало укрепляться понимание, что компьютерная грамотность и умение программировать — не совсем одно и то же. Началось осмысление того факта, что название дисциплины "Информатика" происходит от слова "информация", что именно информация, а не алгоритм является центральным понятием предмета. Алгоритм — всего лишь одна из разновидностей информации — управляющая информация".

Вот такая длинная цитата из его статьи в газете "Информатика", № 14 за 1997 г. На самом деле компьютерные информационные технологии были известны у нас и развивались без всяких опозданий. Они не были общедоступными — это верно. Но это совсем другой поворот темы. На мой взгляд, политический. Те, кому сейчас около пятидесяти и более, еще помнят те времена, когда пишущие машинки стояли на учете в компетентных органах — как-никак множительная техника! О каком массовом владении компьютерными информационными технологиями могла идти речь в таком обществе? Это сейчас любой текст можно набрать и распечатать на принтере в любом количестве экземпляров[1]. Это сейчас можно записать информацию с одного компьютера и перенести на другой. Это сейчас можно выйти в Интернет или отправить электронное письмо хоть в процветающие США, хоть в самую что ни на есть Тмутаракань.

Величайшее политическое искусство отцов-основателей школьной информатики состояло в том, что им удалось убедить руководство страны смотреть на компьютер как на средство программирования, а не информационное средство. Главные учебные цели — воспитание алгоритмического мышления, главное понятие — формальный исполнитель, не задумывающийся о сути и цели исполняемого алгоритма. Это ли не идеал руководства нашей страной в то время, желающего видеть руководимое общество комплектом винтиков-исполнителей? Ведь не удалось же ввести в школу курс кибернетики даже после того, как она перестала именоваться "продажной девкой капитализма". Была разработана программа [28]* (в ряде пунктов весьма близкая к некоторым современным программам по информатике), были экспериментальные курсы (см. [22]), но все кончилось ничем. Оно и понятно: разве можно всех школьников учить основам управленческой науки — они же тогда начнут задумываться, правильно ли управляется страна, область, город, район...

Школьной информатике повезло больше — она не только уцелела в школе, но и все глубже в нее проникает. И уже на самом начальном этапе В.А. Каймин внушал учителям, что компьютерная грамотность — это умение читать, писать, рисовать, считать с помощью компьютера (не следует забывать, что именно учебник [18], созданный под руководством В.А. Каймина, стал победителем конкурса учебников информатики, проводившегося Министерством образования в 1985 г. — году, когда в школе только-только появилась информатика). К тому времени относится и разработка под руководством Ю.А. Первина программно-методического комплекса "Роботландия", в котором детей начальной школы учат в первую очередь простейшим приемам обработки информации с помощью компьютера. Г.В. Лебедев, соавтор А.Г. Кушниренко по одному из наиболее распространенных в то время учебников информатики, в статье [27] ставит знак равенства между информатикой и программированием, но не утверждает, что это компьютерная грамотность. И никто из стоявших у истоков школьной информатики не говорил, что компьютерная грамотность — это умение программировать. Вот как расшифровывал понятие компьютерной грамотности в своей работе [3], опубликованной еще в 1988 г., академик А.П. Ершов. По его словам (см. с. 360), компьютерная грамотность предполагает

"— навыки грамотной постановки задач, возникающих в практической деятельности, для их решения с помощью ЭВМ;

— навыки формализованного описания поставленных задач, элементарные знания о методах математического моделирования и умение строить простые математические модели поставленных задач;

— знание основных алгоритмических структур и умение применять их для построения алгоритмов решения задач по их математическим моделям;

— понимание устройства и функционирования ЭВМ и элементарные навыки составления программ для ЭВМ по построенному алгоритму на одном из языков программирования высокого уровня;

— навыки использования основных типов информационных систем и прикладных программ общего назначения для решения с их помощью практических задач и понимания основных принципов, лежащих в основе функционирования этих систем;

— умение грамотно интерпретировать результаты решения практических задач с помощью ЭВМ и применять эти результаты в практической деятельности".

Отождествление компьютерной грамотности и программирования, указанное в статье Е.К. Хеннера, — это отражение восприятия школьной информатики учительской общественностью на том этапе развития курса информатики. Как было выше отмечено, это результат как общей целевой установки курса на развитие алгоритмического мышления, так и реалий кадровой и технической оснащенности школьной информатики в те годы. Спустя 5 лет после появления цитировавшейся выше статьи А.П. Ершова в статье [11], излагающей целевые установки авторского коллектива учебника [25], указывается: "Для

[1] Более того, авторы некоторых учебников по информатике (см., например, [17], с. 24) к программным продуктам общего назначения относят настольные издательские системы, подразумевая, что каждый может, а значит, должен уметь сам изготовить печатное издание с помощью такой системы (в указанном учебнике соответствующему обучению посвящено 98 страниц из 298). Судьбе авторского коллектива, пропагандирующего такие идеи, не только в 37-м, но и в 80-м году не позавидуешь.

* Список литературы будет опубликован в № 39.

нас важно то, что существует специфический стиль мышления, который принято называть алгоритмическим. Развитие этого стиля мышления и есть *основная цель* курса информатики" (курсив мой. — **А.Г.**).

Мы отложим до главы 6 анализ того, как в курс школьной информатики внедрялись информационные технологии (об осознании их роли учительской общественностью говорится и во второй половине цитаты из статьи Е.К. Хеннера). Здесь же только отметим, что авторы "Концепции содержания образовательной области "Информатика и информационные технологии" в двенадцатилетней школе" (см.: "Информатика", № 15/2001) к основным тенденциям развития указанной образовательной области относят "отказ от обязательного освоения школьниками сред и языков профессионального программирования как составной части общеобразовательной подготовки школьников". Тем не менее изучение алгоритмов, средств их описания и методов построения остается обязательным пунктом раздела "Теоретическая информатика", который предусмотрен в курсе информатики указанной концепцией.

§ 23. Понятие алгоритма: свойства алгоритмов, исполнители алгоритмов, система команд исполнителя

Начнем, как и раньше, с определений, которые даются понятию "алгоритм" в наиболее распространенных школьных учебниках.

В [35], с. 17, и [23], с. 16: "Под **алгоритмом** понимают понятное и точное предписание (указание) исполнителю совершить последовательность действий, направленных на достижение указанной цели или на решение поставленной задачи".

В [25], с. 25: "**Алгоритмом** называется программа, записанная на школьном алгоритмическом языке".

В [18], с. 59: "**Алгоритм** — это последовательность действий со строго определенными правилами выполнения"; с. 272: "**Алгоритм** — детальный план действий на получение определенных результатов".

В [16], с. 60: "**Алгоритм** — совокупность правил выполнения определенных действий, обеспечивающих решение задачи".

В [14], с. 195: "**Алгоритм** — это последовательность команд, управляющих работой какого-либо объекта"; с. 198: "**Алгоритм** — понятное и точное предписание исполнителю выполнить конечную последовательность команд, приводящую от исходных данных к искомому результату".

В [7], с. 139: "**Алгоритм** — это организованная последовательность действий, допустимых для некоторого исполнителя, приводящая к определенному результату".

Примерно одно и то же говорится во всех этих определениях, но все же по-разному. Попытаемся разобраться во всех нюансах[2].

Прежде всего мы должны констатировать, что алгоритм — базовое понятие информатики и, как практически всякое базовое понятие, не имеет строгого определения. В "Математической энциклопедии" (см. [32], с. 205) говорится: "Понятие алгоритма в его общем виде принадлежит к числу основных первоначальных понятий математики, не допускающих определения в терминах более простых понятий". Так что все приведенные выше формулировки следует понимать как описания, разъяснения понятия "алгоритм", а не как строгое его определение. Но из всех перечисленных выше учебников только в [7] говорится, почему предложенную формулировку нельзя считать определением: в ней не объясняется, например, что означают слова "организованная", "действия", "результат". Смысл этих и других слов, входящих в формулировку, будет раскрываться в последующем — при изучении алгоритмических конструкций, при разъяснении понятия допустимого действия исполнителя, при установлении связей результата с достижимыми целями исполнителя и т.д.

Анализ различий в вышеприведенных формулировках мы начнем с рассмотрения двух, как нам представляется, наиболее полярных формулировок.

Повторим еще раз формулировку из учебника [25]: "Алгоритмом называется программа, записанная на школьном алгоритмическом языке". Здесь алгоритм выступает как частный случай программы, причем программа на Паскале или Бейсике к алгоритмам, по-видимому, никакого отношения не имеет. Более того, возникает ощущение, что алгоритмы вообще появились только в связи с изобретением школьного алгоритмического языка. Знаменитый алгоритм Евклида нахождения наибольшего общего делителя двух натуральных чисел, изложенный им в своих "Началах", — это, по мнению авторов [25], вовсе не алгоритм: ведь Евклид, конечно, не знал школьного алгоритмического языка. Нельзя не отметить и усиленное внимание к синтаксису алгоритмического языка с первых минут его изучения (см. [25], с. 27), что подчеркивает формальный характер этого языка. Иными словами, в этом учебнике фактически

[2] При этом мы воздержимся от повторения приводимого во всех учебниках рассказа, откуда взялось название "алгоритм", хотя и здесь имеются весьма различные трактовки. Поистине, в нашей стране даже мировая история непредсказуема.

изучается не алгоритмизация, а программирование. Впрочем, эту подмену алгоритмизации программированием вовсе не отрицают и сами авторы учебника [25]. Более того, в [27], с. 27, один из авторов учебника Г.В. Лебедев выдвигает тезис, что "различение алгоритмизации и программирования — одна из самых крупных ошибок первых учебников" (речь идет о школьных учебниках [35] и [36], созданных под руководством А.П. Ершова).

Рассмотрим теперь формулировку, предложенную авторами [16]: "Алгоритм — совокупность правил выполнения определенных действий, обеспечивающих решение задачи". Попытаемся понять, что сказали авторы учебника. Вот, к примеру, задача — изобличить преступника. Следователь выполняет определенные действия для решения этой задачи. Имеются правила выполнения этих действий, скажем, нельзя производить обыск без санкции прокурора. Эти правила собраны в так называемый процессуальный кодекс. Но никому не придет в голову назвать процессуальный кодекс алгоритмом.

Или другой пример. Наверное, все согласятся, что поведение ученика в школе — это его определенные действия, направленные, разумеется, на решение какой-то задачи, скажем, покушать в школьном буфете. Существуют правила поведения ученика в школе, например, запрещающие толкаться. Но разве совокупность правил поведения в школе — это алгоритм?

Подобная небрежность плоха еще и тем, что действительно существуют системы программирования, где для решения задачи записывается не алгоритм, а совокупность правил и фактов. К таким системам относится так называемое логическое программирование, а одним из наиболее распространенных языков логического программирования является язык Пролог (см. по этому поводу [14], с. 162—163, 301—303).

Конечно, при том определении алгоритма, которое дано в [16], это понятие весьма далеко располагается от понятия программы. Авторы [16] описывают соотношение "алгоритм — программа" так:

"Каждый алгоритм может быть реализован разными программами, совпадающими только в главном".

"Для любого объекта алгоритм — ориентировочный план действий без учета его особенностей".

"Программа — это реальные действия, которые должен выполнять конкретный объект в соответствии с заданным алгоритмом".

А в алгоритме, значит, действия нереальные? И что значит совпадение программ в главном? И на что годится алгоритм, не учитывающий особенности объекта? Еще в учебнике [4] более чем десятилетней давности приводился некий алгоритм переправы через Волгу в районе г. Саратова:

"Подойти к реке.

Войти в реку.

Идти по дну, пока не выйдешь на другой берег".

Обычный человек такой алгоритм исполнить не может, а робот-подводник исполнит его запросто. Можно ли, составляя алгоритм, не учитывать особенности объекта? Впрочем, учитывать надо особенности не только исполнителя алгоритма, но и среды. Ведь если дело происходит не около Саратова, а в верховьях Волги, где она неглубока, то этот алгоритм исполним и человеком.

Уже это обсуждение показывает, что понятие алгоритма нельзя рассматривать изолированно, оно связано с понятием исполнителя, т.е. того объекта или субъекта, который будет исполнять алгоритм, с понятием среды, в которой действует исполнитель, и т.д. Не случайно и авторы "Обязательного минимума..." после слов "Понятие алгоритма" поставили двоеточие и расшифровали пункт словами, на первый взгляд с понятием алгоритма и не связанными (сравните, например, с формулировкой, данной в [18], — ни малейшего намека на каких-то там исполнителей).

Начнем с понятия исполнителя.

Решение любой задачи (в смысле достижения той или иной цели) всегда состоит из выполнения какой-либо последовательности действий. Создание этой последовательности, иными словами, разработка плана решения задачи — это процесс, относимый, как правило, к творчеству. Другое дело — реализация уже имеющегося плана действий. Ее можно поручить субъекту или объекту, который не обязан вникать в суть дела, а возможно, и не способен его понять. Такой субъект или объект принято называть **формальным исполнителем** (для краткости его обычно называют просто исполнителем).

