№ 14 (207) апрель1999

еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИК

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

Жаркое лето 99-го

ждет читателей "Информатики"

В летних номерах — тематические сборники лучших материалов этого учебного года, новые учебники, множество оригинальных интереснейших статей.

Не забудьте подписаться на нашу газету!

Подписной лист изданий "Первого сентября" в каталоге агентства "Роспечать":
1-е полугодие 1999 года — стр. 26
2-е полугодие 1999 года — стр. 18

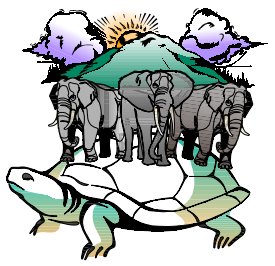
Конец учебного года... Вы устали от серьезных анкет и серьезных вопросов? Но очень хотите выиграть подписку на "Информатику"? Найдите все отличия данной картины от оригинала, отправьте письмо в редакцию не позднее 30.04.99, и у вас появится шанс выиграть подписку на комплект номеров **"Жаркое лето 99-го"** за июль — август 1999 г.

НАШИ ДЕТИ БУДУТ ЖИТЬ В XXI ВЕКЕ

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

Лекция 12

А.Г. КУШНИРЕНКО, Г.В. ЛЕБЕДЕВ



Окончание. Начало в № 1, 3, 5, 6, 8, 10, 11, 12/99

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

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

Системы счисления и компьютерная арифметика

Е.В. АНДРЕЕВА, И.Н. ФАЛИНА

"Информация вместе с веществом и энергией есть важнейшая сущность нашего мира". Чтобы передать информацию, ее надо представить в каком-либо виде.

Разделы:

Тема 1. Системы счисления как разновидность информационных систем.

Тема 2. История систем счисления.

Тема 3. Позиционные системы счисления.

Продолжение следует

"Объектно-ориентированное программирование — парадигма программирования, в которой программа рассматривается как набор дискретных объектов, содержащих, в свою очередь, наборы структур данных и процедур, взаимодействующих с другими объектами" (Толковый словарь по вычислительной технике фирмы Microsoft).

Статья адресована преподавателям информатики, которые хотят познакомить своих учеников с объектно-ориентированным программированием на наглядных и не очень сложных примерах.

Используется система Турбо Паскаль.

Продолжение открытой в № 3/99 дискуссии о комплекте учебников по информатике для школьников под редакцией профессора Н.В. Макаровой, и в частности о новом учебнике "Информатика. 6—7".

Задача 11-й Кировской областной олимпиады по информатике. Другие задачи этой олимпиады были опубликованы в № 13/99.

Первое знакомство с объектно-ориентированным программированием

А.А. СЕМЕНОВ, А.Г. ЮДИНА

Господа программисты,
Теперь Вам не надо изобретать свой велосипед!
Просто наследуйте наш.

(Из рекламного буклета фирмы Borland)

Эта статья адресована тем преподавателям информатики, которые хотели бы познакомить своих учеников с ООП на примерах, по возможности наглядных и не очень сложных.

Попробуем разработать на Паскале программу — хранитель экрана (*screen saver*)с фигурками различной формы, гуляющими по экрану.

Для работы потребуется Турбо Паскаль версии не меньше 6. Никаких особенных требований к технике (от 286-го). Достаточно поставить только TURBO.EXE (интегрированную среду), TURBO.TPH (файл помощи) и TURBO.TPL (библиотеки SYSTEM и CRT). Все вместе занимает 1,3 Мбайта и умещается на дискету. Необходимый уровень подготовки — умение использовать основные конструкции языка (ветвления и циклы), а также описывать и применять процедуры и функции для решения несложных задач вычислительного характера (включая работу с массивами).

По ходу работы над нашим проектом будут затронуты также такие темы, как работа с графикой, использование встроенного ассемблера, устройство видеопамяти, создание модулей — самодельных библиотек процедур и функций. Возможно, вы сочтете это удобным поводом для расширения кругозора ваших учеников.

Занятие 1

Итак, мы решили познакомиться с ООП на примере графических объектов.

Стандартная VGA-графика Паскаля (640×480 точек, 16 цветов) не очень годится для анимации — в ней маловато цветов.

Поэтому попробуем использовать для работы графический режим VGA с шестнадцатеричным номером 13h (320×200 точек, 256 цветов). Доступ к видеопамяти в этом режиме очень простой, а перевести монитор в данный режим можно вызовом 10h прерывания BIOS. И никакого модуля Graph!

Пишем текст:

```
begin
{установить видеорежим 13h — всего 2 строчки на встроенном ассемблере}
  asm
    mov ax, 13h  {пересылка номера видеорежима в регистр}
    int 10h  {вызов прерывания с шестнадцатеричным номером 10h}
  end;
  Readln
end.
```

Запустив программу, мы увидим, что на черном экране нет курсора — это верный признак графического видеорежима. Прежде чем нажать Enter, попробуем печатать на клавиатуре. Не правда ли, экзотический вид у букв, как в старых играх для DOS?

Теперь попытаемся зажечь на экране какой-нибудь пиксель. Для этого достаточно вывести в оперативную память видеокарты (она начинается с адреса `$A000:0000`) байт, значение которого равно номеру цвета в палитре ($0 \le$ номер ≤ 255). Для доступа к памяти используем стандартный массив Паскаля `Mem`. (Об адресации оперативной памяти см. примечание.)

Итак,

```
begin
  asm
    mov ax, 13h
    int 10h
  end;
  Mem[$A000:0]:=14;  {14 — желтый цвет}
  Readln
end.
```

Ура! В верхнем левом углу экрана загорелась желтая звездочка!

Теперь напишем процедуру для вывода точки в любом месте экрана, а затем с ее помощью нарисуем прямоугольник, закрашенный в цвета палитры.

```
procedure PutPixel(x,y: integer; color: byte);
{0≤x≤319;0≤y≤199;0≤color≤255}
var a:word;
begin
  a:=320*y+x;
  Mem[$A000:a]:=color
end;
var
  i,j: integer;
begin
  asm
    mov ax, 13h
    int 10h
  end;
  for i:=0 to 255 do
    for j:=0 to 10 do
     PutPixel(i,j,i);
  Readln
end.
```

Младшие 16 цветов соответствуют 16-цветной палитре режима VGA 640×480. Далее идут оттенки серого и 3 набора цветов различной интенсивности.

Палитра устанавливается 10h прерыванием BIOS во время инициализации видеорежима. Если она нас не устроит, ее нетрудно будет сменить.

Почему в процедуре `PutPixel` вспомогательная переменная `a` описана как `word`, а не как `integer`?

В нашем графическом режиме экран разбит на 320×200=64 000 точек, а максимальное значение для `integer` 32 767. Для типа `word` максимальное значение 65 535.

Домашнее задание

Написать следующие процедуры:
`FillScreen(color: byte)` — заливает экран цветом `color`;
`HLine(x1,x2,y: integer; color: byte)` — рисует горизонтальную линию цвета `color`;
`VLine(x,y1,y2: integer; color: byte)` — рисует вертикальную линию цвета `color`.

Примечание. Почему адрес ячейки оперативной памяти (например, `$A000:16`) записывается так странно?

Восьмиразрядные микропроцессоры (8080, 8085) имели 16-битовую адресную шину и как следствие объем адресуемой памяти 64 Кбайт. Первый 16-разрядный процессор Intel 8086 имел уже 20-битовую шину адреса и мог обращаться к 1 мегабайту оперативной памяти. Для программной совместимости с предшествующими моделями было решено организовать адресацию памяти с помощью двух 16-битовых регистров. Первый из них задает так называемый сегмент памяти, а второй — смещение в пределах данного сегмента. Физический адрес ячейки памяти процессор находит по формуле `A=16*адрес сегмента + смещение`. Таким образом, к одной ячейке памяти можно обратиться несколькими способами!

Довольно-таки странно, зато в пределах одного сегмента памяти поддерживается совместимость по адресации с 8-разрядными процессорами.

Обычно адрес записывают так: `(адрес сегмента) :(смещение)`. Если адрес представлен в шестнадцатеричном формате, то вначале ставят знак $ (в Паскале это признак шестнадцатеричного числа).

Например, адрес `$A000:16` можно записать как `$A000:$10`, или, если учесть, что $10 \cdot 16^3 = 40960$, как `40960:16`. Физический адрес будет равен $40960 \cdot 16+16=655376$.

Занятие 2

Теперь можно перенести все процедуры в самодельную библиотеку (файл сохранить как GRAPH13H.PAS):

```
Unit Graph13h;
interface
procedure Screen13h;
procedure PutPixel(x,y: integer; color: byte);
procedure FillScreen(color: byte);
procedure FillRect(x1,y1,x2,y2:integer;color:byte);
procedure HLine(x1,y,x2: integer; color: byte);
procedure VLine(x,y1,y2: integer; color: byte);

implementation
procedure Screen13h; assembler;
asm
    mov ax, 13h
    int 10h
end;

procedure PutPixel(x,y: integer; color: byte);
var a:word;
begin
     a:=320*y+x;
     mem[$a000:a]:=color
end;

procedure FillScreen(color: byte);
var m,n: integer;
begin
     for m:=0 to 199 do
          for n:=0 to 319 do PutPixel(n,m,color)
end;

{Заливка прямоугольника}
procedure FillRect(x1,y1,x2,y2:integer;color:byte);
var m,n:integer;
begin
    for m:=x1 to x2 do
        for n:=y1 to y2 do PutPixel(m,n,color)
end;

procedure HLine(x1,y,x2: integer; color: byte);
var m,n,k: integer;
begin
     if x1<x2 then begin n:=x1;k:=x2 end else begin n:=x2;k:=x1 end;
     for m:=n to k do PutPixel(m,y,color)
end;

procedure VLine(x,y1,y2: integer; color: byte);
var m,n,k: integer;
begin
      if y1<y2 then begin n:=y1;k:=y2 end else begin n:=y2;k:=y1 end;
      for m:=n to k do PutPixel(x,m,color)
end;
end.
```

Откомпилируем файл GRAPH13H.PAS. Будет создан файл библиотеки GRAPH13H.TPU с объектным кодом процедур.

Программа проверки созданных инструментов существенно упростится:

```
uses Graph13h;
begin
  Screen13h;
  FillScreen(3);
  HLine (0, 150, 50, 2);
  VLine (150, 0, 100, 4);
  readln;
end.
```

и можно будет приступить к созданию объектов.

Создание и использование модулей — очень удобный прием, так как они разгружают текст проекта от рутинных процедур. К тому же они дают возможность одолеть 64-килобайтный предел размера кода программы.

Домашнее задание

Дополнить модуль `Screen13h` процедурой `FillRect(x1,y1,x2,y2:word;color:byte)`, которая заливает прямоугольник с заданными вершинами цветом `color`.

Дополнительное задание “со звездочкой” — дополнить библиотеку процедурой `Line(x1,y1,x2,y2:word;color:byte)` для рисования произвольного отрезка, заданного двумя точками.

Продолжение читайте в № 15/99

Исходя из того, что было сказано выше в теме 3.1, натуральное число a в P-ичной системе счисления можно записать как

$$a=a_nP^n+a_{n-1}P^{n-1}+\ldots+a_1P+a_0,\ 0\le a_i<P,\ i=0,\ 1,\ \ldots,\ n. \quad (1)$$

Правильную конечную P-ичную дробь b можно записать в виде

$$b=b_{-1}P^{-1}+b_{-2}P^{-2}+\ldots+b_{-k}P^{-k},\ 0\le b_i<P,\ i=-1,-2,\ \ldots,\ -k. \quad (2)$$

В формулах (1) и (2) коэффициенты a_i и b_i при степенях основания P являются цифрами в данной позиционной системе счисления. В общем виде любое вещественное число X в позиционной системе счисления с основанием P можно представить следующей формулой:

$$X = a_nP^n+a_{n-1}P^{n-1}+\ldots+a_1P+a_0+\ b_{-1}P^{-1}+ \\ +b_{-2}P^{-2}+\ldots+b_{-k}P^{-k}+\ \ldots \quad (3)$$

Таким образом, P-ичная система счисления позволяет с помощью заранее ограниченного набора цифр записать как сколь угодно большое, так и сколь угодно малое число в виде суммы степеней основания системы.

С другой стороны, любое число в P-ичной системе счисления можно записать просто в виде последовательного перечисления его цифр начиная со старшей. Целая часть от дробной отделяется запятой. То есть представлению вещественного числа a по степеням P вида (3) соответствует запись

$$a=a_n\ldots a_1a_0,b_{-1}\ldots b_{-k}\ldots \quad (4)$$

Определение 10. Представление числа в P-ичной системе счисления в виде (3) называется ***развернутой формой*** записи числа (эта форма в основном используется при решении задач).

Определение 11. Представление числа в P-ичной системе счисления в виде (4) называется ***свернутой формой*** записи числа (как правило, именно так изображаются числа в позиционных системах счисления).

Как было сказано выше, в любой P-ичной системе счисления с помощью заранее ограниченного набора цифр можно записать как сколь угодно большое, так и сколь угодно малое число — в виде ***суммы положительных и отрицательных степеней числа P***. Существует доказательство того, что это представление числа в виде суммы степеней числа P единственно для каждого P. Таким образом,

любое действительное число можно записать в любой *P*-ичной системе счисления в виде степенного ряда и притом единственным образом.

Пример 13. Десятичное число 61 можно записать
в двоичной системе счисления как 111101_2;
в троичной системе счисления как 2021_3;
в четверичной системе как 331_4;
в шестнадцатеричной системе как $3D_{16}$;
в 61-ичной системе как 10_{61}.

Для того чтобы научиться производить арифметические действия в какой-либо системе счисления, прежде всего необходимо уметь вести в ней счет, т.е. перечислять по порядку натуральные числа. Кроме того, нужно знать и то, как представляются в ней дробные части действительных чисел.

Начнем со счета.

В любой P-ичной системе счисления натуральные числа, меньшие основания P, представляются с помощью одной цифры данной системы. Для чисел же, больших или равных P, требуется по крайней мере две цифры. Само число P в системе счисления с основанием P записывается в виде 10_p, что следует из развернутой формы записи числа P в P-ичной системе: $P=1\cdot P+0$.

Выпишем представление первых девяти натуральных чисел в некоторых системах счисления:

$P=10$	1	2	3	4	5	6	7	8	9
$P=2$	1	**10**	11	100	101	110	111	1000	1001
$P=3$	1	2	**10**	11	12	20	21	22	100
$P=4$	1	2	3	**10**	11	12	13	20	21
$P=5$	1	2	3	4	**10**	11	12	13	14

Вопросы и задания

1. Чем характеризуется каждая позиционная система счисления?

2. Сколько и каких требуется цифр, чтобы можно было любое число записать в семеричной системе счисления? А в двенадцатеричной?

3. Выпишите все цифры 20-ричной системы счисления.

4. Выпишите базис 5-ричной системы счисления.

5. Укажите, какие числа записаны с ошибками, ответ объясните: 123_7; 3005_4; $12AAC09_{20}$; 13476_7.

6. Известно, что алфавитом некоей традиционной позиционной системы являются следующие символы: 0, 1, 2, ✂, 🖂, ⇦, ➢, ✓. Каково основание этой системы?

7. Запишите число 8 в системе счисления из предыдущей задачи.

8. Выпишите первые 10 чисел натурального ряда в системе счисления из задачи 6.

9. Запишите в 6-ричной системе счисления число, следующее по порядку за числом 5.

10. Какое число следует за числом 111_4 в 4-ричной системе счисления?

11. Какое число предшествует числу 10_8 в 8-ричной системе счисления?

12. Запишите в развернутом виде числа 65_7; 1998_{10}; $0{,}15A_{16}$; $1AF1H{,}A9_{20}$.

13. Какое минимальное основание может иметь система счисления, если в ней записаны все следующие числа: 432, 120, 111, 2331?

14. Какое из чисел больше: 5_{10} или 5_8; 1111_2 или 1111_8?

15. В каком случае при прибавлении единицы к числу в P-ичной системе счисления количество цифр в результате возрастет по сравнению с исходным числом? Может ли при таком сложении количество цифр возрасти больше чем на одну?

16. Как будет выглядеть в двоичной системе счисления десятичное число 0,125?

17. Запишите в системе счисления с основанием 240 числа 241, 242, 243, 250, 251.

18. Подсчитайте количество двоичных чисел в диапазоне 10_2 до 1000_2.

19. В бумагах чудака математика была найдена его автобиография. Она начиналась следующими удивительными словами: “Я окончил курс университета 44 лет от роду. Спустя год, 100-летним молодым человеком, я женился на 34-летней девушке. Незначительная разница в возрасте — всего 11 лет — способствовала тому, что мы жили общими интересами и мечтами. Спустя немного лет у меня уже была маленькая семья из 10 детей. Жалованья я получал в месяц всего 200 рублей, из которых $\frac{1}{10}$ приходилось отдавать сестре, так что мы с детьми жили на 130 рублей в месяц”.

Чем объяснить странные противоречия в числах приведенной автобиографии?

Информатика — это наука, которая занимается вопросами представления и обработки информации.

(Манфред Брой,
профессор, лауреат премии Лейбница
в области информатики)

Системы счисления и компьютерная арифметика

Е.В. АНДРЕЕВА,
И.Н. ФАЛИНА

ЧАСТЬ I. Представление информации (базовый курс)

Тема 1
Системы счисления как разновидность информационных систем

Понятие информации является центральным в информатике. Информация вместе с веществом и энергией есть важнейшая сущность нашего мира. Информация — это не только сведения из книг, радио- или телепередач, но и сведения, которые хранятся в структуре сложнейшей биологической молекулы, сведения, которые мы получаем, любуясь картиной в музее или наслаждаясь лесными ароматами. Информацию получает и ребенок из маминой сказки... Примеров можно привести великое множество.

Очень важным является способ представления той или иной информации. Только представив информацию в каком-либо виде, ее можно передавать.

Пример 1. Перед посадкой самолета в Париже стюардесса объявляет: “Сейчас в городе идет дождь, температура воздуха — два градуса выше нуля”. Эту информацию она получила от бортинженера. Для передачи полученной информации она воспользовалась английским языком.

Пример 2. Каждый дверной ключ имеет бороздки и выступы, при помощи которых замку передается информация, “своим” ключом пытаются открыть дверь или “чужим”.

Пример 3. Для записи чисел мы используем десятичную систему счисления и договариваемся, что числа записываем слева направо, а не наоборот. Так, число 1998 будет означать “одна тысяча девятьсот девяносто восемь”, а не “восемь тысяч девятьсот девяносто один” или еще какое-нибудь число.

Рассмотрим ситуации, связанные с приведенными выше примерами.

1. (Пример 1) В самолете, летящем в Париж, есть люди, которые не понимают английского языка. Тогда информацию о погоде, переданную стюардессой на английском языке, они воспримут как некий “шум”, а не как совет приготовить зонтик.

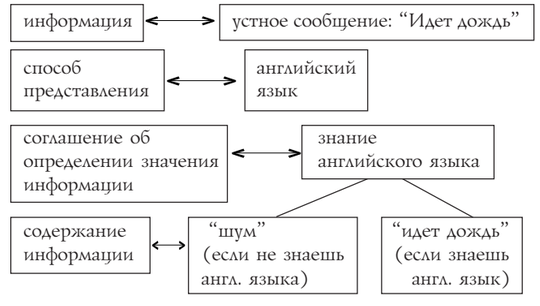
Выражаем признательность издательству “Бином” за содействие в подготовке публикации.