В своих действиях исполнитель руководствуется инструкцией, при этом каждое указание инструкции имеет для исполнителя только один смысл, может быть им понято и должно быть исполнено однозначно. Согласно § 20 это означает, что *инструкция исполнителю дается всегда на соответствующем формализованном языке.* Более того, исполнитель, реализованный на компьютере, понимает только те инструкции, которые даны ему на формальном языке. Инструкцию, записанную уже на формальном языке, обычно называют **программой**. Скажем, запись фактов и правил на языке Пролог тоже называют программой (см., например, [14], с. 300), хотя никакого алгоритма при этом не составляется.

Таким образом, по существу, мы можем ограничиться рассмотрением инструкций для исполнителя, записанных на формализованном языке. Как уже было сказано, такая инструкция описывает план решения задачи. Что такое план решения задачи, можно понимать по-разному. Наиболее распространенный вари-

ант — это указание последовательности действий, которые надлежит совершить исполнителю, чтобы, по мнению составителя плана, решить задачу. В этом случае формализованный язык, понимаемый исполнителем, должен содержать имена всех действий, которые может совершать исполнитель. Среди этих действий имеются прежде всего операции над объектами, с которыми работает исполнитель. Совокупность этих объектов и отношений между ними составляет **среду**, в которой исполнитель "обитает"[3]. В табл. 24 описаны исполнители, встречающиеся в учебниках информатики, рекомендованных Министерством образования РФ для общеобразовательных школ. Знакомство с ними позволит читателю иллюстрировать приводимые ниже общие рассуждения конкретными примерами.

Отметим две особенности перечисленных в табл. 24 исполнителей. Первая из них состоит в том, что каждый фигурирующий в ней исполнитель определен описанием структуры среды обитания и набором допустимых действий над объектами этой среды. Исполнитель, заданный таким способом, называется исполнителем **алгоритмического типа**. Такого исполнителя удобно представлять себе как некий объект, состоящий из устройства управления и набора инст-

Таблица 24

Исполнитель	Среда		Действия над объектами
	Объекты	Отношения	
Перевозчик [17, 10, 12]	Волк; Коза; Капуста; Берег1; Берег2[4]	Быть съеденным, если находятся на одном берегу в отсутствие Перевозчика; Волк, Коза и Капуста могут находиться на любом из двух Берегов	Перемещение ровно одного объекта — Волка, Козы или Капусты — с одного Берега на другой
Кукарача [39]	Клетки; Кубики с символами	Клетки образуют квадратное поле 10 × 10; Кубик размещается в точности на одной из Клеток	Переход на одну Клетку (влево, вправо, вверх, вниз); перемещение Кубика в направлении движения; проверка, есть край поля или нет; проверка, имеется ли на Кубике заданный символ
Робот [25]	Клетки радиоактивные; Стены	Клетки образуют бесконечное поле; Стена может располагаться только на стороне Клетки[5]	Переход на одну Клетку (влево, вправо, вверх, вниз); закрашивание Клетки; определение температуры и уровня радиации в Клетке, на которой стоит Робот; определение, стоит или не стоит Стена на Границе клетки; определение, закрашена ли Клетка, на которой стоит Робот
Чертежник [4]	Прямоугольный лист	—	Смещение на 1; поворот на 90° против часовой стрелки; рисование отрезка во время смещения; проверка, есть край поля или нет в направлении движения
Паркетчик [4, 5, 18]	Клетки; Колонны; Плитки красные и зеленые	Клетки образуют прямоугольное поле, ограниченное Колоннами; Колонна может стоять только на свободной Клетке; Плитка покрывает ровно одну Клетку; на одной Клетке не может лежать двух плиток; Колонна и Плитка не могут находиться на одной Клетке	Переход на одну Клетку (влево, вправо, вверх, вниз); размещение плитки заданного цвета на свободной Клетке; снятие Плитки; определение, стоит ли Колонна на Клетке, соседней слева, справа, снизу, сверху к Клетке, где стоит Паркетчик; определение, лежит ли Плитка данного цвета на той Клетке, где стоит Паркетчик

[3] Взаимодействие исполнителя со средой делает необходимым иметь не только действия над объектами, но и действия по проверке истинности тех или иных высказываний, относящихся к обстановке в той среде, где действует исполнитель. На рассмотрении этих действий мы остановимся ниже, в § 35.

[4] В [1] на с. 37 при описании среды Перевозчика (здесь в отличие от [39] он называется Крестьянин) в число объектов включена еще и лодка. На наш взгляд, лодка — это не более чем антураж, поясняющий, почему перемещать с одного берега на другой можно ровно один объект. Вместо лодки в распоряжении Перевозчика мог быть мост малой грузоподъемности или пропеллер, как у Карлсона... Ненужность рассмотрения лодки как объекта среды становится сразу ясной, как только мы взглянем на систему команд исполнителя, приведенную на той же странице [1], — в ней фигурируют все перечисленные нами объекты, а лодки нет.

[5] Так, по-видимому, надо понимать слова "стена стоит между клетками" в учебнике [25], с. 25.

рументов, посредством которых он и выполняет допустимые действия[6]. Инструменты и соответствующий набор действий у каждого исполнителя индивидуальны[7]. Что касается устройства управления, то мы можем не вникать в возможные различия в их физической реализации. Важно лишь то, что для всех исполнителей их устройства управления понимают одни и те же формы организации действий, так называемые алгоритмические конструкции, речь о которых пойдет в § 25 и § 26.

Вторая особенность состоит в том, что каждый из этих исполнителей выделен из своей среды обитания. Такой взгляд на исполнителя достаточно характерен для большинства существующих курсов школьной информатики. Он инспирирован, по-видимому, антропоцентричной позицией, которую в XX веке достаточно долго занимало человечество в своих взглядах на мир (человек — царь природы, человек — вершина эволюции природы, преобразование природы — цель деятельности человека и т.п.), закреплен тем обстоятельством, что человек долгое время служил едва ли не единственным примером исполнителя (не считая дрессированных животных в цирке), и поддержан методическими решениями, найденными авторами этих курсов. В дальнейшем происходил пересмотр роли этого понятия, и, скажем, в [25] оно выступает уже как базовое, определяющее парадигму программирования, близкую к объектно-ориентированному подходу. Исполнитель при этом рассматривается как один из объектов среды. А раз исполнитель — всего лишь один из объектов, то и над ним допустимо выполнять те или иные действия, меняющие его состояние. Исторически, правда, было наоборот — реально конструировавшиеся исполнители нередко имели в своем арсенале команды, менявшие состояние самого исполнителя. Так, самый первый учебный исполнитель — придуманная С.Пейпертом черепашка Лого имеет команды, изменяющие ее собственное состояние ("Подними перо" и "Опусти перо"). Но ни сам С.Пейперт [38], ни многочисленные его последователи *никогда* не обращали внимания на этот аспект данного исполнителя. Более того, всегда подчеркивалось его выделение из среды, ибо Лого, по замыслу С.Пейперта, служит не средством обучения программированию (и изучения информатики), а средством, моделирующим мыслительную деятельность ребенка при изучении математики. Тем самым налицо именно антропоцентричная позиция в концепции исполнителя[8].

Тем не менее наличие исполнителей, программно меняющих свое состояние, по-видимому, способствовало осознанию того, что исполнитель вовсе не обособлен от среды своего обитания. Отметим параллелизм в изменении взглядов человечества на место человека в окружающем его мире и взглядов программистов на место исполнителя в среде его обитания. Экологические проблемы заставили людей осознать, что они находятся не над природой (или вне ее), а являются ее составной частью. Точно так же современное программирование рассматривает исполнителя как один из объектов среды. Его отличие от других объектов состоит в том, что он активен, причем его активность двояка — он воздействует на окружающую среду и сам реагирует на изменение среды, меняя соответствующим образом свое состояние. Ясно также, что в рамках одной среды могут существовать несколько исполнителей. К сожалению, на сегодняшний день относительно широкое распространение получила только одна среда учебного назначения, допускающая несколько исполнителей, — ЛогоМиры. В этой среде возможно запрограммировать и запустить одновременно несколько черепашек Лого.

Любой исполнитель, решая указанную ему задачу, выполняет некоторую последовательность действий. Эта последовательность действий, которую надлежит исполнить, называется **алгоритмом**. Разумеется, алгоритм должен содержать только те действия, которые допустимы для исполнителя. Но, чтобы исполнитель исполнил алгоритм, мы должны сообщить ему этот алгоритм записанным на языке, который понятен исполнителю. В соответствии с тем, что было сказано выше, алгоритм в этом случае превращается в программу. Вместо действий в такой программе будут записаны **команды** исполнителю на выполнение соответствующих действий. Совокупность всех команд, которые умеет выполнять исполнитель, называется **системой команд данного исполнителя**. Именно такое определение системе команд исполнителя (сокращенно СКИ) дано в [7], с. 135; [14], с. 196; [23], с. 18.

---

[6] Оговоримся, что такая точка зрения не является единственной. В [25] устройством управления исполнителем выступает ЭВМ, сам же исполнитель никаким устройством управления не обладает (см. с. 43). При таком понимании естественно единство всех существующих алгоритмических конструкций. Однако тогда сам компьютер оказывается выведенным из числа формальных исполнителей, иначе получается, что должен существовать компьютер, который управляет компьютером, и т.д.

[7] Инструменты исполнителя, как и объекты среды, над которыми он выполняет действия, надо понимать в широком смысле — это совсем не обязательно физически существующие объекты и инструменты, они вполне могут быть абстрактными. Например, в роли объектов могут выступать целые числа, а инструментами будут операции над числами и т.п.

[8] Антропоморфизм Черепашки — один из краеугольных камней в концепции С.Пейперта, на котором выстроена методика использования Лого для обучения (см. [38], с. 68, 72 и др.). Наличие команд, меняющих состояние исполнителя, здесь скорее невыдержанность "чистоты стиля", нежели обдуманное рассмотрение исполнителя как некоего объекта среды.

Различие между алгоритмом и программой довольно тонкое. Прежде всего отметим, что программа состоит из команд, записанных на формальном языке, т.е. синтаксис команд задан однозначно. Алгоритм же может записываться словами естественного языка, и единственное требование к языку, посредством которого фиксируется алгоритм, — это однозначность смысла каждой записи.

Рассмотрим такие два алгоритма:

How to make a good tea:	**Как приготовить хороший чай:**
Bring fresh water to boil.	Вскипятите свежую воду.
Warm tea-pot by rinsing out with hot water.	Ополосните заварочный чайник крутым кипятком.
Put one teaspoonful of tea per cup into the tea-pot and pour immediately the boiling water into the tea.	Положите чай в заварочный чайник из расчета одна чайная ложка на чашку и сразу же залейте кипятком.
Stir the tea after 3—5 minutes.	Через 3—5 минут размешайте.
Add sugar to taste.	Добавьте сахар по вкусу.

Это один и тот же алгоритм (признаемся, что и списан он с одной упаковки чая), но алгоритм слева написан для того, кто говорит по-английски, а алгоритм справа — для нашего соотечественника. Ясно, что каждый из них будет выполнять одни и те же действия, и поэтому, наблюдая за их работой, мы не сможем отличить одного от другого. Более того, можно заснять на видеокассету все манипуляции при исполнении алгоритма, а затем показать любому жителю Земли, и ему будет ясно, как заваривать чай. И совершенно не важно, какой язык понимает этот житель![9]

Пришла пора немного отвлечься и попить чайку. Например, того, что приготовили по написанным выше алгоритмам какой-нибудь русский школьник Ваня и его английский приятель Джонни. К нашему величайшему изумлению, чаек оказался разным: от Ваниного чая язык прилипает к губам, а чай Джонни почти несладкий. Почему один и тот же алгоритм в разном исполнении привел к столь различным результатам?

Очевидно, все дело в последнем действии: добавлении сахара по вкусу. Это, конечно, допустимое действие для каждого из указанных исполнителей. Просто у каждого свой вкус. Но важен вывод из этого рассмотрения: *результат исполнения алгоритма зависит от того, какой исполнитель этот алгоритм исполняет.*

[9] Вот, кстати, еще одно преимущество, с точки зрения человека, видеоинформации перед информацией символьной.

Если кому-то не нравится бытовой пример, на основе которого был сделан вывод, приведем другой — более близкий к компьютерным реалиям.

Пусть с помощью микрокалькулятора выполняется следующий алгоритм:

- Сложить 83,2438 и 57,6847;
- Полученный результат умножить на 10.

Будут ли одинаковыми результаты исполнения этого алгоритма для калькуляторов с 6- и 8-разрядными дисплеями?

Каждому ясно, что результаты получатся разными, поскольку в первом случае результат будет округлен, а во втором — нет.