2. (Пример 2) Если человек потерял ключ от двери, то сколько бы он ни говорил, стоя около двери: “Я живу в этой квартире, я открывал замок своим ключом много раз”, — в квартиру он не попадет, так как замок “понимает” только бороздки и выемки, а не человеческую речь.

3. (Пример 3) Если исторический документ, датированный XIII веком, сегодня будет читать обыкновенный человек, то фразу “...произошло в 104 году от Рождества Христова...” он поймет так: указанное событие произошло в 104 году нашей эры. А на самом деле человек, написавший этот документ в XIII веке, имел в виду 401 год нашей эры, поскольку в XIII веке было принято записывать числа справа налево. Подобный способ записи чисел был заимствован у арабов.

Таким образом, очень важно принимать соглашения об ***установлении значения информации ее способу представления, или просто представлению***, т.е. должны быть соглашения (правила), по которым определяется значение информации при данном представлении информации.

Это лучше оформить в таблицу.



Без такого соглашения все представления не имеют определенного смысла, то есть ***содержания***. Только предписывание определенного значения некоему представлению переводит это представление в информацию.

Это условие становится особенно понятным при рассмотрении древних надписей и рисунков, смысл которых пока неизвестен. Несомненно, такие надписи и рисунки несут какую-то информацию, но в настоящий момент она от нас скрыта. Однако закономерности в

формах представления, дополнительные знания и предположения о виде содержащейся там информации позволяют иногда восстановить смысл таких рисунков и надписей. Именно так в 1822 году Шампольоном были расшифрованы древнеегипетские надписи (об этом и других открытиях в области археологии и истории можно прочитать в книге К.Керама "Боги, гробницы, ученые").

Подведем итог сказанному. У каждой информации есть свое собственное содержание и форма его представления. Чтобы понять содержание информации, надо знать правило (соглашение), по которому представление переводится в содержание (смысл, значение).

В этой книге мы будем рассматривать способы представления в основном числовой информации, познакомимся с правилами перевода одного представления числа в другое, а также попытаемся понять, почему одно и то же число в различных ситуациях необходимо представлять по-разному. Таким образом, целью нашего изучения будут ***системы счисления***.

Системы счисления — это системы, созданные человеком. Называют такие системы ***искусственными***, в отличие от ***естественных*** систем, созданных природой. К естественным (природным) системам относятся галактики, наша Солнечная система, человек как единое целое и так далее. К искусственным системам относятся города, заводы, система образования, национальные языки — то есть все, что сделано людьми.

Искусственные системы можно разделить на:

- ***материальные*** — автомобили, самолеты, дома, города, плотины и т.д.;
- ***общественные, т.е. различные объединения людей*** — парламент, система народного образования, шахматный клуб и т.д.;
- ***информационные*** — национальные языки, компьютерная сеть Internet, системы счисления и т.д.

Каждая искусственная система создается с определенной целью. Можно утверждать: из двух искусственных систем лучше та, которая более эффективно обеспечивает достижение цели ее создания.

Целью создания системы счисления является выработка наиболее удобного способа записи чисел. Но "наиболее удобный способ" для кого или для чего? Это понятие несколько расплывчато, и его можно рассматривать с нескольких сторон:

1. Удобство способа записи на физическом носителе (бумаге, камне, дереве и т.д.). Можно предположить, что в вавилонской системе, о которой рассказано в следующей теме, в качестве "цифр" использовались клинья именно потому, что еще не была изобретена бумага, и числа наносились скорее всего на влажные глиняные доски при помощи штампа. Этот штамп поворачивали или острием клина вниз, или острием клина вправо. Точно так же клинья выбивались и на камне.

2. Удобство выполнения арифметических операций над числами в предложенной записи. Именно поэтому позиционные системы счисления чуть ли не сразу же после своего появления практически вытеснили другие системы.

3. Наглядность обучения основам работы с числами. Если бы у нас на каждой руке было четыре пальца, то, вероятнее всего, наша система счисления была бы не десятичной, а восьмеричной.

Вопросы и задания

1. Приведите примеры естественных систем.
2. Приведите примеры искусственных систем.
3. С какой целью создаются такие искусственные системы, как город?
4. Какова цель создания такой искусственной системы, как национальный язык?
5. В примере 2 данной темы укажите, что является информацией, каково представление этой информации и каково соглашение о переводе представления информации в содержание.

Тема 2
История систем счисления

Современный человек в повседневной жизни постоянно сталкивается с числами: мы запоминаем номера автобусов и телефонов, в магазине подсчитываем стоимость покупок, ведем свой семейный бюджет в рублях и копейках (сотых долях рубля) и т.д. и т.п. Числа, цифры... они с нами везде. А что знал человек о числах несколько тысяч лет назад? Вопрос непростой, но очень интересный. Историки доказали, что и пять тысяч лет назад люди могли записывать числа и производить над ними арифметические действия. Конечно, принципы записи были совсем не такими, как сейчас. Но в любом случае число изображалось с помощью одного или нескольких символов.

Эти символы, участвующие в записи числа, в математике и информатике принято называть ***цифрами***.

Но что же люди понимают тогда под словом ***число***?

Первоначально понятие отвлеченного числа отсутствовало, число было "привязано" к тем конкретным предметам, которые пересчитывали. Отвлеченное понятие натурального числа появляется вместе с развитием письменности. Дробные же числа изобрели тогда, когда возникла необходимость производить измерения. Измерение, как известно, это сравнение с другой величиной того же рода, выбираемой в качестве эталона.

Эталон называется еще единицей измерения. Понятно, что единица измерения не всегда укладывалась целое число раз в измеряемой величине. Отсюда и возникла практическая потребность ввести более "мелкие" числа, чем натуральные. Дальнейшее развитие понятия числа было обусловлено уже развитием математики.

Понятие числа — фундаментальное понятие как математики, так и информатики. В дальнейшем при изложении материала ***под числом*** мы будем понимать ***его величину, а не его символьную запись***.

Сегодня, в самом конце XX века, для записи чисел человечество использует в основном десятичную систему счисления. А что такое система счисления?

Система счисления — это способ записи (изображения) чисел.

Различные системы счисления, которые существовали раньше и которые используются в настоящее время, делятся на две группы: позиционные и непозиционные.

Наиболее совершенными являются ***позиционные*** системы счисления, т.е. системы записи чисел, в которых вклад каждой цифры в величину числа зависит от ее ***положения (позиции)*** в последовательности цифр, изображающей число. Например, наша привычная десятичная система является позиционной: в числе 34 цифра 3

Определение 8. Традиционные позиционные системы счисления с основанием P называются ***P-ичными***.

Наряду с широко известными традиционными системами счисления существуют так называемые ***смешанные*** P—Q-ичные системы счисления, а также другие позиционные системы счисления: факториальные, фибоначчиевы, уравновешенные и другие (будем называть их ***нетрадиционными***).

Пример 12. Выпишем базисы некоторых нетрадиционных систем счисления.

Факториальная система: 1!, 2!, 3!, 4!, ..., $(n-1)!$, $n!$, ...

Фибоначчиева система: 1, 2, 3, 5, 8, 13, 21, ...

Таким образом, позиционные системы счисления разделяются на несколько видов:

Подробнее о смешанных и нетрадиционных системах счисления можно прочитать в части II, а также в книге В.Н. Касаткина "Новое о системах счисления" (*В.Н. Касаткин.* Новое о системах счисления. Киев: Вища школа, 1982).

В части I мы будем рассматривать только традиционные P-ичные системы счисления.

Очевидно, что любое натуральное число в десятичной системе счисления можно представить в виде ***конечной суммы степеней числа*** 10 с коэффициентами, равными соответствующим десятичным цифрам натурального числа. Например:

$123=1\cdot 10^2+2\cdot 10^1+3\cdot 10^0=100+20+3$;

$1023=1\cdot 10^3+0\cdot 10^2+2\cdot 10^1+3\cdot 10^0=1000+20+3$.

Аналогично любую десятичную дробь можно представить в виде ***суммы отрицательных степеней числа*** 10. Например:

$0{,}123=0{,}1+0{,}02+0{,}003=1\cdot 10^{-1}+2\cdot 10^{-2}+3\cdot 10^{-3}$.

Так как любое положительное действительное число можно представить в виде суммы его целой части (натуральное число) и дробной части, то в десятичной системе счисления оно представляется в виде суммы положительных и отрицательных степеней числа 10 с коэффициентами, равными десятичным цифрам действительного числа. Поэтому в соответствии с определением 3 наша десятичная система является позиционной. Соответственно, число 10 является основанием десятичной системы счисления.

Таким образом, относительно привычной десятичной системы можно сказать следующее:

1) базисом десятичной системы счисления являются члены геометрической прогрессии со знаменателем 10 : 1, 10, 100, 1000, ...;

2) основанием системы является число 10;

3) алфавитом десятичной системы являются цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Как уже было сказано, существуют и другие P-ичные системы счисления (традиционные позиционные системы), например, двоичная, троичная, восьмеричная, шестнадцатеричная. Название каждой такой системы образовано от названия ее основания.

Система счисления	Основание	Кол-во цифр	Цифры
Двоичная	2	2	0, 1
Троичная	3	3	0, 1, 2
Восьмеричная	8	8	0, 1, 2, 3, 4, 5, 6, 7
Шестнадцатеричная	16	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Если мы захотим записать какое-либо число в двоичной системе счисления, то сможем это сделать, используя только цифры 0 и 1. Например, число 9 в двоичной системе счисления имеет вид 1001. Для записи числа в восьмеричной системе счисления мы будем пользоваться цифрами от 0 до 7 включительно. Так, число 10 в восьмеричной системе счисления имеет вид 12.