Указанный эффект связан с тем, что алгоритм рассчитан на некоего идеального исполнителя, а реальный исполнитель каждую команду выполняет в меру своих возможностей. На любом из общепринятых алгоритмических языков можно написать, к примеру, команду вычисления суммы синусов чисел 2 и 3. Но такое вычисление в силу ограниченности разрядной сетки всегда будет производиться приближенно, а следовательно, неизбежно будут и ошибки округления. От нас, пользователей данным алгоритмическим языком, скрыто, как именно они обрабатываются. И на самом деле для каждого алгоритмического языка (и транслятора с него, а также набора значений физических характеристик компьютера) это может происходить по-своему. Так что вполне можно ожидать разных результатов для разных алгоритмических языков (даже на одном компьютере).

Ограниченность разрядной сетки (т.е. работа компьютера в основном с округленными числами) приводит, в частности, к тому, что могут не выполняться привычные с детства законы ассоциативности сложения и умножения, позволяющие не писать скобки при вычислении сумм или произведений более чем двух чисел[10].

Итак, повторим еще раз: алгоритм — это неопределяемое понятие, смысл которого состоит в том, что посредством алгоритма задается последовательность действий, допустимых для некоторого исполнителя, обеспечивающая достижение поставленной цели. Но

[10] Примеры, демонстрирующие нарушение ассоциативности, хотя и не сложны, но их обсуждение потребовало бы достаточно подробного рассмотрения вопросов представления чисел в компьютере. Желающие могут найти соответствующую информацию, например, в [19]. Разумеется, профессиональный программист должен досконально знать и представление чисел в памяти компьютера (в том числе и в различных его регистрах), и указанные особенности машинной арифметики. Целесообразность же включения этого материала в общеобразовательный курс информатики сомнительна; что касается курса информатики, профильно ориентированного на подготовку будущих профессиональных программистов, то там такое рассмотрение, разумеется, уместно. Отметим, однако, что в одном из общеобразовательных школьных учебников по информатике, представленных на конкурс в 1987 г., эти вопросы рассматривались.

под такое описание понятия "алгоритм" подходят самые разнообразные инструкции, рецепты (например, из кулинарной книги), руководящие указания и т.п. Вот примеры подобных инструкций:

1. Добавьте щепотку соли. Слегка потрясите, чтобы смесь стала комковатой. Подогрейте коньяк в маленькой кастрюльке и влейте его в смесь.
2. Если врач не прописал иначе, то 3—4 раза в день по 15—20 капель, лучше всего в горячей сладкой воде.

Чтобы отличать алгоритмы от других инструкций, отметим несколько общих черт алгоритмов и часто признающихся характеризующими алгоритмы среди множества всяческих инструкций. Обычно эти характеристики называют **свойствами** алгоритмов. В разных учебниках выделяют разные наборы свойств. Например, в [15], с. 178, названо три свойства: понятность, точность и конечность. Мы в своем изложении этого вопроса будем следовать в основном работам [21] и [30].

Одним из свойств алгоритмов является **дискретность**. Под дискретностью понимается то, что алгоритм состоит из описания последовательности тактов обработки, организованной таким образом, что в начальный момент задается исходная ситуация, а в каждый следующий момент ситуация преобразуется на основе данных, полученных в предшествующие такты обработки (см. [29], с. 10; [2], с. 67). Дискретность алгоритма означает, что он исполняется по шагам: каждое действие, предусмотренное алгоритмом, исполняется только после того, как закончилось исполнение предыдущего (см. [13], с. 32; [23], с. 18).

Другое свойство принято называть **детерминированностью** (см. [29], с. 10; [2], с. 68). Оно означает, что на каждом шаге однозначно определено преобразование объектов среды исполнителя, полученных на предшествующих шагах алгоритма. В [13], с. 32, и [14], с. 197, это свойство алгоритма названо *точностью*, в [20], с. 30, — *определенностью*[11], а в [18], с. 62, — *однозначностью*. К примеру, в первой из инструкций исполнителю неясно, требуется ли трясти смесь, пока она вся не возьмется комом, и какой все-таки величины маленькая кастрюлька. Так что это наверняка не алгоритм.

Третье свойство — **результативность** алгоритма. Это свойство подразумевает, что каждый шаг (и алгоритм в целом) после своего завершения дает среду, в которой все имеющиеся объекты однозначно определены. Если это по каким-либо причинам невозможно, то алгоритм должен сообщать, что решение задачи не существует. Хотя термин "результативность" взят нами из [13], с. 32, там речь идет только о результативности алгоритма в целом; данное же здесь определение близко к понятию *направленности* алгоритма, определенному в [29], с. 10. В [18] результативность алгоритма определена в целом, а не пошагово и объявлена лишь желательным (а не обязательным) свойством (с. 62—63). В [14] результативность включена в определение алгоритма (оно процитировано выше). Исходя из данного свойства, можно сделать вывод, что вторая из приведенных выше инструкций не является алгоритмом, поскольку в ней не определено, например, когда должно заканчиваться ее исполнение — когда кашель пройдет или лекарство кончится?

Свойство результативности определено в [35], с. 19, так, что оно одновременно подразумевает и **конечность** алгоритма, т.е. завершение его работы за конечное число шагов (при этом количество шагов может быть заранее неизвестным и различным для разных начальных ситуаций). Ту же формулировку мы видим и в [23], с. 18. В [20] и [2] конечность выделена как самостоятельное свойство алгоритма, причем Д.Кнут считает конечность алгоритма его главным свойством. Он, в частности, предлагает процедуру, обладающую всеми свойствами алгоритма, кроме конечности, называть *вычислительным методом*. В [14] конечность тоже выделена как самостоятельное обязательное свойство алгоритма (см. с. 197), хотя оно включено в само определение (отметим еще, что авторы [14] отождествляют конечность и результативность, что, на наш взгляд, неправомерно). Зато в [29] свойство конечности вообще не упоминается, и, по-видимому, в связи с этим результативность алгоритма рассматривается как локальное свойство (т.е. имеющее место на каждом шаге). В [32], с. 203—204, также допускаются как бесконечно работающие алгоритмы, так и алгоритмы, процесс исполнения которых при тех или иных начальных данных может прекратиться безрезультатной остановкой. В школьных учебниках [18] и [23] конечность алгоритма понимается локально (т.е. как завершенность каждого шага; выше мы это свойство назвали дискретностью) и в целом (т.е. как завершимость выполнения алгоритма за конечное число шагов).

Каждый шаг алгоритма обязательно представляет собой какое-либо допустимое действие исполнителя. Это свойство алгоритма в [13], с. 32, 35, [35], с. 20, [14], с. 197, и [23], с. 18, названо *понятностью*, в [20] — *эффективностью*, а в [29] — *элементарностью* шагов алгоритма. Термин "понятность" неявно используют авторы [39] — исполнители, реализованные в программной поддержке, при обращении к ним посредством синтаксически неправильной команды (что интерпретируется как отсутствие соответствующего допустимого действия у исполнителя) в качестве результата своей работы сообщают: "Не понимаю".

[11] Впрочем, не исключено, что это просто перевод английского слова *determinate*.

В рамках обсуждения набора допустимых действий исполнителя естественно поставить вопрос, может ли существовать такой исполнитель, для которого любое действие является допустимым. Такой вопрос предлагается учащимся в учебниках [4], [6] и [40]. Ответ на него отрицателен и, как не покажется удивительным, был дан задолго до того, как появилась теория алгоритмов. Внутренняя противоречивость понятия Всемогущего Исполнителя может быть показана так: если бы такой Всемогущий Исполнитель мог существовать, то тогда он был бы способен создать камень, который он сам поднять не сможет в противоречии с тем, что он все может. Фактически это рассуждение — о внутренней противоречивости исполнителя, который все может, — принадлежит И.Канту как одно из доказательств отсутствия Всемогущего Бога. Важнейший вывод из этого рассуждения состоит в том, что *для любого исполнителя есть задачи, которые с его помощью решить нельзя*. Класс всех задач, которые можно решить, используя данного исполнителя, называют **достижимыми целями** этого исполнителя.

Хотя Всемогущего Исполнителя и не существует, можно ставить вопрос о существовании Универсального Исполнителя. **Универсальным** *называется формальный исполнитель, который может имитировать любого другого формального исполнителя*. Возможность существования такого исполнителя логически недоказуема, хотя и неопровержима. Поэтому существование Универсального Исполнителя постулируется. На самом деле за аксиому берется даже более сильное утверждение — так называемый тезис Черча, одного из основоположников теории алгоритмов. Этот тезис не только говорит о существовании Универсального Исполнителя, но и фактически описывает одного из них. Разумеется, Универсальный Исполнитель может быть не один. Однако из определения Универсального Исполнителя легко получить, что *любые два Универсальных Исполнителя имеют одни и те же достижимые цели*. Поэтому можно считать, что они друг другу **эквивалентны**.

Поскольку каждый исполнитель может решать лишь ограниченный класс задач, то и для Универсального Исполнителя имеются задачи, которые невозможно решить с его использованием. Такие задачи называются **алгоритмически неразрешимыми**. Ясно, что алгоритмически неразрешимая задача не может быть решена никаким формальным исполнителем.

Одним из наиболее ярких примеров алгоритмически неразрешимых задач является задача поиска целочисленных решений произвольного алгебраического уравнения с целыми коэффициентами. Вопрос о существовании алгоритма решения таких уравнений был сформулирован Д.Гильбертом в 1900 году (в списке нерешенных проблем, представленных Математическому конгрессу этим великим математиком, он шел под номером 10, и с тех пор называется "10-я проблема Гильберта"). Лишь в 1970 году наш соотечественник Ю.Матиясевич доказал отсутствие такого алгоритма.

Доказательство, что та или иная задача алгоритмически неразрешима, — как правило, довольно трудная проблема. При этом редко используется именно тот исполнитель, который фигурирует в тезисе Черча. В каждом случае подбирается универсальный исполнитель (или таковой конструируется, т.е. определяется среда и набор его допустимых действий), наиболее подходящий для данной задачи. Таких исполнителей принято называть машинами, и к настоящему времени их наберется около десятка — машина Тьюринга, машина Поста, машина Минского и т.д. Каждая из этих машин несет имя своего изобретателя — одного из корифеев математической логики и теории алгоритмов. Нужно подчеркнуть, что речь вовсе не идет о физической реализации этих машин; данные рассмотрения носят чисто теоретический характер.

В школьных учебниках Универсальные Исполнители фактически не рассматриваются, хотя их дидактическая ценность весьма велика, поскольку они позволяют выпукло продемонстрировать учащимся целый ряд эффектов алгоритмизации. Единственный Универсальный Исполнитель, используемый в учебной литературе, предназначенной для школьников, — машина Поста. Ее описание, предназначенное специально для начального обучения основам программирования, первоначально было разработано в [41], почти в том же виде оно приведено в [15], с. 180—184. Кроме того, в Санкт-Петербурге издана тетрадь на печатной основе "Информатика для 1-го класса", которая также содержит описание машины Поста и задания по ее программированию.

Наконец, рассмотрим еще одно свойство алгоритма — **массовость**. В [13], с. 32, это свойство описывается как возможность с помощью одного и того же алгоритма решать однотипные задачи. В [18] то же самое, только вместо однотипных задач сказано: "целый класс конкретных задач, отвечающих общей постановке задачи" (с. 63). Эти довольно расплывчатые определения (какие задачи считать однотипными? что такое "общая постановка задачи"?) фактически означают, что имеется некоторое множество начальных ситуаций — скажем, для вычислительных алгоритмов это множество допустимых значений входных величин, — которые могут обрабатываться данным алгоритмом. Именно так понимается массовость алгоритма в [29], с. 11, и [20], с. 30. С точки зрения практической ценности алгоритмов важно, чтобы множество допустимых начальных ситуаций было достаточно большим (как сказано в [29], потенциально бесконечным); как правило, практическая ценность алгоритма невелика, если его можно использо-

вать только один раз. Ну перевезли Волка, Козу и Капусту на другой берег, а что еще можно сделать с помощью этого алгоритма?

В учебнике [18] авторы одним из важнейших свойств алгоритма считают его правильность (правда, не ясно, какие еще из свойств алгоритмов авторы намерены были наградить эпитетом "важнейшее"). В тавтологическом духе этого учебника правильным авторы называют алгоритм, выполнение которого дает правильные результаты решения поставленной задачи при любых допустимых значениях исходных данных. Из этого определения авторы выводят, что правильный алгоритм конечен и результативен (хотя, напомним, результативность, по их мнению, всего лишь желательное свойство алгоритма; отсюда следует, что важнейшее свойство — правильность алгоритма — тоже всего лишь желательное, но необязательное). Конечно, каждый, кто составляет алгоритм, надеется, что алгоритм получится правильным, но возводить эту надежду в ранг свойства неправомерно: свойство — это утверждение, истинность которого зависит *только* от того одного объекта, о котором идет речь в утверждении; правильность же алгоритма, в понимании авторов [18], — это утверждение, истинность которого зависит от двух объектов: алгоритма и задачи, для решения которой этот алгоритм предназначается.