Количество цифр в алфавите P-ичной системы равно ***основанию*** системы счисления.

Алфавитом произвольной системы счисления с основанием P служат числа 0, 1, ..., $P-1$, каждое из которых должно быть записано с помощью одного уникального (то есть отличного от других) символа, младшей же цифрой всегда является 0.

Основанием позиционной P-ичной системы счисления может быть любое натуральное число, большее или равное двум.

Если основание системы счисления P меньше десяти, то в качестве алфавита удобно (но не обязательно) использовать первые P арабских цифр. Например, в пятеричной системе счисления будут использоваться пять младших десятичных цифр: 0, 1, 2, 3, 4.

Если основание системы счисления больше 10, но меньше либо равно 36, то в качестве первых 10 цифр обычно используют десять арабских цифр, а в качестве остальных цифр — буквы латинского алфавита.

Для систем счисления с основаниями, большими 36, единых правил для формы записи цифр не существует.

Определение 9. Число, записанное в десятичной системе счисления, будем называть ***десятичным***, число, записанное в P-ичной системе счисления, будем называть ***P-ичным***.

Например, числа, записанные в двоичной системе счисления, будем называть ***двоичными***, в шестнадцатеричной системе счисления — ***шестнадцатеричными***.

Для того чтобы различить числа, записанные в системах счисления с разными основаниями, после записи числа в виде нижнего индекса будем ставить значение основания. Исключение может составить лишь десятичная система счисления, индекс которой часто опускается. Например: $45054=127776_8=AFFE_{16}=1010111111111110_2$.

3.2. Представление чисел в P-ичных системах счисления

В десятичной системе счисления 10 единиц каждого разряда образуют единицу следующего старшего разряда. В P-ичной системе счисления единицами разрядов служат последовательные степени числа P, иначе говоря, P единиц какого-либо разряда образуют единицу следующего старшего разряда. Для записи чисел в P-ичной системе счисления необходимо P различных цифр.

мов не бывает. Есть разные способы записи алгоритмов, но нет алгоритмов вне всяких форм записи.

Я, конечно, имею в виду формальное понятие алгоритма, а не "идею алгоритма", понимаемую на бытовом уровне. (Точно так же энергия в физике — это строго определенная величина, которую можно измерить. Говоря в быту "энергичный человек", мы имеем в виду отнюдь не количество "физической" энергии, которое может быть извлечено из жировых запасов человека.)

Так вот, если говорить о строгом формальном понятии алгоритма, то это понятие ***в точности совпадает*** с понятием программы. Термин алгоритм, правда, возник задолго до термина программа и поэтому применяется обычно не к любой программе, а лишь к программам с ***ярко выраженной идеей***, как правило, ***математической***. Так, например, можно говорить об алгоритме Евклида, алгоритме нахождения корня непрерывной функции методом деления отрезка пополам и т.п.

Конечно, можно попробовать записать алгоритм Евклида в разных нотациях, в разных системах обозначений, а потом сказать, что алгоритм — это то общее, что останется, если отрешиться от формы записи. В наши дни большинство философов, логиков и математиков верят в так называемый тезис Черча, который состоит в том, что такое понятие алгоритма законно, что все такие интуитивные понятия последовательного детерминированного алгоритма совпадают.

Но для наших целей — для преподавания информатики — такое понятие бесполезно. Ведь именно курс информатики призван выработать у ученика то интуитивное представление об алгоритме, на которое опирается этот подход. А не зафиксировав нотацию, не зафиксировав форму записи, невозможно ни показать пример алгоритма, ни составить алгоритм, ни проверить решение.

Поэтому в нашем курсе и в нашем учебнике, в отличие от первых учебников А.П. Ершова, алгоритмом называется просто соответствующая конструкция алгоритмического языка от **алг** до **кон**. Или, другими словами, ***алгоритм — это программа на алгоритмическом языке***. Слова "алгоритмизация" и "программирование" мы считаем ***синонимами***. А технический перевод программы (или алгоритма) с одного языка на другой мы называем кодированием (если это делает человек) или компиляцией (если перевод осуществляет компьютер).

Естественно, нас часто обвиняют в подмене информатики и/или алгоритмизации программированием. При этом в слово "программирование" вкладывается негативный оттенок "технической" деятельности из первых учебников А.П. Ершова.

Беда в том, что до сих пор очень часто в разных курсах учат конкретному языку программирования, а не программированию как таковому. В наших аналогиях это все равно, как если бы мы пытались научить человека решать математические задачи, изучая устройство ручки и бумаги, мела и доски. Я не буду здесь повторять все, что говорил о целях и методах построения нашего курса, но, надеюсь, вы уже почувствовали разницу между изучением конкретного языка и обучением основам алгоритмизации.

Что же касается призыва не подменять информатику программированием, то им, к сожалению, слишком часто прикрывают элементарное незнание предмета, помноженное на желание выглядеть специалистом в этой области. Ибо для того, чтобы рассуждать об алгоритмах типа перехода улицы или заварки кофе, говорить о "понятности" алгоритма исполнителю или изучать такое свойство информации, как "своевременность", вообще ничего знать не надо. Это даже проще, чем перечислять операторы Бейсика, не говоря уже о том, чтобы учить думать и решать задачи, учить основам алгоритмизации. Лично мне до сих пор не встретилось ни одного учебного пособия, в котором информатика бы сильно отличалась от алгоритмизации (или программирования в нашем смысле слова) и при этом про разницу между информатикой и алгоритмизацией было бы сказано хоть что-нибудь содержательное, имеющее отношение к какой-нибудь науке.

Конечно, для работы на компьютере в наши дни человеку приходится использовать много сложных программных систем. Для эффективного использования таких систем нужно определенное время на их изучение, на привыкание, на тренинг. Бывают нужны и специальные курсы по изучению конкретного программного обеспечения.

Но наша точка зрения состоит в том, что образование в области информатики не может сводиться к таким специализированным курсам, к изучению особенностей конкретных систем, а должно обязательно включать курс основ алгоритмизации, который и обсуждался в этой книге.

Литература

[Ершов] Под ред. *Ершова А.П., Монахова М.В.* Основы информатики и вычислительной техники. Чч. I и II. М.: Просвещение, 1986.

[Ершов—мы—Шень] *Ершов А.П., Кушниренко А.Г., Лебедев Г.В., Семенов А.Л., Шень А.Х.* Основы информатики и вычислительной техники. М.: Просвещение, 1988.

[Инф] *Кушниренко А.Г., Лебедев Г.В., Сворень Р.А.* Основы информатики и вычислительной техники. М.: 1990, 1991, 1993, 1996.

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

[Авербух] *Авербух А.В., Гисин В.Б., Зайдельман Я.Н., Лебедев Г.В.* Изучение основ информатики и вычислительной техники. Пособие для учителя. М.: Просвещение, 1992.

[Дрофа 9, 10] *Под ред. Кушниренко А.Г., Эпиктетова М.Г.* Кодирование информации. 9—10-е классы. М.: Дрофа, 1996.

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

[Каймин] *Каймин В.А. и др.* Основы информатики и вычислительной техники. Пробный учебник для 9—10-х классов средних школ. М.: Просвещение, 1988—1989.

[Шень] *Звонкин А.К., Ландо С.К., Семенов А.Л., Шень А.Х.* Алгоритмика. М.: Институт новых технологий образования, 1994.

[Шень-2] *Звонкин А.К., Ландо С.К., Семенов А.Л., Шень А.Х.* Алгоритмика. М.: НМУ, 1995.

[Звенигородский] *Звенигородский Г.А.* Вычислительная техника и ее применение. М.: Просвещение, 1987.

[Гельфанд] *Гельфанд И.М.* Функции и графики. Изд. 5-е. М.: Наука, 1973.

[ЗК] Знакомьтесь: компьютер. М.: Мир, 1989.

№ 14 (207) март 1999

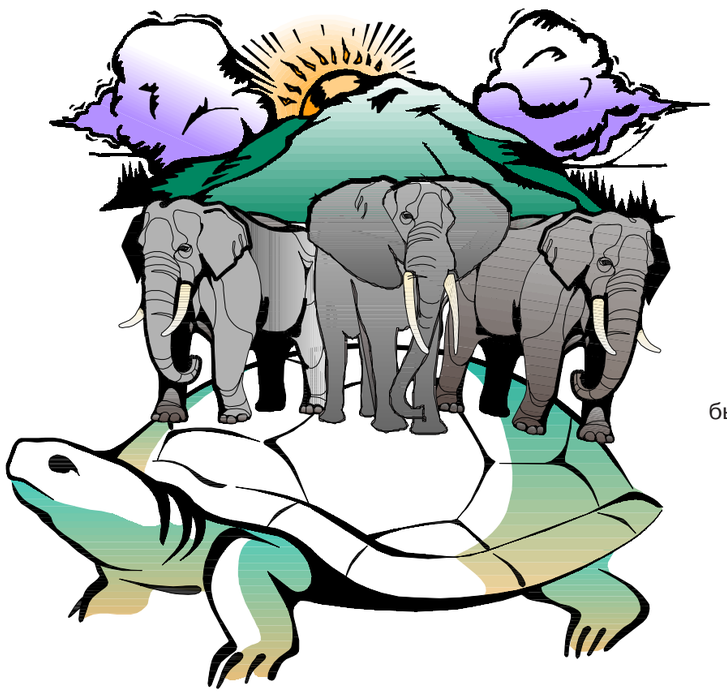
еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИК

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

Лекция 12

А.Г. КУШНИРЕНКО, Г.В. ЛЕБЕДЕВ



Введение, Лекции 1, 2, 3, 4, 5, 6, 7—8, 9, 10 были опубликованы в № 1, 3, 5, 6, 8, 10, 11/99

Выражаем признательность издательству "Дрофа" за содействие в подготовке публикации.