Разнобой в названиях для одного и того же свойства связан не только с тем, что теория алгоритмов довольно молодая наука (ей нет еще и пятидесяти). Дело в том, что для нужд практики бывает обременительно требовать от алгоритмов обязательного наличия указанных свойств. Для построения же теории приходится накладывать более жесткие ограничения. Специалисты, работающие в области методов вычислений, хорошо знают аналогичный эффект: при доказательстве того, что метод дает нужный результат, часто формулируются лишь достаточные условия, в то время как на практике метод хорошо работает и при значительно более слабых допущениях.

Да и само понятие алгоритма развивается, освобождаясь от тех или иных ограничений. Теперь, к примеру, рассматривается не только последовательное, но и параллельное исполнение шагов алгоритма (см. выше трактовку свойства дискретности). Алгоритм не обязан быть детерминированным — на очередном шаге может осуществляться случайный выбор из некоторого множества возможных результатов. Теория недетерминированных алгоритмов — современная бурно развивающаяся ветвь общей теории алгоритмов (см., например, [2], пункт 7.1.2.5). При этом многие недетерминированные алгоритмы приводят тем не менее к однозначно определенному результату. Такие алгоритмы называют *однозначными* (см. примечание переводчиков на с. 68 [2]). Вот (шуточный) пример недетерминированного однозначного алгоритма.

Среда: чудо-дерево, на котором растут апельсины и бананы, обладающее следующим свойством — если с него срывают два одинаковых плода, то тут же вырастает один апельсин, а если два разных, то один банан.

Исполнитель: Садовник.

Допустимые действия: сорвать два плода с чудо-дерева; сосчитать количество плодов на чудо-дереве.

```
Алгоритм "Сбор урожая"
Начало
   Начало цикла
     Пока на чудо-дереве более одного плода
       Сорвать два плода
   Конец цикла
Конец
```

Недетерминированность этого алгоритма очевидна — исполнителю не указано, какие именно плоды срывать на каждом шаге, он выбирает их случайным образом. После каждого исполнения тела цикла количество плодов на дереве уменьшается ровно на 1. Поэтому через конечное число шагов на дереве останется 1 плод, и исполнение цикла, а вместе с ним и алгоритма закончится. То, каким будет этот последний плод, однозначно определяется начальной ситуацией: если вначале было нечетное число бананов, то последний плод — банан, если четное, то последний плод — апельсин. Доказать это нетрудно, если заметить, что после каждого срывания двух плодов количество бананов либо не меняется, либо уменьшается на 2. Следовательно, после каждого действия садовника количество бананов имеет ту же четность, что и исходное их количество. А значит, количество бананов по окончании исполнения алгоритма должно иметь ту же четность, что и до его исполнения. Тем самым вид последнего плода на дереве полностью определяется тем, четно или нечетно первоначальное число бананов на чудо-дереве[12].

[12] Для тех, кому не до шуток, мы приведем алгоритм на QBasic, имитирующий алгоритм "Сбор урожая".

```
INPUT "Количество апельсинов"; M
INPUT "Количество бананов"; N
WHILE M + N > 1
  Z = RND(1)
  IF (Z =< 0.66 AND M > 0) OR N < 2 THEN
    M = M - 1
  ENDIF
  IF (Z > 0.66 AND N > 1) OR M < 1 THEN
    M = M + 1
    N = N - 2
  ENDIF
WEND
IF M > N THEN
  PRINT "Остался апельсин"
ELSE
  PRINT "Остался банан"
```

Конечность алгоритма — это тоже свойство, присущее далеко не всем алгоритмам, используемым на практике. Почти все алгоритмы, используемые для управления объектами в режиме реального времени, не являются конечными. Трудно представить себе конечный алгоритм компьютерного управления полетом космической станции или работой ядерного реактора. Да что далеко ходить: включив компьютер, вы тут же запускаете целый комплекс бесконечно работающих алгоритмов — алгоритм работы операционной системы, алгоритм работы видеопроцессора и т.д., — которые исполняются до тех пор, пока компьютер не будет выключен. И если алгоритм работы операционной системы вдруг сам по себе завершил свою работу, то это едва ли способно вызывать радостное настроение у пользователя.

Что касается свойства массовости, то выше мы уже говорили, что оно важно как с практической, так и теоретической точек зрения. Но даже в учебной деятельности школьников встречается немало алгоритмов, этим свойством не обладающих.

Вывод же нужно сделать такой: при всей важности таких свойств, как конечность, массовость, результативность, детерминированность, они не являются обязательными для всех алгоритмов. Тем не менее сама задача, для которой составляется алгоритм, может требовать выполнения для алгоритма того или иного свойства. Например, если в задаче требуется получить в качестве результата некоторое число, то ясно, что алгоритм вычисления этого числа заведомо должен обладать свойством конечности. Поэтому мы скажем несколько слов о том, как можно обосновать конечность алгоритма и/или его результативность.

Исторически первым алгоритмом, для которого специально доказывалась его конечность, был алгоритм Евклида нахождения наибольшего общего делителя двух натуральных чисел. Возможно, в этом кроется причина, что именно он выбирается для иллюстрации способа доказательства конечности алгоритма (и в школьном учебнике [35], с. 18, и в монографии [20], с. 26). Мы приведем несколько иной пример алгоритма, для которого тоже установим его конечность.

```
алг Загадка (арг цел m, n, рез цел x, y)
 дано m > 0 и n > 0 | натуральные числа
 надо               | что?
 нач цел u, v
   x := m; y := n; u := m; v := n
   нц пока x < y или x > y
     если x < y
       то y := y − x; v := v + u
       иначе x := x − y; u := u + v
     все
   кц
   y := (u + v) / 2
кон
```

В условии цикла сказано, что тело цикла будет исполняться до тех пор, пока x и y не станут равными. Однако совершенно неясно, почему они должны стать таковыми в результате выполняемых в цикле операций. Но давайте рассмотрим величину $z = \min\{x, y\}$ и как она меняется в результате исполнения цикла. Очевидно, что пока выполняется условие цикла, ее значение — целое положительное число и остается таковым после каждого исполнения тела цикла. Однако ее значение после исполнения тела цикла заведомо меньше, чем до его исполнения. Бесконечно так продолжаться не может. Значит, через конечное число шагов исполнение цикла должно прекратиться, и алгоритм благополучно завершит свою работу.

Как же мы *доказали*, что алгоритм конечен? Мы построили некоторую функцию от тех переменных, которые участвуют в алгоритме. Эта функция обладает двумя свойствами:

— она принимает лишь конечное число значений;

— значение, которое она принимает при очередном исполнении тела цикла, она не принимала ни при каком из предшествующих исполнений тела цикла (в нашем примере это свойство выполнено в силу того, что значения функции строго убывают).

Функцию, обладающую указанными свойствами, естественно назвать **лимитирующей функцией** — она ограничивает количество исполнений тела цикла (в [33], с. 120, она названа *управляющей величиной*). Конечно, необходимость в специальном построении лимитирующей функции возникает тогда, когда она не извлекается непосредственно из условия окончания (или продолжения) цикла. К примеру, в приведенном выше алгоритме "Сбор урожая" лимитирующая функция очевидна — общее количество плодов на дереве: она убывает на 1 при каждом исполнении тела цикла и проверка, что ее значение больше 1, прямо записано как условие продолжения цикла. Но если бы в жизни всегда было все так очевидно...

Вот еще одна ситуация (идея примера принадлежит одному из классиков программирования Э.Дийкстре; здесь он воспроизводится по [40], с. 57—58).

Библиотекарь, к своему ужасу, обнаружил, что тома полного собрания сочинений Вальтера Скотта (а это больше 20 книг!) стоят в полном беспорядке. Чтобы расположить их по порядку, библиотекарь решил воспользоваться следующим алгоритмом:

```
Алгоритм "Упорядочение"
  Начало цикла
    Пока есть два соседних тома,
    стоящие в неправильном порядке
        Поменять эти тома местами
  Конец цикла
Конец алгоритма
```

Библиотекарь при этом не придерживается какого-либо правила поиска перепутанных томов (так что перед нами еще один недетерминированный алгоритм).

Здесь, наверно, не так уж очевидно, что этот алгоритм конечен. Не будет ли библиотекарь обречен на сизифов труд? (Миф о Сизифе является, по-видимому, первым упоминанием о никогда не кончающихся циклических алгоритмах).

Вот доказательство конечности этого алгоритма. Заменим каждый том гирей, вес которой равен номеру заменяемого тома. Воспринимая такое "собрание сочинений" как единое целое, можно говорить о его центре тяжести. Легко понять, что, меняя местами более тяжелую гирю с более легкой, мы смещаем центр тяжести совокупности гирь вправо. Однако ясно, что число всех возможных расположений центров тяжести системы гирь конечно — не больше, чем различных перестановок этих гирь. Значит, неограниченно долго смещаться вправо центр тяжести не может, и потому наступит момент, когда перестановки соседних гирь (томов) закончатся. Но это означает, что нарушится условие продолжения цикла, т.е. все тома окажутся расположенными по порядку. Кстати, заодно мы доказали, что этот алгоритм однозначен.

Если внимательно присмотреться, то и здесь видна та же идея лимитирующей функции. Только математическая запись этой функции выглядит не так уж просто. Чтобы ее получить, нужно:

- ввести координатную ось;
- положение каждой гири описать координатой центра тяжести;
- вычислить координаты центра тяжести совокупности гирь (это и будет значение лимитирующей функции);
- проверить, что функция возрастает при каждом исполнении тела цикла.

Как видно, поиск лимитирующей функции может оказаться весьма непростой задачей. Нельзя ли указать какой-нибудь универсальный метод, позволяющий по предъявленному алгоритму автоматически решать вопрос, конечен он или нет? Увы, нет такого метода! Задача определения конечности алгоритма алгоритмически неразрешима (см., например, [33], с. 155).

Второй метод анализа алгоритмов, который мы обсудим, состоит в нахождении для алгоритма или какого-либо его фрагмента подходящего **инварианта**. Всем хорошо знакомы слова того же корня: "вариант", "вариация", "варьировать" и т.п., означающие изменение чего-либо. Приставка *ин-*, как обычно, соответствует русской приставке *не-*. Так что слово "инвариант" означает неизменность. В школьных учебниках метод инварианта рассматривается в [25] и [40]; методические рекомендации по изучению этого метода с учащимися учитель может найти в [8].

Общая идея довольно проста: каким бы изощренным ни был алгоритм, он не может за конечное число шагов изменить все свойства объектов среды, в которой действует исполнитель, и все отношения, существующие между ними. Такие неизменные свойства объектов или неменяющиеся отношения между ними и называются инвариантами.

В качестве примера снова рассмотрим алгоритм "Загадка", в котором пока неясно, что же надо найти[13]. Один из инвариантов обнаружить нетрудно. Поскольку у пар чисел (x, y), $(x, y - x)$, $(x - y, y)$ наибольший общий делитель один и тот же, то сколько бы раз цикл ни исполнялся, величина НОД(x, y) остается неизменной. Тем самым функция $f(x, y) =$ $=$ НОД(x, y) — инвариант цикла. Другой инвариант, зависящий от четырех переменных, найти несколько сложнее. Им является, например, функция $g(x, y, u, v) = xv + yu$. Действительно, если $x < y$, то после того, как в очередной раз исполнение тела цикла закончилось, новое значение функции g подсчитывается по формуле $g(x, y - x, u, v + u) =$ $= x(u + v) + (y - x)u = xv + yu = g(x, y, u, v)$; если же $x > y$, то значение функции g после очередного выполнения тела цикла подсчитывается по формуле

$$g(x - y, y, u + v, v) = (x - y)v + y(u + v) = xv + yu = g(x, y, u, v).$$

Значит, функция $g(x, y, u, v)$ тоже инвариант.

Чем же стало лучше от того, что мы построили два инварианта цикла? Можно и еще построить сколько угодно. Например, сложить функции f и g. Очевидно, снова получится инвариант. Кому нужна такая прорва инвариантов?

И действительно, инвариантов всегда можно построить бесконечно много. А вот какой окажется полезен — это непростой вопрос.

Используют же инвариант обычно так: вычисляют его значение до самого первого исполнения тела цикла, а затем после последнего его исполнения. Скажем, для инварианта $f(x, y)$ до исполнения цикла его значение НОД(m, n). По окончании цикла из условия продолжения цикла следует, что $x = y$. Поэтому значение $f(x, y) = f(x, x) =$ НОД$(x, x) = x$. Но по окончании цикла x служит одним из

[13] В [25] на с. 140 приведен фактически тот же алгоритм, но с указанием, что именно получается в результате его исполнения, и с заданием доказать это с использованием понятия инварианта цикла. Знание того, что надо доказать, конечно, облегчает решение задачи, но едва ли способствует угадыванию, какой именно инвариант здесь оказывается полезным. Как ни покажется странным, одновременное рассмотрение двух результатов, как это предложено в алгоритме "Загадка", оказывается проще, чем одного.

результатов работы алгоритма. Следовательно, один из результатов работы алгоритма — наибольший общий делитель его аргументов. И мы доказали, что именно таков результат работы алгоритма.

Попробуем теперь с помощью инвариантов определить второй результат. Для этого построим инвариант

$$h(x, y, u, v) = g(x, y, u, v) \ / \ (2f(x, y)).$$

До начала исполнения цикла

$$h(x, y, u, v) = h(m, n, m, n) =$$
$$= g(m, n, m, n) \ / \ (2f(m, n)) = mn \ / \ \text{НОД}(m, n).$$

После завершения исполнения цикла

$$h(x, y, u, v) = h(x, x, u, v) =$$
$$= g(x, x, u, v) \ / \ (2f(x, x)) = x(u + v) \ / \ (2x) =$$
$$= (u + v) \ / \ 2.$$

Тем самым инвариант $h(x, y, u, v)$ по окончании исполнения цикла дает нам второй результат работы алгоритма. Но через аргументы этот инвариант выражается как $mn \ / \ \text{НОД}(m, n)$. В общем-то это уже ответ на вопрос, что должно быть написано после слова **надо**. Если же обратиться к математикам, то они тут же скажут, что произведение наибольшего общего делителя двух чисел на их наименьшее общее кратное равно произведению исходных чисел. Поэтому второй результат алгоритма — это наименьшее общее кратное его аргументов.

Такое использование инварианта — доказательное выявление того, что является результатом работы алгоритма — может показаться счастливым случаем. Однако теоретики современного программирования (Э.Дийкстра в книге "Дисциплина программирования", Д.Грис в книге "Наука программирования" и другие) высказывают аргументированную точку зрения, что всегда имеется инвариант, описывающий результат работы цикла. Более того, по их мнению, стратегия конструирования цикла состоит в изначальном определении инварианта и лимитирующей функции, а затем в относительно формальном построении тела цикла, который имеет данный инвариант и данную лимитирующую функцию (см. [9], глава 15). Если инвариант и лимитирующая функция построены правильно, то в алгоритме не будет ошибок алгоритмизации. О чем еще мечтать любому программисту! Возможно, поэтому авторы учебника [18] сочли необходимым обсуждать со школьниками вопросы *доказательства* правильности алгоритмов. Сразу скажем, что любое доказательство в рамках формализованной системы (напомним, что алгоритм *всегда* записывается на формализованном языке, и потому все, что мы о нем хотим сказать, также должно быть формализованным) является частью математической логики — предмета, совершенно иного, нежели информатика. Основы доказательного программирования изложены в упомянутой выше книге [9] "Наука программирования"[14]. В ней автор прямо указывает, что языком этой науки является язык математической логики[15]. Поэтому нам представляется абсолютно верным невключение в "Обязательный минимум..." вопросов доказательного программирования.

В эту благостную бочку меда теоретической алгоритмизации приходится добавить ложку дегтя практического программирования. Рассмотрим два алгоритма:

```
алг Сумма 1
нач вещ S; цел I
  S := 0; I := 1
 нц пока S < 1000
   S := S + 1 / I
   I := I + 1
 кц
 вывод (I)
кон

алг Сумма 2
нач вещ S; цел I
  S := 0; I := 1
 нц пока S < 1000
   S := S + 1 / I ^ 2
   I := I + 1
 кц
 вывод (I)
кон
```

Математик, взглянув на алгоритм "Сумма 2", немедленно объявит, что алгоритм "зациклится" — нетрудно доказать, что сколько бы слагаемых мы ни взяли, сумма S не превзойдет числа 2. Поэтому условие продолжения цикла всегда окажется выполненным.

Иное дело алгоритм "Сумма 1". Теоретически работа этого алгоритма обязательно закончится, ибо сумма

$$1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

[14] В предисловии к своей книге Д.Грис так объясняет названи своей книги:

"Однако иногда термин "наука" понимается расширительно так что в него включаются области практической деятельности, в кото рых требуется знание общих принципов и их осознанное примене ние; в противоположность этому в искусстве требуется лишь знани традиционных правил и достаточно высокое мастерство.

Именно в этом контексте и выбрано название настоящей книги"

Нам представляется, что этим Д.Грис обозначил границу межд принципами программирования, которым посвящена его книга, традиционным подходом к созданию программ, который энцикло педически изложен в серии книг Д.Кнута, названной "Искусств программирования для ЭВМ". Весьма многозначительна такая пер кличка в названиях книг!

[15] В предисловии редактора перевода книги Д.Гриса "Наука пр граммирования" академик А.П. Ершов пишет: "В первых ее глав показывается, что логика становится прикладной наукой, чья нот ция, обогащенная символикой языков программирования, становит такой же неотъемлемой частью человеческой интеллектуальной пра тики, как и нотация математического анализа. В главной же сво части книга претендует на то, чтобы внушить учащемуся убежден в *невозможности* (курсив А.П. Ершова) программировать инач нежели по дисциплине, подсказываемой логическими правилам синтеза программ".

с ростом n становится как угодно большой и, значит, при каком-то n превысит 1000. Знаток математики, правда, добавит, что произойдет это, когда в записи n будет не менее четырехсот цифр. Но такое число не умещается в разрядную сетку компьютера! Поэтому вместо значения I компьютер сообщит об отказе ввиду переполнения.

Проконсультировавшись с математиками, изменим алгоритм "Сумма 1" на следующий:

```
алг Сумма 1М
нач вещ S; цел I
  S := 0; I := 1
 нц пока S < 100
   S := S + 1 / I
   I := I + 1
 кц
 вывод (I)
кон
```

Математики уверяют, что при таком ограничении на S величина I не превзойдет 10^9 — такое число вполне по силам компьютеру. Тем не менее при запуске соответствующей программы результат все равно не будет получен, на этот раз алгоритм "зациклится". Если для контроля внутри цикла печатать значение S, то выяснится, что S никак не может преодолеть некий барьер — число 15. Дело тут опять в машинной арифметике: когда мы к уже имеющемуся S пытаемся добавить $1/I$ при $I > 10^8$, то его прибавление никак не сказывается на величине S[16]. Тем самым из этого рассмотрения следует такой вывод: *свойства алгоритмов нельзя переносить на программы, не учитывая особенностей исполнителя, который эти программы будет проводить в жизнь*[17].

§ 24. Способы записей алгоритмов

Основных способов записи алгоритмов два: в виде текста, состоящего из команд исполнителю, и в виде блок-схемы.

Текст пишется, как уже было сказано, на формализованном языке, описывающем того исполнителя, который предполагается к использованию для решения данной задачи. Кроме того, в грамматику этого языка входят правила оформления текста алгоритма, цель которых состоит прежде всего в том, чтобы исключить неоднозначное толкование организации последовательности действий. Прежде всего это относится к средствам обособления тела ветвления, тела цикла и тела вспомогательного алгоритма. В некоторых случаях это аббревиатуры, обозначающие начало и конец соответствующей конструкции (как, например, в [25] — **нц** и **кц**, **все** и т.д., в [6] и [7] — операторные скобки). Нередко текст алгоритма пишется сразу на языке программирования целиком или фрагментарно. В этом случае возникают дополнительные синтаксические ограничения на запись алгоритма. Скажем, в учебнике [25] целый пункт 4.4 (с. 26—27) отведен под описание жестко фиксированной записи алгоритма, которую школьники обязаны заучить.

Приведем пример записей на разных языках одного и того же алгоритма, предназначенного для решения следующей задачи:

Последовательность X_n строится так:

$$X_1 = 1;\ X_2 = 3;\ \ldots;\ X_n = X_{n-1} - 2X_{n-2}$$

для каждого $n > 2$. Составьте алгоритм подсчета количества положительных чисел в этой последовательности среди первых 10 000 *ее членов.*

1. На школьном алгоритмическом языке (см. [25]):

```
алг Задача
 дано | рекуррентно заданная последователь-
      |ность X_n = X_{n-1} - 2X_{n-2} с
      | первыми членами X_1 = 1; X_2 = 3.
 надо | напечатать количество положительных
      | членов последовательности среди
      | первых 10 000 ее членов.
 нач цел X, Y, Z, N, K
  X := 1
  Y := 3
  N := 2
  нц для K от 3 до 10 000
   Z := X - 2 * Y
   X := Y
   Y := Z
   если Z > 0
    то N := N + 1
   все
  кц
  вывод N
кон
```

Зарезервированные слова выделены жирным шрифтом и подчеркнуты.

---

[16] Мы экспериментировали на компьютере, в котором для хранения вещественного числа в формате IEEE-754 (*Institute of Electrical and Electronics Engineers*) выделяется 4 байта. Имеется множество различных форматов представления вещественных чисел. Для другого формата может потребоваться изменить конкретные значения в нашем примере, но существо дела не меняется.

[17] Рассмотрение этого вопроса, адаптированное к уровню школьников, имеется в [40], с. 60.

2. На языке, используемом в [6] и [7]:

```
Программа
цел: А, Б, В, Номер, Счетчик;
{ А := 1;
  Б := 3;
  Номер := 2;
  Делать от Счетчик := 3 до 10000
  { В := А – 2 * Б;
    А := Б;
    Б := В;
    Если (В > 0)
      то { Номер := Номер + 1;
         }
  }
  Сообщить Номер;
}
```

Зарезервированные слова подчеркнуты.

3. На языке QBasic:

```
X = 1
Y = 3
N = 2
FOR K = 3 TO 10000
  Z = X – 2 * Y
  X = Y
  Y = Z
  IF Z > 0 THEN N = N + 1
NEXT K
PRINT N
```

Зарезервированные слова никак не выделяются, их надо помнить.

4. На языке Паскаль:

```
PROGRAM Zadacha;
VAR X, Y, Z, N, K : INTEGER;
BEGIN
  X := 1;
  Y := 3;
  N := 2;
  FOR K := 3 TO 10000 DO
  BEGIN
    Z = X – 2 * Y;
    X = Y;
    Y = Z;
    IF Z > 0 THEN N := N + 1
  END;
  WRITELN(N)
END.
```

Зарезервированные слова никак не выделяются, их надо помнить.

Запись алгоритма блок-схемой представляет собой визуализированную форму организации действий в алгоритме. Если для текстового способа записи алгоритмов каждой стандартной алгоритмической конструкции соответствует зарезервированный текстовый элемент, обозначающий данную конструкцию и соответствующий ей оператор, то в блок-схемах алгоритмические конструкции и стандартные операторы представляются геометрическими средствами. Так, действие исполнителя записывают в прямоугольник, проверяемое условие — в ромб и т.п. (перечисление графических элементов, применяемых в блок-схемах, можно найти, например, в [14], с. 235), а переходы между этими элементами указываются линиями (со стрелками или без), которые могут быть помечены результатами проверки условий. Вот как представляется алгоритм решения выше сформулированной задачи блок-схемой:

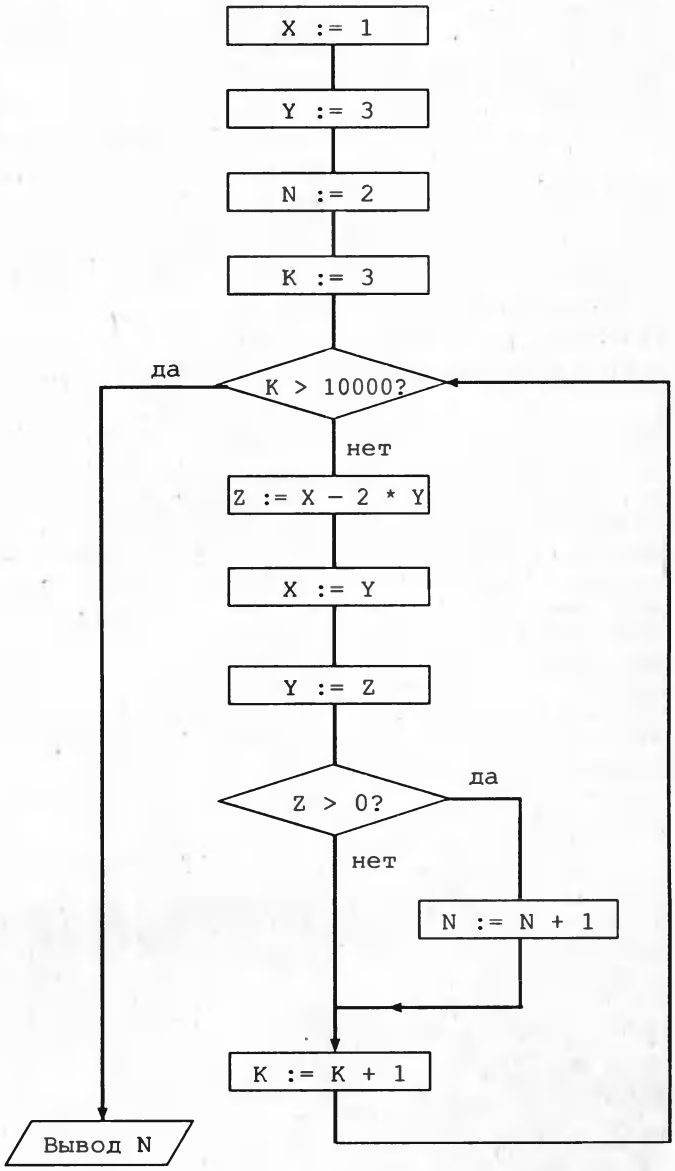


Рис. 64. Блок-схема

Отметим, что договариваются в блок-схемах ставить стрелки только на тех линиях, которые выходят влево или вверх.