СОДЕРЖАНИЕ

плоскости будет отмечена точка (A, B), а в левой — прямая $y = A \cdot x + B$, касающаяся графика $y = -x^3$. (В этой точке уравнение $x^3 + A \cdot X + B = 0$ имеет корень.) Найдите еще несколько таких точек. Попробуйте составить уравнение кривой, на которой расположены все эти точки.

(*Ответ*: $4 \cdot a^3 + 27 \cdot b^2 = 0$. Это выражение называется дискриминантом кубического уравнения $x^3 + a \cdot x + b = 0$. Когда дискриминант положителен, уравнение имеет один корень, а когда отрицателен — три).

E7. Возможности системы КуМир — система “Функции и графики”

Большую и сложную программную систему в КуМире — из-за ограничений на объем программы — реализовать нельзя. Но можно разбить систему на две части — базовую и интерфейсную. Базовую часть можно реализовать на Си как внешнего исполнителя, подключаемого к КуМиру при запуске. После реализации и отладки эта базовая часть уже не нуждается в изменении. Интерфейсная же часть может быть реализована на алгоритмическом языке в самом КуМире с тем, чтобы потом ее было легко менять. Именно таким образом была реализована система “Функции и графики”, поддерживающая одноименную книгу [Гельфанд].

E8. Возможности системы КуМир — система КуМир-Гипертекст

Гипертексты, то есть тексты с дополнительной структурой, использовались задолго до появления компьютеров. Примерами гипертекстов могут служить толковый словарь, в котором одни словарные статьи ссылаются на другие, или энциклопедии, где набраны специальным шрифтом слова, значение которых можно найти в этой же энциклопедии. Совокупность научных статей всех математических журналов также представляет собой гипертекст — в каждой статье есть ссылки на другие статьи. С появлением компьютеров работа с гипертекстами настолько упростилась, что гипертекстовое представление информации стало преобладающим.

Компьютерный гипертекст — это обычный текст с картинками, в котором выделены — как правило, цветом — отдельные знаки, слова, фразы или картинки, называемые ***полями***. С каждым полем в гипертексте (при нажатии на кнопку мышки) связано определенное действие, например:

— переход в другое место того же гипертекста;
— переход в определенное место другого гипертекста;
— проигрывание фрагмента звукозаписи или видеозаписи;
— запуск на выполнение заданной программы и т.д.

Снабдив гипертекст некоторым количеством программ, можно подготовить так называемый ***активный гипертекст,*** содержание и поведение которого зависят от работы человека с этим гипертекстом. Для записи и программирования гипертекстов используются специальные языки. Система КуМир-Гипертекст позволяет создавать активные гипертексты, используя школьный алгоритмический язык и систему программирования КуМир.

В комплекте учебных гипертекстов НовоМир есть гипертексты, знакомящие с Роботом и Чертежником, обучающие конструкциям алгоритмического языка. Есть и небольшой задачник: школьник решает задачу, а специальная программа (также написанная на алгоритмическом языке в КуМире) проверяет решение и сообщает об ошибках.

В 1997 году в рамках развития системы КуМир-Гипертекст начаты работы по переводу этой системы на мировой стандарт HTML.

E9. Возможности системы КуМир — система ПланиМир

Основным объектом любой геометрической конструкции, задачи или теоремы является чертеж. ПланиМир позволяет ученику строить чертежи на экране компьютера (т.е. на плоскости — планиметрия) с помощью мышки. Результаты построения запоминаются, но в виде программы, а не картинки. Эта программа управляет внешним исполнителем PlaniMir, который, собственно, и рисует картинку на экране. Система ПланиМир устроена так, что программа выполняется несколько десятков раз в секунду. Когда ученик с помощью мышки меняет положение каких-то элементов чертежа, мгновенно изменяется программа, а следовательно, и картинка на экране. Если, например, ученик построил чертеж треугольника с тремя медианами, то при изменении (с помощью мышки) положения одной из вершин треугольника он увидит “мультфильм” — непрерывное изменение треугольника, при котором три медианы продолжают пересекаться в одной точке.

Таким образом, результатом решения задачи на построение является программа, описывающая построение. Правильность этой программы может быть проверена с помощью заранее подготовленной программы на школьном алгоритмическом языке.

Именно так организован практикум по решению задач на построение с использованием систем КуМир-Гипертекст и ПланиМир.

E10. Алгоритмы и программы. Алгоритмизация и программирование

В заключение от вопросов, связанных с конкретным программным обеспечением, вернемся к еще одному часто задаваемому фундаментальному вопросу о взаимосвязи алгоритмизации и программирования в нашем курсе.

Я уже говорил, что в первых школьных учебниках [Ершов] алгоритмы противопоставлялись программам. Алгоритмизацией называлась творческая деятельность по придумыванию алгоритма, т.е. “существа” вычислений. Программированием называлась техническая работа по формальной записи на одном из языков программирования уже придуманного и неформально записанного (на алгоритмическом языке) алгоритма.

С превращением школьного алгоритмического языка в один из языков программирования вопрос о сходстве и различии понятий “алгоритм” и “программа” возник вновь. После долгого и тщательного изучения этого вопроса мы вынуждены были констатировать, что “неформальных”, или “никак не записанных”, алгорит-

бований (средствами КуМира новый тип величин ввести нельзя). Но создавать придется только нового исполнителя (Вычеты по модулю 2), система КуМир потом настроится автоматически.

E6. Возможности системы КуМир — реализация учебной системы с нуля

Простенькая учебная система может быть создана в КуМире с нуля. Вот описание одной такой системы, которую вы можете предложить реализовать школьникам, используя исполнитель Гратекс. (А.Г. Кушниренко и С.Ю. Оревков в свое время написали такую систему за вечер. Ее объем — меньше 300 строк на алгоритмическом языке.) Система называется "Фазовое пространство" и предназначена для иллюстрации некоторых понятий аналитической геометрии и алгебры.

Описание системы "Фазовое пространство".

На экране постоянно изображены две плоскости:
- — плоскость ***точек*** с координатами (x, y) в левой половине экрана;
- — плоскость ***прямых*** с координатами (a, b) в правой половине экрана.

Если указать мышкой точку (A, B) на плоскости прямых и нажать правую кнопку, то в плоскости точек (x, y) появится нарисованная прямая $y = A \cdot x + B$. Если двигать мышку при нажатой правой кнопке, то прямая на плоскости точек будет меняться в соответствии с изменением координат (A, B). При отпускании кнопки прямая исчезнет.

При нажатии левой кнопки точка (A, B) в правой плоскости и соответствующая ей прямая в левой плоскости будут нарисованы (и останутся при отпускании кнопки).

Если указать мышкой точку (X, Y) на плоскости точек и нажать правую кнопку, то в плоскости прямых (a, b) будет нарисована прямая с уравнением $b = -a \cdot X + Y$. Если двигать мышку при нажатой правой кнопке, то прямая будет меняться в соответствии с изменением (X, Y). При отпускании кнопки прямая исчезнет.

При нажатии левой кнопки точка (X, Y) в левой плоскости и соответствующая прямая в правой плоскости будут нарисованы (и останутся при отпускании кнопки).

Эта система позволяет проиллюстрировать тот факт, что множество невертикальных прямых в заданной плоскости само может рассматриваться как плоскость с координатами (a, b). Множество всех возможных состояний какого-то объекта в математике и физике называется ***фазовым пространством*** этого объекта. В нашем случае объект — это невертикальная прямая на плоскости (x, y), а фазовое пространство — плоскость с координатами (a, b).

Работа с системой состоит в решении нескольких приведенных ниже задач. Перед решением каждой из них по нажатии на клавишу F10 обе плоскости очищаются. Следующие клавиши позволяют изобразить на экране данные, необходимые для решения разных задач:

F1 — в плоскости точек появляется случайная точка;

F2 — в плоскости точек появляется случайная пара точек;

F3 — в плоскости прямых появляется случайная точка;

F4 — в плоскости прямых появляется случайная пара точек;

F5 — в плоскости точек появляется график функции $y = x^2$;

F6 — в плоскости точек появляется график функции $y = |x|$;

F7 — в плоскости точек появляется график функции $y = -x^3$.

Задачи, решаемые в системе "Фазовое пространство".

1. Задана точка в левой плоскости (F1). Отметьте на правой плоскости несколько точек так, чтобы соответствующие прямые проходили через заданную точку на левой плоскости. Что можно сказать про расположение этих точек?

2. Задана пара точек в левой плоскости (F2). Нажмите правую кнопку мыши в правой полуплоскости и переместите мышку так, чтобы соответствующая прямая в левой плоскости прошла через заданные две точки. Проделайте это для нескольких пар точек. Сформулируйте эмпирический алгоритм, как надо действовать мышкой в правой плоскости, чтобы провести прямую через две заданные точки в левой плоскости.

3—4. Аналог задач 1—2 с использованием клавиш F3 и F4 и перемещением мышки в левой плоскости.

5. Задана парабола в левой плоскости (F5). Ведите мышку вдоль параболы и периодически нажимайте левую кнопку мыши. В правой плоскости окажется "заштрихованным" прямыми некоторое множество M. Опишите границу этого множества. Переместите мышь в правую плоскость и нажмите на левую кнопку мыши на границе множества M, внутри M, снаружи M. Как будут расположены соответствующие прямые в левой плоскости по отношению к параболе?

6. Аналог задачи 5 для функции $y = |x|$ (клавиша F6).