Долгое время блок-схемы оставались исключительно средством записи алгоритмов во время их разработки. В настоящее время появились трансляторы с языка блок-схем. Теперь программист может на экране компьютера просто начертить

блок-схему алгоритма, а соответствующая программа переведет ее либо в машинные коды, либо в программу, записанную на каком-либо языке программирования высокого уровня. На сегодняшний день имеются две такие системы программирования с русскоязычным интерфейсом: "Пифагор" и "Дракон". Для работы в этих системах, как и в любой системе программирования, приходится жестко, можно сказать, драконовскими мерами формализовать использование графических средств описания алгоритма[18].

Итак, в учебной литературе для школьников отчетливо просматриваются две позиции:

— запись алгоритма производится на обычном русском языке с соблюдением требований формализации (т.е. однозначности смысла и значения всех слов) и договоренностей по оформлению алгоритмических конструкций (неважно, в словесно-символьной форме или в виде блок-схемы);

— запись алгоритма производится на языке программирования (опять-таки неважно, будет это текст или блок-схема на соответствующем формальном языке), ориентированном на последующее применение конкретного транслятора.

Сторонники первого подхода убеждены, что алгоритмы составляются человеком для человека (для самого себя или кого-то другого) с тем, чтобы было легко понимать его смысл, а значит, и легко исправлять ошибки алгоритмизации. При таком подходе поиск синтаксических ошибок — это отдельный процесс, связанный с переводом алгоритма на формальный язык того формального исполнителя, которым намерен в дальнейшем воспользоваться составитель алгоритма для решения задачи. Здесь налицо разделение творческого и рутинного компонентов умственной деятельности человека.

Сторонники второго подхода считают, что естественный язык никогда не может быть избавлен от неоднозначности, поэтому строгость алгоритмического мышления может быть воспитана только кнутом формального языка записи алгоритмов[19]. Их не смущает, что человек при этом становится интеллектуальным придатком компьютера, поскольку главной целью алгоритмизации становится не вообще видение алгоритмических аспектов в реальной жизни, а сугубо программное обслуживание неодушевленного объекта с заведомо ограниченными возможностями. В этом случае обучение алгоритмизации становится, на наш взгляд, учебной деятельностью уже не общеобразовательного, а профессионального направления.

Продолжение следует.
Окончание главы 5 читайте в следующем номере.

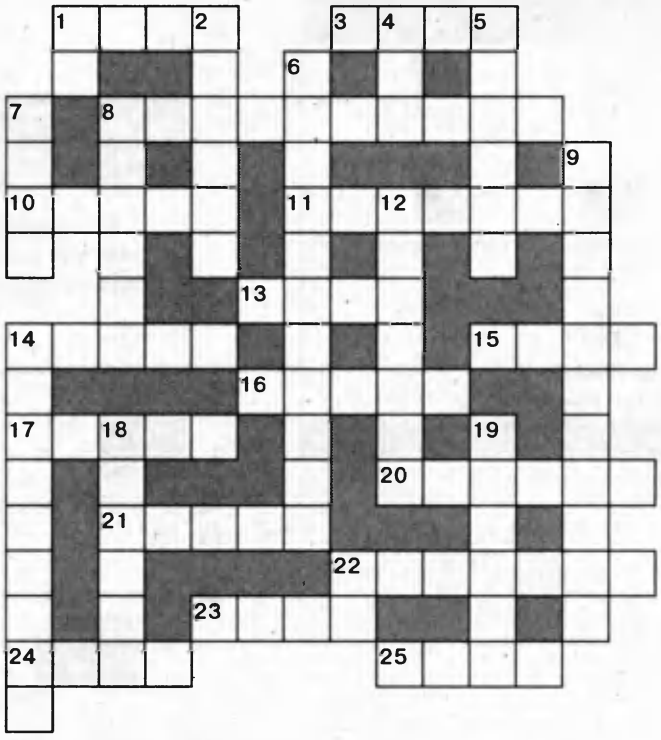
[18] Познакомиться с системой "Дракон" можно по книге [37]. Мы, однако, не рискнем рекомендовать эту книгу для изучения алгоритмизации хотя бы уже потому, что многие определения, приведенные в этой книге, на самом деле таковыми не являются. Например, цикл ПОКА определен как цикл, который может ни разу не выполняться, либо выполняться один раз, либо много раз (см. с. 98). Однако таким же свойством обладает гибридный (в терминологии автора книги) цикл (с. 100). Отличать же циклы рекомендуется по внешнему виду, т.е. сначала нарисуй блок-схему (как говорит автор, дракон-схему) и по ее виду определи, каким циклом воспользовался. Это довольно странная методическая новация: до сих пор считалось, что школьника надо научить сначала принять решение, какой алгоритмической конструкцией в данной ситуации надо воспользоваться, а уже затем рисовать блок-схему.

[19] Странно при этом выглядит сентенция авторов [1], что "хорошо оформленный алгоритм должен читаться как литературное произведение". Неужели наша литература заслужила такой оценки своих художественных возможностей?

Кроссворд с фрагментами

Д.М. Златопольский,
Москва

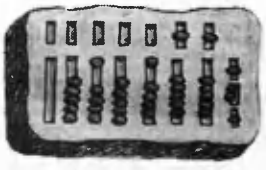
В приведенном кроссворде записанные в нем слова "зашифрованы" рисунками, таблицами, поясняющими текстами.



Вопросы к кроссворду

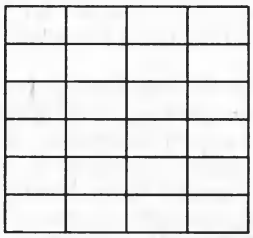
По горизонтали:

1. +

3. 

8. 

10. — — · · — — · —

11. 

13. "Сапер", "Тетрис", "Duna".

14. fio@provider.ru

15.
```
нц для i от 1 до 20
| вывод нс, "Привет всем!"
кц
```

```
for i := 1 to 20 do
  writeln ('Привет всем!');
```

```
FOR i = 1 TO 20
  PRINT "Привет всем!"
NEXT I
```

16. Shift — "шифт",
Alt — "альт",
Pause — ...

17. 

20. 274571

21. ω

22. 

23. 

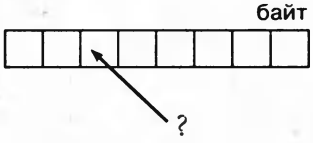
24. 0

25. —, <, ?, :

По вертикали:

1. 3,1415926

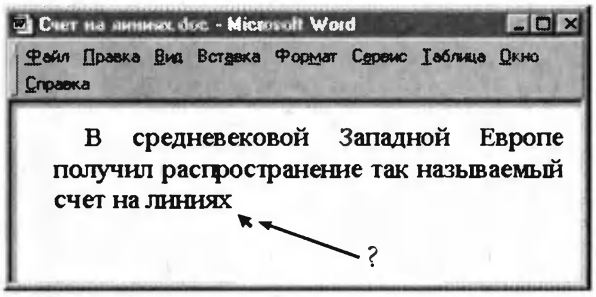
2. 

4. 

5. *Перфокарта* — носитель информации в компьютерах первых поколений.

6. 

7. VII

8. 

9. 

12. Microsoft Internet Explorer, Netscape Navigator.

14. &

18. 

19. Каталог — Папка,
Монитор — Дисплей,
Алфавит — ...

22. ρ

Ответы.

По горизонтали: 1. Плюс. 3. Абак. 8. Клавиатура. 10. Морзе. 11. Таблица. 13. Игра. 14. Адрес. 15. Цикл. 16. "Пауза". 17. Папка. 20. Разряд. 21. Омега. 22. Рисунок. 23. Окно. 24. Ноль. 25. Знак.

По вертикали: 1. Пи. 2. Сканер. 4. Бит. 5. Курсив. 6. Гистограмма. 7. Семь. 8. Курсор. 9. Калькулятор. 12. Браузер. 14. Амперсанд. 18. "Пробел". 19. Азбука. 22. Ро.

Калейдоскоп

Интернет в помощь

Исследование показало, что более 2/3 американских учащихся используют Сеть для выполнения больших проектов: сочинение, исследование, реферат. Большинство подростков говорят, что могут найти в Сети все, что необходимо для выполнения домашних заданий.

Отношение же взрослых американцев к Интернет-навыкам собственных детей весьма неоднозначно. Опрос, проведенный Ассошиэйтед Пресс, показал, что половина респондентов считает такие способности важными, а другая половина особой важности им не придает или вообще считает неважными.

Мнения преподавателей по поводу важности Интернета также разделились: часть опасается, что благодаря Интернету студенты отучаются искать, анализировать и перерабатывать информацию. Кроме того, остро стоит вопрос о плагиате. Другие же, наоборот, поощряют Интернет-навыки детей.

Что касается возрастных категорий, то чем старше был респондент, тем меньше доверия выказывал он к использованию Интернета. А жители больших городов были более благожелательно настроены, чем те, кто проживает в сельских районах.

По материалам сайта www.ap.com

Вопросы, задания и контрольные работы для начинающих программистов

В.П. Гладков, А.П. Шестаков,
г. Пермь

Продолжение. См. № 20, 33, 34, 35, 37/2001

6. Программы на Паскале. Ввод, вывод, присваивание

6.1. Что такое "комментарий"? Как он записывается и для чего используется?

6.2. Правильно ли употреблены комментарии в следующем фрагменте программы?

```
{Этот фрагмент Паскаль-программы содержит
 очень много к о м м е н т а р и е в}
 a{переменная левой части} := a{выражение} +
                              b{правой} - 16{части};
writeln(a,{разделяем выражения}b)
```

6.3. Как задаются правила построения различных конструкций Паскаля?

6.4. Из каких частей состоит программа на Паскале?

6.5. Всегда ли в Паскаль-программе имеются описания?

6.6. Для чего описывают константы?

6.7. Для чего описывают типы?

6.8. Для чего описывают переменные?

6.9. Верно ли, что в Паскаль-программе нужно описывать все переменные?

6.10. Какую информацию извлекает транслятор из описания переменных? Как он ее использует?

6.11. Какие значения имеют переменные в начале выполнения программы?

6.12. Можно ли менять значения констант в процессе выполнения программы?

6.13. В описании нетипизированных констант их типы не указываются. Как определяет типы констант транслятор?

6.14. Почему не надо описывать константы **maxint**, **true**, **false**?

6.15. Для чего описывают процедуры и функции?

6.16. Как описываются константы, типы, переменные, процедуры и функции?

6.17. Найдите ошибки в программе:

```
programm 101;
{ эта программа печатает число недель в году {
var s; integer:
begon
  s := 56;
  write(В году s недель);
end
```

6.18. Найдите ошибки в программе:

```
program 102;
constanta dwa = 2;
tri = dwa + 1;
пять = tri + dwa;
var info, comma : integer,
tigr, lev, volk : real;
serg : olga;
type olga = real;
begin readln(serg);
      write('Введите имя ');
      writeln(serg = olga)
end.
```

6.19. Что будет напечатано после выполнения программы, если ввести с клавиатуры числа 1, 2, 3, 4, 5?

```
program abba;
var a, b, c : integer;
begin read(a,b,a,c,b);
  write(a,b,c)
end.
```

6.20. Что будет напечатано в результате выполнения программы?

```
program print;
begin write(1); write(2,3); writeln(4);
      write(5); writeln(6,7); writeln;
      write(8)
end.
```

6.21. Что будет напечатано?

```
write(1); writeln(2,3); write(4);
writeln(5,6); write(7); writeln(8,9);
```

6.22. Как изменится результат, если в приведенном фрагменте поменять местами второй и третий операторы присваивания?

```
a := b;
b := c;
c := a.
```

6.23. Как изменится результат, если в приведенном фрагменте третий оператор присваивания поставить на первое место?

```
a := b;
b := c;
c := a.
```

6.24. Есть ли среди записей `a := a + 1` и `a = a + 1` ошибочная?