7. Задан график функции $y = -x^3$ (F7). Нажмите правую кнопку мыши в правой плоскости и попробуйте переместить мышку так, чтобы соответствующая прямая в левой плоскости касалась кривой $y = -x^3$. В этом положении нажмите левую кнопку мыши. В правой

Лекция 12

D2. Место курса в "большой информатике"

Вы знаете, что есть курсы, в которых упор делается на Бейсик; есть наш курс; есть курсы, опирающиеся на Пролог или Лого; есть какие-то загадочные функциональные и параллельные языки и т.д. И, в общем, не очень понятно, как то, что здесь рассказано, связано, например, с параллельными процессами, логическими языками и пр., о чем речи не было. Сейчас я коротко попробую это изложить.

Процедурная, функциональная и логическая традиции.

С некоей совсем глобальной точки зрения в современной информатике (*computer science*) существуют три основных направления, три ветви, почему-то называемые "традициями": процедурная, функциональная и логическая.

"Процедурная" традиция предполагает описания того, что компьютер должен сделать, т.е. задаются и описываются ***действия***. Бейсик, Паскаль, школьный алгоритмический язык, Фортран, Алгол, Ада, Си, Ассемблер, как и большинство других известных языков программирования, относятся к процедурной традиции. Это основное течение, основное направление современной информатики.

Для контраста рассмотрим другие традиции, например, "функциональную". Поскольку любая программа преобразует набор исходных данных в некоторый результат, то программу можно считать функцией, которая исходные данные отображает в результаты. Эту функцию можно пытаться задать как композицию других, более простых функций: к примеру, $(x - 1)^2$ можно записать в виде композиции $f(x) = g^2(x)$, где $g(x) = x - 1$. Примерно так в функциональных языках программирования программу в целом рассматривают как преобразование исходных данных в результаты, т.е. как некоторую сложную функцию, которую пытаются задать в виде композиции более простых функций. Те, в свою очередь, пытаются задать в виде композиции еще более простых функций и т.д. При этом нет места никаким циклам, ветвлениям и прочим "процедурным" вещам, поскольку используется совсем другой подход. Место циклов, например, занимают рекурсивные функции — вспомните последовательность Фибоначчи, задаваемую соотношением $f(n) = f(n - 1) + f(n - 2)$.

Типичными представителями функциональных языков являются Лисп и Рефал, хотя, скажем, Лисп содержит и некоторые элементы из процедурной традиции. В функциональных языках, повторю, программа задается как некоторая функция, которая выражается через другие более простые функции, и т.д. — до каких-то базовых функций. Роль вспомогательных алгоритмов (накопленных знаний) при этом играют ранее составленные функции. Таковы функциональная традиция, функциональные языки программирования. Ни в одном из распространенных школьных курсов про это нет ни слова. Хотя Лисп является популярным языком в задачах, именуемых "системы искусственного интеллекта", а Рефал часто используется в системах обработки текстов: преобразование одного текста в другой задается как некоторая функция (для тех, кто знаком с теорией алгоритмов Маркова, добавлю, что Рефал можно считать программной реализацией этого подхода).

Обращаю ваше внимание, что функциональная традиция опирается на математический стиль мышления: одну функцию представляем в виде композиции других, другие, в свою очередь, в виде композиции третьих и т.д.

Наконец, "логическая" традиция. Я не могу привести вам названия широко распространенных "логических" языков. Обычно это специальные языки для управления базами данных, которые описывают данные и связи между ними. Чтобы понять, о чем идет речь, вполне достаточно примера из учебника [Каймин] (язык Пролог): задается, что "мама мамы есть бабушка" (связь), что "Катя — мама Оли", а "Таня — мама Кати" (факты), и компьютер должен сделать заключение, что "Таня — бабушка Оли". Подобного рода описания каких-то фактов и связей, или, говоря другим языком, аксиом и правил вывода (правил, по которым из одних фактов можно получать другие), составляют основу логических языков.

В логических языках мы описываем только факты и только связи, а потом задаем вопросы, "верно ли, что..." или "какова связь между...". В чисто логической традиции компьютер должен перебрать все возможные варианты и установить, есть ли такие факты, из которых по заданным правилам вывода можно вывести то, что мы спросили. Соответственно, мы либо получим ответ (вместе с "доказательством" — выводом), либо ответ, что таких фактов и выводов нет.

Пролог, впрочем, не чисто логический язык — в нем также есть элементы процедурной традиции, т.е. возможность явно указать, что надо ***делать*** (чтобы не перебирать уж совсем все).

Процедурная традиция является сейчас абсолютно доминирующей. Функциональная достаточно развита в некоторых специальных областях. Что же касается логической, то ее область применения пока чрезвычайно узка — работа с базами данных. Я думаю, это происходит из-за того, что в "чистом" виде "логическая" традиция жутко неэффективна. Дело в том, что в чисто логической традиции не предполагается вообще задание программы ни в виде алгоритма, ни в виде функциональной взаимосвязи. Машина просто должна перебрать все возможные варианты, а перебор вариантов — дело долгое и изначально неэффективное.

ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ. Японцы пытались эту проблему обойти, создав ЭВМ с большим количеством процессоров. Всякий раз, когда задача разбивается на подзадачи, когда надо проверить вариант *а* и вариант *б* (а ведь там свои подслучаи, скажем, *а*1, *а*2, *а*3 и т.д.), они на каждой развилке включали новые процессоры, для параллельного анализа разных вариантов, случаев и подслучаев (обычно именно эту идею у нас называют "японским компьютером 5-го поколения"). Другими словами, алгоритмическую неэффективность подхода пытались компенсировать огромным количеством процессоров, работающих каждый над своим случаем. За счет этого перебор (пока процессоров хватает на новые случаи и подслучаи) не должен приводить к затяжке времени ответа.

Я вам скажу свою глубоко субъективную точку зрения — вы ее можете принимать или не принимать, — но лично я считаю, что этот проект был затеян для того, чтобы тянуть деньги из японского правительства. Было взято изначально неэффективное решение, и под него начали разрабатывать какую-то жуткую аппаратуру. Конечно, проект крайне дорогостоящий, сюда можно вбухать уйму денег, а проект все будет тянуться и тянуться, пока идея себя полностью не дискредитирует.

Конечно, моя точка зрения не отрицает возможности каких-то открытий и достижений в рамках этого проекта. Когда вы варите все подряд, пытаясь получить золото, у вас случайно может получиться резина. Я лишь призываю вас с некоторым скепсисом относиться собственно к идее варки всего подряд с целью получения золота и не строить школьный курс, полагаясь на безответственные высказывания о том, что в будущем нас ждет исключительно и только Пролог и машины "5-го поколения".

В заключение обзора логической традиции я бы хотел заметить, что у этой традиции есть вполне эффективная область применения — компактификация информации в базах данных (такие базы называются обычно "базами знаний"), когда хранятся не только факты, но и некоторые связи и большой пласт информации может не храниться, а генерироваться из фактов и связей.

Если вы еще раз взглянете на эти традиции, то увидите, что процедурная традиция связана с алгоритмическим стилем мышления, а функциональная и логическая опираются на математический или логический стиль мышления. Поскольку основной целью нашего курса является развитие алгоритмической составляющей мышления, мы пошли по "процедурной" ветви, являющейся, впрочем, основной и доминирующей и в "большой" информатике. Это первая развилка.

Базовые понятия процедурной традиции.

Вторая развилка будет уже внутри процедурной традиции. Языки процедурной традиции можно делить по тому, какие основные понятия алгоритмизации они содержат или не содержат. Первое, что мы можем сделать, — разбить языки в зависимости от того, какие из четырех базовых понятий информатики (см.: Лекция 1, № 1/99) они содержат или не содержат. На самом деле, впрочем, во всех процедурных языках есть циклы и таблицы (массивы), и рассматривать мы должны лишь наличие вспомогательных алгоритмов с параметрами и исполнителей.

Я вам говорил, что в последние годы эти понятия, включая "исполнителя", появились во всех современных языках программирования.

Но если мы говорим о традиционном Бейсике, то в нем нет ни того, ни другого. (Кстати, авторы Бейсика объясняли появление языка True Basic примерно так: "Некоторые думают, что Бейсик — конкретный язык с номерами строчек и прочими особенностями. Они ошибаются. Мы, авторы, считаем, что Бейсик — это когда удобно. Жизнь изменилась, и сейчас удобно совсем не то, что было удобно двадцать лет назад. Поэтому настоящий Бейсик (True Basic) сейчас — это язык без номеров строк, в котором есть вспомогательные алгоритмы с параметрами и т.д.".)

Но Бейсик, обычно используемый в наших школах, — традиционный Бейсик 30-летней давности — не содержит ни вспомогательных алгоритмов с параметрами, ни тем более понятия исполнителя.

К сожалению, исполнителей нет и в традиционном, классическом Паскале. (Как и с другими языками, новые версии обычно существенно отличаются от классических, например, понятие исполнителя было введено в UCSD Паскале). Но, возвращаясь к классическим версиям этого языка, обычно используемым в школах, мы должны констатировать, что Паскаль не содержит понятия исполнителя. У Паскаля есть и другие недостатки (например, то, что называется "блочной структурой"), но нас сейчас интересует только наличие в языке базовых понятий информатики.

Именно отсутствие фундаментальных понятий и определило наш отказ от использования Бейсика или Паскаля. Что же касается школьного алгоритмического языка, то тут картина следующая. К моменту написания учебника школьный алгоритмический язык все еще был языком, "придуманным" Андреем Петровичем Ершовым. А поскольку он был "придуманным", то мы без труда "допридумали" к нему конструкцию исполнителя. И получилось уже то, что надо. Полноценные реализации этого языка (система КуМир) появились позже и сразу содержали все конструкции, включая исполнителя.

Замечу, что, кроме языков, явно содержащих понятие исполнителя, таких, как Simula-67, Smalltalk, Ада, Модула, школьный алгоритмический, C++ и др., есть языки, в которых такой явной конструкции нет, но она достаточно легко имитируется (подобно тому как с помощью **goto** можно реализовать цикл **пока**). К таким языкам относятся, в частности, язык Си и — как это ни удивительно — очень старый, но бывший в свое время очень популярным язык Фортран (в котором исполнителей можно смоделировать через механизм COMMON-блоков).

Итак, мы использовали школьный алгоритмический язык А.П. Ершова, "допридумав" в него конструкцию исполнителя, примерно так же, как авторы UCSD Паскаля "допридумывали" Паскаль, авторы C++ "до-

— простые программы легко набрать и просто отладить;
— лексика полностью совпадает с учебником;
— есть ввод-вывод, в том числе и файловый;
— есть графика;
— ученики могут написать свою игровую программу и играть с ней;
— объем программы — до 1000 строк;
— имеется возможность задания тела алгоритма в машинных кодах;
— есть команда "выход" для неструктурного завершения конструкции;
— есть полная рекурсия;
— есть конструкция "исполнитель", общие величины исполнителей;
— возможно гибкое подключение исполнителей с полным контролем правильности вызовов;
— есть возможность реализации исполнителей на Си и их подключения к КуМиру.

E5. Возможности системы КуМир — внешние исполнители

Уже на самом старте проектирования и разработки КуМира было ясно, что в дальнейшем понадобится как совершенствовать курс информатики, так и разрабатывать учебное программное обеспечение для математики, физики и других предметов.

Чтобы КуМир можно было использовать в этой деятельности, а также чтобы обеспечить возможность развития системы (и даже языка), в КуМире была реализована концепция внешних пакетов программ, внешних исполнителей. Пакет программ реализуется на производственном языке Си или C++ по определенным правилам и подсоединяется к системе КуМир при ее запуске. Вот несколько примеров таких пакетов — внешних исполнителей, реализованных в комплекте КуМир:

1) Вездеход — учебный исполнитель, моделирующий вездеход с лазерным дальномером, перемещающийся по ровной местности с произвольно расположенными стенами (задаются ломаными линиями);
2) Двуног — учебный исполнитель, который я уже подробно описывал (см. стр. 318);
3) FIO — файловый ввод-вывод;
4) Гратекс — пакет программ для работы с графикой и текстами;
5) Комплексные числа;
6) Функции и графики (см. ниже);
7) Планимир (см. ниже).

Что дает возможность подключения внешних пакетов с методической точки зрения.
Первый аспект — ничего лишнего.

Допустим, на первом занятии учитель работает с одним исполнителем — Роботом. Тогда он может организовать запуск КуМира только с Роботом. При получении справок ученик будет видеть только информацию о командах Робота — ничего лишнего. По одной-двум первым буквам команды КуМир сможет автоматически "дописать" команду целиком, и при этом названия команд других исполнителей не будут "мешать".

Для одновременной работы с Роботом и Чертежником к КуМиру при запуске подключаются оба этих исполнителя. Соответственно меняются подсказки, автоматическое продолжение команд и пр.

Если на уроке решается задача о росте суммы вклада (алгоритм А57 учебника), то можно запустить КуМир без внешних исполнителей — простейшие команды ввода-вывода "встроены" в язык. Если же необходимо прочесть какую-то начальную информацию из файла, то можно при запуске КуМира загрузить исполнитель FIO.

Второй методический аспект — возможность менять обстановку и создавать новых исполнителей.

Допустим, учитель решил на уроке геометрии использовать Вездеход, но при подготовке к уроку понял, что имеющаяся в комплекте КуМир программа расстановки стен на поле Вездехода не соответствует целям его урока, что ему нужна другая расстановка. Поскольку программа расстановки стен для Вездехода (как и для Робота) написана на алгоритмическом языке, она может быть легко изменена одним из учеников по указанию учителя. Причем можно модифицировать уже имеющуюся программу, а не писать ее "с нуля".

А как быть, если у учителя возникла идея нового учебного исполнителя? Одно решение — написать на алгоритмическом языке информационную модель нового исполнителя на базе существующих. Это может быть сделано прямо в системе КуМир, и она даже поможет (если вы этого пожелаете) скрыть детали вашей реализации от учеников. Другое решение — реализовать нового исполнителя на языке Си с соблюдением определенных правил так, чтобы КуМир мог подключить его при запуске.

Третий методический аспект — введение новых типов величин.

А теперь предположим, что учитель решил поработать с комплексными числами. Можно, конечно, задавать комплексные числа в виде пары вещественных. Но можно — и это будет гораздо лучше — запустить КуМир с подключением исполнителя Комплексные числа. При этом в справках (в *help*-файлах) появится информация о типе "комплексное число" и допустимых операциях над величинами этого типа; можно будет использовать алгоритмы с комплексными аргументами и результатами и др. При вводе алгоритмов будет обеспечен мгновенный полный синтаксический контроль, при пошаговом выполнении результаты присваиваний будут выводиться на поля и т.д. Но вся эта информация и все эти возможности появятся в КуМире только при условии загрузки исполнителя Комплексные числа. При работе над другими темами она не будет мешать.

Исполнитель Комплексные числа входит в состав комплекта КуМир. Однако учитель может вводить и свои типы величин. Например, для работы с вычетами целых чисел по модулю два можно ввести соответствующий новый тип. Правда, в этом случае придется программировать на Си с соблюдением определенных тре-

массового решения обучаемыми задач на ЭВМ, а с другой стороны — считали обычным делом создание специального программного обеспечения для поддержки курса, мы реализовали алгоритмический язык на ЭВМ, создав систему, названную “Е-практикум” [Е-практикум]. (“Е” — в честь автора языка А.П. Ершова. В отличие от реализации Путника, которая заняла одну ночь, “Е-практикум” оказался серьезной работой, и на его реализацию была потрачена целая неделя.) Конечно, нам при этом пришлось формализовать те части языка, которые не были формализованы в учебнике.

В результате статус алгоритмического языка резко изменился: теперь он стал ***одним из*** языков программирования, исчезла ненужная работа по ручному “переводу” алгоритмов на “настоящий” язык программирования, надуманное противопоставление алгоритмического языка языкам программирования, а алгоритмов — программам. Соответственно, в более поздних учебниках [Ершов—мы—Шень], [Инф] алгоритмический язык рассматривается как один из языков программирования. К 1987 году более развитые “Е-практикумы” (Е-86 и Е-87) были реализованы на всех школьных ЭВМ (Д.В. Варсанофьев, А.Г. Дымченко, А.Г. Леонов, М.Г. Эпиктетов). А затем на IBM PC была реализована полноценная и полномасштабная система программирования на школьном алгоритмическом языке, получившая название КуМир (реализация на IBM PC — М.Г. Эпиктетов, адаптация на УКНЦ — Г.В. Лебедев).

Е4. История возникновения системы КуМир

К 1987 году учебная система программирования “Е-практикум” и ее более поздние версии “Е-86” и “Е-87” были реализованы на мини-ЭВМ СМ-4 и на всех школьных ЭВМ: на Ямахе, Корвете, УКНЦ и даже на БК-0010. Системы эти позволяли учителю работать по учебникам [Ершов], организуя на ЭВМ практикум по составлению программ на школьном алгоритмическом языке. Однако все эти “Е-практикумы” были чисто учебными — отсутствовали файловый ввод-вывод и графика, а длина программы была ограничена тремя десятками строк. Школьники, интересующиеся информатикой и программированием, быстро выполняли в “Е-практикуме” учебные задания и переключались на Бейсик (реже—Паскаль), чтобы написать свои собственные программы, выгодно отличающиеся от учебных тем, что они

а) имели длину аж до 100 и более строк;
б) позволяли нарисовать что-то графическое на экране;
в) давали возможность в начале работы что-то прочесть из файла, а по окончании работы что-то записать в файл.

Желание писать не только учебные, но и миниатюрные “настоящие” программы вполне естественно. Мы сами по опыту использования “Е-практикума” во вводном курсе программирования на 1-м курсе мехмата МГУ поняли, что для поддержки курса информатики в школе и начального курса программирования в вузе больше подходит не чисто учебная, а учебно-производственная система, которая

— построена на развитии школьного алгоритмического языка;
— допускает динамическое подключение Робота, Чертежника и других внешних исполнителей;
— наследует удобный интерфейс “Е-практикума”: ввод конструкций нажатием одной-двух клавиш, мгновенная диагностика ошибок при наборе программы, отслеживание ошибок в процессе выполнения программы, пошаговая отладка с выдачей на поля результатов присваиваний и пр.;
— не уступает Бейсику по производственным характеристикам.

К этому моменту для курса “Основы программирования” на мехмате МГУ уже была разработана и эксплуатировалась учебно-производственная система программирования ФортранМир на основе языка Фортран77 для ЭВМ Электроника-85 и PDP-11/70 (В.В. Борисенко, Д.В. Варсанофьев, А.Г. Дымченко). Система унаследовала и улучшила интерфейс “Е-практикума” и на ЭВМ с памятью 128К выигрывала у Бейсика по производственным показателям. Словом, было ясно, что учебно-производственный “Е-практикум”, этакий “Русский Бейсик”, в принципе сделать можно. В то время в образовании только-только начали появляться ЭВМ типа IBM PC, и мы решили разрабатывать новую систему на IBM PC, но так, чтобы потом ее можно было “перенести” на “массовые” школьные ЭВМ Ямаха и УКНЦ. (Перенос, а точнее, переделка системы на УКНЦ позже действительно состоялись, однако производственной системы на УКНЦ не получилось. Перенос на Ямаху даже не рассматривался — пришли другие времена.)