6.25. Как напечатать фразу: "Хотите купить утку?"?

6.26. Каков будет результат работы программы?

```
program wr(input,output);
var i : integer;
begin  i := maxint;
       writeln(i,' ',i + 1,' ',i + 2);
       readln
end.
```

6.27. В программе используется оператор `i := i + 33000` где **var** `i : integer`. Ошибочен ли этот оператор?

6.28. Исправьте ошибки в программе:

```
proqram temp;
 const  tri = 3,
        dba = 2,
        odin = 1;
```

```
var    g; h : real;
       tax,rate : real;
end
g = e21;
tax = rate * g;
begin.
```

6.29. Исправьте ошибки в программе:

```
Program Ой-ей-ей(input,output);
const b = Ай-яй-яй;
      x = 10;
var age : integer;
    name : string;
begin write("Введите, пожалуйста, свое имя ');
  readln(name);
  write("Прекрасно, ",name,",
        сколько Вам лет? ";
  readln(age);
  xp := age + x;
  write(name,"! Вам должно быть
        по крайней мере ",xp," лет")
edn.
```

6.30. Пусть x = 2, y = 3. Напишите оператор печати фразы "Сумма 2 + 3 = 5", используя значения переменных x и y.

6.31. Известно, что a **mod** b не равно нулю. Какое число нужно добавить к переменной a, чтобы она стала кратной b?

6.32. Даны значения переменных: a = 1, b = 5. Какими будут их значения после выполнения последовательности операторов a := b; b := a?

6.33. Какие значения получат переменные x и y после выполнения последовательности операторов?

```
а) x := 15 div (8 mod 3);
   y := 17 mod x * 5 - 19 mod 5 * 2;
б) x := 2 * 5 div 3 mod 2;
   y := 2 * 5 div (3 mod 2);
   x := x * y,
   y := y * y.
```

6.34. Значение переменной a — трехзначное натуральное число. Результат — двузначное число b, которое получается из a вычеркиванием средней цифры. Напишите операторы для получения числа b.

6.35. Укажите ошибки в записи операторов ввода:

```
а) read(a,b,ab);
б) rEaD(i)4;
в) read(begin);
г) read{возможно ли такое?}(a,b);
д) READ(a{а такое возможно?},b);
е) read({может быть, можно так?});
ж) re{или так?}ad(a,b);
з) read((a,b));
и) read((a),b);
к) read(a,a,a);
л) read(a,1,b);
м) read('Введите число ',a);
н) read( a,
         b,
         c)
о) r
   e
   a
   d
   (
   a
   ,
   b
   ).
```

6.36. Что будет напечатано в результате выполнения следующей программы, если ввести числа 1, 2, 3?

```
program test2;
{что же получится?}
var a, b, c : integer;
begin   write('Введите три целых числа ');
   readln(a,b,a);
   c := a + b;
   write('a + b = ',c)
end.
```

Укажите оператор, который можно убрать из программы без изменения результата ее работы.

Укажите минимальное количество изменений в программе, достаточное для того, чтобы результат ее работы изменился.

6.37. Запишите самый короткий оператор Паскаля.

6.38. Запишите самую короткую программу на Паскале.

6.39. Найдите формулу, вычисляемую программой:

```
program exampl;
begin a, b, c : integer; {коэффициенты}
   d, e : integer; {промежуточные переменные}
   x : integer; {аргумент}
   r : integer; {результат}
begin write('Введите коэффициенты ');
   readln(a,b,c);
   write('Введите значение аргумента ');
   readln(x);
   d := a * sqr(x);
   e := b * x;     {*}
   r := d + e + c;
   write('результат=',r:5)
end.
```

Что будет получено, если в операторе, отмеченном звездочкой, вместо операции умножения (*) использовать операцию деления (/)?

6.40. Что будет напечатано в результате работы программы:

```
program ex;
var a, b, c, d : integer;
    {исходные данные}
    a1, b1, c1, d1 : integer;
    {копии исходных данных}
    r : integer;
    {для временного хранения значения}
begin write('Введите исходные данные:
            четыре целых числа ');
  readln(a,b,c,d);
  a1 := a; b1 := b; c1 := c; d1 := d;
  r := a;
  a := b;
  b := c;
  c := d;
  d := r;
```

```
  a1 := a1 + b1;
  b1 := a1 - b1;
  a1 := a1 - b1;
  b1 := b1 + c1;
  c1 := b1 - c1;
  b1 := b1 - c1;
  c1 := c1 + d1;
  d1 := c1 - d1;
  c1 := c1 - d1;
  write((a = a1) and (b = b1) and
        (c = c1) and (d = d1))
end.
```

6.41. Что будет напечатано следующей программой при вводе чисел 5, 4, 3?

```
program primer;{schlecht arbait} var
fir:integer;funf,a,b:real;begin
read(a,fir,b);funf:=5*(a+b)-
fir;fir:=a*funf;write(funf, ' ',fir);{какова
программка?вам нравится?}b:=sqr(abs(funf-
fir)*sin(a)-3);writeln(a+b-a*b+(funf-fir));;
a:=abs(b-a+funf-fir)*a+b-fir*funf-sin(a+cos(b-
sqrt(funf)+2));writeln('результат=',a, ' ',b,
' ',funf, ' ',fir, ' ',a+b, ' ',funf+fir, '
',a-b-fur-fir);;;end.
```

6.42. Найдите ошибки в каждой из следующих программ.

а)
```
program A;
  const d = 5;
  begin d := sqr(d);
        writeln('d**2=',d
  end.
```

б)
```
program B;
  const k = true;
  var x : real;
  begin read(x);
        writeln(ord(x) = k)
  end.
```

в)
```
program C;
  var a, b, c : integer;
  begin read(a,b);
        writeln(( a + b + c) / 3)
  end.
```

г)
```
program D;
  var x : real;
  begin read(x);
        y := sqr(x) + 1;
        writeln(y)
  end.
```

д)
```
program E;
  const B := 2.5;
  var a, b, c : real;
  begin read(a,c);
        writeln(a * c > b)
  end.
```

е)
```
program F;
  var a, b : integer;
  begin
        read(A);
        d := odd(pi * 0) and b > a;
        writeln(d)
  end.
```

ж)
```
program G;
  var a, b : integer;
    r : integer;
  begin readln(a,b);
        r := a / b;
        write(r)
  end.
```

з)
```
program H;
  var a, b : real;
     r, p : integer;
  begin readln(a,b);
        r := a div b;
        p := a mod b;
        write(r, ' ',p)
  end.
```

и)
```
program I;
  var a, b : integer;
     r : real;
  begin readln(a,b);
        r := a div b;
        write(r)
  end.
```

6.43. Каким образом можно вводить значения логических переменных?

6.44. Задача о расчетливой Дорис. На вопрос, пойдет ли она на вечеринку, Дорис ответила: "Мне нужен кавалер на этот вечер. Хорошо, если там окажется Боб или Карл. Только не оба сразу, я не хочу оказаться между двух огней. Впрочем, если там будет Анна, то Боб на меня и не посмотрит".

Ниже приводится программа, которая, по мнению авторов, позволяет определить, пойдет ли Дорис на вечеринку, если туда пойдут ее друзья. Известно, что если кто-то идет на вечеринку, то для него вводится символ T, в противном случае — F. Установите, совпадет ли ответ программы с мнением Дорис. Какие претензии можно высказать по поводу этой программы?

```
program Doris;
{задача о расчетливой Дорис}
var A{nna}, B{ob}, C{arl}, D{oris}: boolean;
    s : string;
begin write('Пойдет ли Анна на вечеринку? ');
  readln(s);
  a := s[1] = 'T';
  write('Пойдет ли Боб на вечеринку? ');
  readln(s);
  b := s[1] = 'T';
  write('Пойдет ли Карл на вечеринку? ');
  readln(s);
  c := s[1] = 'T';
  d := not a and b xor c;
  write('Пойдет ли Дорис на вечеринку? ',d:5,'!')
end.
```

6.45. Что будет напечатано следующими операторами?

```
  d := true;
  writeln('1) ', d);
  writeln('2) ', d = false);
  writeln('3) ', d < true);
  writeln('4) ', not d or d);
  writeln('5) ', not d > false);
  writeln('6) ', not (d and not d) = d or not d).
```

6.46. Найдите ошибки в программах:

1)
```
i : integer;
begin write('задайте целое число ');
      readln(i);
      write(i)
end;
```

2)
```
var i : real;
begin i := 1;
      writeln('i=',j)
end.
```

6.47. Всегда ли выполнение последовательностей операторов

```
read(x,y); z := sqr(x) + 1;
write(x / z,' ',y / z,' ',x)
```
и
```
read(x,y);
write(x / sqr(x) + 1,' ',y / sqr(y) + 1,' ',x)
```
при одних и тех же исходных данных приведет к выводу одних и тех же чисел?

6.48. Какие скобки в приведенных выражениях можно снять, не изменив значения этих выражений? Предложите алгоритм проверки правильности расстановки скобок в выражении.

```
((a + b) / c)
((a + (b / c)) - ((2 * d) / (2 + d)))
not ( not (a < b) or (c >= d))
((a + b) - (((c + d) + (2 * e))))
((a + b) + c) + ((c + d) * (2 * e))
(((a - b) - c) - d)
(a - (b - (c - d)))
```

6.49. В каком порядке и какие числа должны вводиться, если после выполнения операторов Паскаля было напечатано: 1, 2, 3 ?

```
read(a, b, c, a, b);
write(c, b, a);
```

6.50. Сколько строк выходных данных будет напечатано следующим фрагментом программы?

```
x := 1;
while x <= 5 do
begin y := 10;
  while y > 2 do
  begin writeln(x * y);
    y := y - 0.5
  end;
  x := x + 0.5
end;
```

6.51. Сколько строк выходных данных будет напечатано следующим фрагментом программы?

```
x := 1;
while x <= 5 do
begin y := 20;
  while y >= 7 do
  begin writeln(x * y);
    y := y - 1
  end;
  x := x + 0.5
end;
```

6.52. Какую задачу решает приведенный фрагмент?

```
q := 0; r := x;
while r >= y do
begin r := r - y;
  q := 1 + q
end;
```

6.53. Какую задачу решает приведенный фрагмент?

```
x := x0; y := y0; z := 0;
while x <> 0 do
begin if x mod 2 <> 0
  then z := z + y;
       y := y * 2;
       x := x div 2
end;
```

6.54. Какую задачу решает приведенный фрагмент?

```
s := 0; i := a;
while i >= b do
begin p := 1;
  for j := i downto 2 do p := p * j;
      s := p + s;
      i := i - 2
end;
```

Продолжение следует

Калейдоскоп

Отпечатки пальцев для учета рабочего времени

В последнее время использование биометрических методов идентификации личности становится все более популярным. Развитие компьютерных технологий значительно упрощает и удешевляет этот процесс. Компания Computer Craft интегрировала технологию идентификации по отпечаткам пальцев в программный пакет для учета рабочего времени на предприятиях. Теперь вместо предъявления традиционных удостоверений или магнитных карт сотрудники, приходя на работу или уходя с нее, будут прикладывать палец к сканирующему устройству, которое будет идентифицировать сотрудника и посылать данные в центральный компьютер. Программный пакет компании Computer Craft может оказаться особенно удобным и надежным для предприятий с гибким графиком работы.

По материалам журнала "BYTE/Россия"

Виртуальный рентген

Американский портал Radiology.com объявил о новом виде услуг — хранении и передаче медицинской информации (оцифрованных рентгеновских снимков, томограмм и любых других изображений) из своего банка данных. Эта новая услуга значительно улучшит возможности здравоохранения, поскольку до сих пор только крупные медицинские учреждения могли создавать подобные банки данных и внутренние сети для их использования. Теперь любой пациент может оставить свои рентгеновские снимки в центральном банке медицинских данных и получить их при помощи сети Интернет в случае необходимости. Предполагается, что передача данных будет осуществляться по виртуальным частным сетям с оплатой в зависимости от количества переданной информации. Передача одного снимка займет менее двух минут.

По материалам сайта www.radiology. com

Почти первый

"Компьютер ENIAC первоначально предназначался для проведения баллистических расчетов в период второй мировой войны, однако его сооружение было завершено лишь после войны. Вплоть до начала 1950-х годов он активно использовался для научных расчетов" [1].
"По своей структуре компьютер ЭНИАК (ENIAC) напоминал механические вычислительные машины. Запоминающие регистры состояли из триггерных колец (по десять триггеров в каждом кольце)... Система переноса десятков в накопителях была аналогична предварительному переносу в машине Бэббиджа" [2].

Джон Маучли

В августе 1942 года Джон Маучли из Высшего технического училища Пенсильванского университета представил проект (меморандум) "Использование быстродействующих электронных устройств для вычислений", который положил начало созданию электронного компьютера ENIAC (от *Electronic Numerical Integrator and Computer* — электронный цифровой интегратор и компьютер) [2, 3]. Около года проект пролежал без движения, но потом им заинтересовалась баллистическая исследовательская лаборатория армии США. Вскоре армия заключила с училищем контракт на крупную сумму, и началась работа, которой руководили Джон У. Маучли и Дж. Преспер Экерт (уже в восьмилетнем возрасте построивший миниатюрный приемник) [2—6]. Разные по характеру и привычкам, эти люди хорошо дополняли друг друга. Быстрый и общительный Маучли генерировал идеи, а сдержанный Экерт подвергал эти идеи анализу. "Он обладал потрясающей способностью переводить все на практический уровень, пользуясь простыми техническими средствами", — так охарактеризовал Экерта один из специалистов, принимавших участие в работе [3].

Конструкция машины выглядела чрезвычайно сложной. В частности, предполагалось, что компьютер будет содержать около 18 000 ламп, причем он должен был работать с десятичными числами. Маучли предпочитал десятичную систему счисления, поскольку хотел, чтобы "машина была понятна человеку". Однако лампы часто перегревались и выходили из строя, а то, что их было много, ухудшало ситуацию. Экерт частично решил проблему, употребив прием, широко применявшийся при эксплуатации больших электроорганов в концертных залах: на лампы стали подавать несколько меньшее напряжение, и количество аварий снизилось до одной-двух в неделю.

Но главный недостаток ENIAC проявлялся при изменении вводимых в него инструкций, т.е. программы. Внутренней памяти машины с трудом хватало для хранения числовых данных, используемых в расчетах. Программа же задавалась вручную с помощью механических переключателей и гибких кабелей со штекерами, которые вставлялись в нужные разъемы. Это очень напоминало телефонные станции начала XX века [4]. Изменение программы вычислений требовало немалых (в том числе физических) усилий, а занимала подобная работа от нескольких часов до нескольких дней.

Таким образом, ENIAC, весивший 30 тонн и занимавший площадь около 150 м², явно не годился на роль универсальной ЭВМ. Но зато быстродействие этой машины в сотни раз превосходило быстродействие "релейных" компьютеров [2, 7].

Вскоре после того, как ENIAC был собран (в 1945 году), он успешно выдержал испытания, обработав около миллиона перфокарт. Через некоторое время компьютер продемонстрировали представителям прессы. По словам одного репортера, ENIAC работал "быстрее мысли" [3].

Позже право называть ENIAC первым цифровым электронным компьютером было оспорено. Дело сводилось к тому, что в конце 1930-х годов в более простой машине американского профессора физики и математики, болгарина по происхождению, Джона Винсента Атанасова были использованы те же принципы конструирования электронных переключателей на вакуумных лампах. В 1973 году состоялось судебное слушание, и суд постановил, что патентные права на основные идеи цифровых электронных машин принадлежат Атанасову [3, 4, 8—10].

Тем не менее ENIAC можно назвать первым успешно функционирующим электронным цифровым компьютером.

Не успел ENIAC вступить в эксплуатацию, как Маучли и Экерт начали работу по заказу военных над новой машиной. Модель EDVAC (от *Electronic Discrete-Variable Automatic Computer* — электронный дискретный автоматический компьютер) была более гибкой. Ее более вместительная внутренняя память могла хранить не только данные, но и программу...

Литература

1. Толковый словарь по вычислительным системам / Под ред. В.Иллингуорта и др.: Пер. с англ. М.: Машиностроение, 1990.

2. *Частиков А.П.* От калькулятора до суперЭВМ // Новое в жизни, науке и технике. Сер. "Вычислительная техника и ее применение", № 1/88.

3. Знакомьтесь: компьютер: Пер. с англ. М.: Мир, 1989.

4. *Леонов А.Г., Четвергова О.В.* История компьютеров // Информатика, № 35/98.

5. Толковый словарь по вычислительной технике (Microsoft Corporation): Пер. с англ. М.: Издательский отдел "Русская редакция" ТОО "Channel Trading Ltd", 1995.

6. ЭНИАК // Информатика, № 19/2000.

7. *Походзило П.В.* Электронная вычислительная машина // Энциклопедия кибернетики. Киев: Гл. редакция Украинской советской энциклопедии, 1975. Т. 2.

8. *Гутер Р.С., Полупанов Ю.Л.* От абака до компьютера. Изд. 2-е, испр. и доп. М.: Знание, 1981.

9. *Мадейчик Х.* Джон Винсент Атанасофф // Компьютер, № 1/90.

10. Atanasoff Berry Computer // Информатика, № 26/2001.

От ARPAnet к Internet

См. с. 1

В 1960-х годах в США начались эксперименты, связанные с соединением компьютеров друг с другом посредством телефонных линий. Эксперименты проводились при содействии Агентства перспективных исследований Министерства обороны США — ARPA (*Advanced Research Project Agency*). Это агентство было создано в 1957 году, после запуска первого советского спутника.

Данную организацию интересовал вопрос, можно ли соединять расположенные в разных местах компьютеры с помощью новой технологии, имевшей название *коммутация пакетов* (*packet switching*) [1, 2]. Конечно, сотрудники ARPA не стремились создать международное компьютерное сообщество. Их целью являлась организация сети передачи данных, способной функционировать в условиях ядерного конфликта.

Система, получившая название ARPAnet, росла, и вскоре несколько предприимчивых студентов колледжа (и один старшеклассник) предложили способ ее применения для проведения "электронных" конференций. Сначала конференции имели вид научных дискуссий, но вскоре сформировались конференции, посвященные очень многим вопросам.

В 1970-х годах при содействии ARPA были разработаны правила, или протоколы, пересылки данных между различными компьютерными сетями. Эти протоколы с общим именем *Internet* позволили создать Всемирную сеть, соединяющую сегодня компьютеры через границы государств.

В 1980-х годах эта сеть сетей, которая стала известна под таким же названием — *Internet* (*Интернет*), развилась до невероятной степени. Сотни, а потом и тысячи организаций стали подсоединять к ней свои компьютеры. Некоторые предприимчивые любители и компании, не желающие много платить за выход в Интернет (или не имеющие возможности соответствовать жестким правительственным требованиям для получения такого доступа), научились присоединять свои системы к сети даже только ради электронной почты и конференций. Стал предлагаться доступ к Интернету для всех. Любой владелец компьютера и модема получил возможность открыть окно в этот мир.

Если пытаться представить себе, что же такое Интернет и как он работает, то, конечно, возникают ассоциации с телефонной сетью. Обе структуры используют электронные средства передачи, обе позволяют устанавливать соединение и передавать информацию. Кроме того, в Интернете применяются в основном телефонные линии. Однако данная аналогия неверна. Телефонная сеть является сетью с коммутацией каналов. Когда вы осуществляете вызов, вам выделяется определенная часть сети. Даже если ее не использовать, она остается недоступной для других абонентов, которым в этот момент надо позвонить.

В большей степени соответствует действительному положению вещей почтовая связь [3]. Почта представляет собой сеть с уже упоминавшейся здесь коммутацией пакетов. (Сети с коммутацией пакетов обладают высокой надежностью и эффективностью, однако они непригодны для передачи речевых сигналов и видеосигналов в реальном времени [2].) Тут у вас нет выделенного участка сети. Ваша корреспонденция смешивается с другими письмами, отправляется в почтовое отделение и сортируется. Несмотря на то, что данные технологии совершенно различны, служба доставки почты в определенном смысле весьма похожа на Интернет.

По сути дела, Интернет является глобальной сетью, объединяющей множество региональных сетей. В то же время она не имеет головной машины, у нее нет "главного хозяина" [1, 4, 5]. Такая организация обеспечивает высокую живучесть сети: выход из строя даже нескольких десятков компьютеров не должен заметно сказаться на работе системы в целом. Всегда есть запасные пути прохождения информации.

Одной из первых российских сетей, подключенных к Интернету, стала сеть Relcom (Релком) [от *Reliable Communications*, т.е. "надежные коммуникации"], созданная в 1990 году на базе Российского научного центра "Курчатовский институт". В работе, связанной с созданием сети, принимали участие специалисты кооператива "Демос" (сейчас это компания "Демос-Интернет") — в большинстве своем сотрудники этого института. Уже к концу года их стараниями к Интернету было подключено около 30 организаций, в том числе центры российской науки в Серпухове, Санкт-Петербурге, Новосибирске, Дубне.

В 1991 году в компьютерной сети Relcom появился первый сервер новостей (электронных конференций). И очень скоро она объединила многие крупные города России (Екатеринбург, Барнаул и др.), а также некоторых других стран СНГ и стран Балтии. Через три года в этой сети насчитывались несколько сотен узлов и сотни тысяч пользователей...

"Сегодня Интернет состоит из миллионов компьютеров, подключенных друг к другу при помощи самых разных каналов, от сверхбыстродействующих спутниковых магистралей передачи данных до медленных коммутируемых телефонных линий. Каждый такой компьютер независимо от его архитектуры, мощности или конфигурации называется *узлом* сети Интернет и может обмениваться данными с любым другим узлом, причем путь, по которому данные будут передаваться, практически непредсказуем (хотя при необходимости его можно точно определить)" [4].

Литература

1. *Гаффин А.* Путеводитель по глобальной компьютерной сети Internet: Пер. с англ. М.: ТПП "Сфера", 1995.

2. *Фафенбергер Б., Уолл Д.* Толковый словарь по компьютерным технологиям и Internet: Пер. с англ. Изд. 6-е. Киев: Диалектика. 1996.

3. *Крол Э.* Все об Internet: Пер с англ. Киев: Торгово-издательское бюро BHV, 1995.

4. *Фролов А.В., Фролов Г.В.* Создание web-приложений: Практическое руководство. М.: Издательско-торговый дом "Русская Редакция", 2001.

5. *Байков В.Д.* Интернет от e-mail к WWW в примерах. СПб.: BHV — Санкт-Петербург, 1996.

Гл. редактор
С.Л. Островский
Зам. гл. редактора
А.И. Сенокосов
Редакция:
Е.В. Андреева
Н.Л. Беленькая
Л.Н. Картвелишвили
Н.П. Медведева
Дизайн и верстка:
Н.И. Пронская
Корректоры:
Е.Л. Володина,
С.М. Подберезина

выходит четыре раза в месяц
При перепечатке ссылка
на ИНФОРМАТИКУ обязательна,
рукописи не возвращаются

Адрес редакции и издателя:
121165, Киевская, 24
тел. 249-48-96
Отдел рекламы
тел. 249-98-70

Учредитель: ООО "Чистые пруды"

Зарегистрировано в Министерстве РФ по делам печати. ПИ № 77-7230 от 12.04.2001.
Отпечатано в ОИД "Медиа-Пресса",
125993, ГСП-3, Москва, А-40, ул. "Правды", 24
Тираж 6000 экз.
Срок подписания в печать по графику 3.10.2001.
Номер подписан 3.10.2001.
Заказ № 16369
Цена свободная

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков 32291
комплекта изданий 32744

Тел.: (095)249-31-38, 249-33-86. Факс (095)249-31-84

Internet: inf@1september.ru
WWW: http://www.1september.ru

Главный редактор комплекта изданий А.С. Соловейчик

Английский язык (Е.В. Громушкина), индекс подписки — 32025; **Библиотека в школе** (О.К. Громова), индекс подписки — 33376; **Биология** (Н.Г. Иванова), индекс подписки — 32026; **Воскресная школа** (монах Киприан (Ященко), индекс подписки — 32742; **География** (О.Н. Коротова), индекс подписки — 32027; **Здоровье детей** (А.У. Лекманов), индекс подписки — 32033; **Информатика** (С.Л. Островский), индекс подписки — 32291; **Искусство** (Н.Х. Исмаилова), индекс подписки — 32584; **История** (А.Ю. Головатенко), индекс подписки — 32028; **Литература** (Г.Г. Красухин), индекс подписки — 32029; **Математика** (И.Л. Соловейчик), индекс подписки — 32030; **Начальная школа** (М.В. Соловейчик), индекс подписки — 32031; **Немецкий язык** (М.Д. Бузоева), индекс подписки — 32292; **Русский язык** (Л.А. Гончар), индекс подписки — 32383; **Спорт в школе** (Н.В. Школьникова), индекс подписки — 32384; **Управление школой** (А.И. Адамский), индекс подписки — 32652; **Физика** (Н.Д. Козлова), индекс подписки — 32032; **Французский язык** (Г.А. Чесновицкая), индекс подписки — 33371; **Химия** (О.Г. Блохина), индекс подписки — 32034; **Школьный психолог** (М.Н. Сартан), индекс подписки — 32898.