Вначале новая система называлась “Гамма-практикум” — продолжение мехматовской серии, состоящей из “Альфа-практикума”, “Бета-практикума” и “С-практикума” (практикум по структурам данных), разработанных неформальной группой “Аттик” мехмата МГУ (из которой позже и возникло объединение ИнфоМир). Потом новую систему стали называть “Мега-Е”, чтобы подчеркнуть преемственность с “Е-практикумом”. Затем тогдашний директор объединения ИнфоМир А.И. Левенчук в рамках приведения “к общему окончанию” всех поставляемых ИнфоМиром систем предложил распространить название КуМир — Комплект Учебных Миров — не только на набор исполнителей (Робот, Двуног, Вездеход и пр.), но и на саму систему программирования. А.Г. Кушниренко некоторое время выступал против, но название быстро распространилось и прижилось.

На первом курсе мехмата МГУ системой КуМир пользуются начиная с 1992 г. для составления программ по алгебре, анализу, геометрии и программированию. КуМир применяется также во многих школах и педвузах для поддержки либо всего курса школьной информатики, либо темы “Основы алгоритмизации”. С 1995 года многие школы используют КуМир-Гипертекст в курсе “Информационная культура”. Основные черты и особенности КуМира таковы:

придумывали” язык С, а авторы Бейсика сами “допридумывали” его до True Basic.

Я еще раз хочу обратить ваше внимание, что наш школьный алгоритмический язык лежит в той же ветви, где находятся все современные процедурные языки, — это основное и доминирующее течение в “большой информатике”. И в школьном курсе мы идем по основному пути, по которому идет весь мир.

Конструирование типов и структур данных.

И последняя развилка, которую я вам обязан показать, — единственная развилка, где мы не пошли тем путем, “которым идет весь мир”.

Есть еще одна классификация, по которой делят языки программирования, — это наличие в языке способов конструирования типов, возможности определять свои типы и структуры данных.

Такие механизмы были впервые предложены Н. Виртом в Паскале (и это та причина, по которой язык Паскаль столь знаменит). В Паскале я могу написать, что у меня в программе будут объекты (величины) ***типа,*** например, **`день недели`**. Что величина этого типа может принимать значения **`понедельник, вторник, среда, четверг, пятница, суббота, воскресенье`**. Таков простейший способ конструирования нового типа — явное перечисление всех возможных значений. Есть и другие способы — вы можете с ними познакомиться, изучив Паскаль.

Естественно, во всех современных языках, которые я неоднократно называл выше, имеются способы создания, описания, конструирования новых типов данных.

Школьный алгоритмический язык (в том виде, как он используется в учебнике) таких способов конструирования данных не содержит. Да и вся область “конструирования типов и структур данных” в учебнике совершенно не затронута. Учебник не содержит даже намеков на то, что такая область существует. В этом смысле школьный алгоритмический язык попал в группу старых языков, таких, как Бейсик, Фортран, Алгол. И это единственная развилка, я подчеркиваю, где мы пошли не по основному пути. Пошли отчасти сознательно, решив, что конструирование данных — отдельная область, которая в заданный нам объем школьного курса просто не уместилась, поскольку в число четырех фундаментальных понятий информатики не входит.

Но при любом расширении курса, увеличении его объема, при преподавании любого более углубленного курса информатики или при проведении факультатива по информатике конструирование типов и структур данных — кандидат номер один на включение.

И чтобы вы себе представляли чуть больше, я замечу, что в этой области сейчас происходит такое “раскручивание”, что ли. Нулевой уровень — нет никаких способов конструирования, типы данных фиксированы (как **`целый, вещественный, целочисленная таблица`** и пр. в школьном языке). Первый уровень — есть способы конструирования данных, но сами способы (как “перечисление”, “отрезок”, “запись” в Паскале) фиксированы в языке. Следующий уровень — языки, в которых есть способы конструирования способов конструирования, т.е. можно придумывать новые способы построения типов. И так далее. Разница примерно такая же, как между

а) сбором мебели из готовых деталей;
б) сбором мебели из деталей, которые мы можем сами изготавливать с помощью фиксированного набора инструментов;
в) сбором мебели из деталей, которые мы можем изготавливать с помощью инструментов, которые мы, в свою очередь, можем изготавливать с помощью “инструментов для изготовления инструментов” и т.п. Тут имеется такое естественное развитие с переходом с уровня на уровень, пока дело не дойдет до “универсальных инструментов”, с помощью которых можно изготавливать все, что угодно, в том числе и новые универсальные инструменты.

Наш курс соответствует нулевому уровню в этой классификации. Мы считаем, что первый уровень (уровень типов данных в Паскале) также может быть включен в школьный курс при увеличении его объема. Остальные уровни — это предмет специальной (уже не общеобразовательной) подготовки.

Повторю, что если в других вопросах мы пошли по основному, “магистральному” пути развития информатики, то в данном вопросе дело обстоит иначе. Мы эту область вообще не затронули, хотя такое направление довольно бурно развивается и уже имеет массу своих “поднаправлений”, ответвлений и нюансов.

Другие развилки.

Наконец, существует целый ряд, скажем так, “специальных” направлений и “развилок”. Например, языки процедурной традиции можно классифицировать по тому, содержат ли они средства организации параллельных процессов (см. описание параллельных процессов). В современных языках, как правило, есть способы организации и обработки исключительных ситуаций (называемых также прерываниями), о которых мы не упомянули вовсе (см., например, язык Ада).

Да и вообще методы алгоритмизации, способы описания действий, конструкции языков программирования непрерывно совершенствуются и развиваются. Многое из того, что здесь излагается, уже через несколько лет может показаться архаичным или, наоборот, широко известным и не заслуживающим специального внимания.

Но с точки зрения целей нашего курса все эти развилки второстепенны. Конечно, мы при всем желании не можем охватить все стороны и аспекты информатики, даже столь фундаментальные и существенные, как, например, параллельные процессы. Поэтому, повторяя, что в курсе изложены все четыре фундаментальных понятия информатики, я хочу, чтобы вы тем не менее прекрасно отдавали себе отчет, что “настоящая” информатика — огромная область со множеством даже не затронутых и не названных нами подразделов, направлений и пр.

Таково место нашего курса в информатике вообще. Кстати, если вы посмотрите на учебник [Каймин], то увидите: там предпринята попытка слегка захватить логическую традицию. А свердловчане [Гейн], как и мы, остались в рамках процедурной традиции, но сделали акценты на построении математических моделей. Соответственно, они сочли возможным использовать Бейсик, поскольку для их задач он ничем не был плох. Наоборот, те, кто преподает на Паскале, как правило, достаточно глубоко затрагивают область конструирования типов и структур данных — область, безусловно, во всех отношениях достойную изучения. Но наш курс преследует другие цели — я вам их формулировал с самого начала, и построен он, соответственно, под эти цели.

D3. Место курса в школе

Теперь о месте нашего курса в школе. Мы как авторы учебника считаем, что у него нет развития. Сам учебник можно переписать, сделать лучше, более интересным, расширить и т.д., но по набору понятий он закончен. После нашего курса можно двигаться дальше, но это движение, на наш взгляд, будет носить не фундаментальный, не общекультурный, а специальный (обычно "программистский") характер.

Вот стандартное продолжение нашего курса в специальные области (для факультативов, более глубоких курсов, для вузов и пр.).

1. Языковые курсы, расширяющие кругозор учащихся за счет изучения других языков (Паскаль, Си, Бейсик, Модула, Алгол-68 и т.д.). Если продолжать говорить о развитии мышления, то я бы особо выделил Алгол-68 (язык, на котором, по-моему, невозможно программировать, но который крайне полезен с точки зрения заложенных в нем идей и который был прообразом многих других языков), а также C++ и Smalltalk. Я вам говорил, что существует объектно-ориентированное программирование. Сейчас это очень популярное словосочетание, и все обычно ссылаются на Smalltalk как на пример того, что понимается под объектно-ориентированным программированием.

Есть языки не из процедурной традиции — Лисп, Рефал; в процедурной традиции интересен язык под названием Форт. Все это языки, безусловно достойные изучения.

Но изучение языков — скорее профессиональная подготовка либо факультатив. Я не думаю, что это необходимо в базовой школе, даже если говорить только о Паскале.

2. Второе направление — способы конструирования типов и структур данных. Тут можно начать с Паскаля и закончить стеками, деками, деревьями, графами и пр. Эта часть, безусловно, входит в общее программистское образование. Ее можно найти в нашем вузовском учебнике [ПдМ], а также в массе другой литературы. Обычно в названиях таких книг фигурируют слова "структуры", "данные" или что-нибудь в этом духе. Это либо второе направление факультатива или более глубокого программистского курса, либо часть вузовского курса.

3. Следующее направление — развитие § 16. Я могу назвать его "методы алгоритмизации" — это огромная область, которая почти не связана с языками программирования.

Материал для этого направления можно взять из нашего вузовского учебника [ПдМ]. Кроме того, есть разные другие хорошие и очень хорошие книги, например, [Вирт], [Грис], целый ряд книг [Дейкстра] и др.

В основном это область, как-то примыкающая к так называемому "доказательству правильности программ", где математические методы применяются для доказательства правильности алгоритмов и программ. Я вам отчасти это демонстрировал (помните "ловкость рук — и никакого мошенства"?), когда мы какими-то рассуждениями доказали, что программа правильная и получится то, что надо, но так и не поняли, как же она будет работать.

Это отдельная, достойная изучения область, и книжка [Вирт] очень хороша для введения. Но, как вы могли почувствовать, область сильно математизирована. И наше "Программирование для математиков" — хорошее название, отражающее существо дела. Для тех, кто силен в математике, изучение этой области (методов алгоритмизации и доказательств правильности программ) может привести к огромному продвижению вперед, даже если говорить о чисто практической (технической) работе по составлению алгоритмов и программ.

4. И, наконец, четвертое направление — способы конструирования языков программирования, управляющих и иных языковых конструкций. Это уже не для факультатива, а чтобы вы себе представляли направления развития науки информатики. Сейчас пока это научное направление — книжек я вам порекомендовать не могу (их просто нет), и преподавать эту область как какой-то курс еще рано.

Но посмотрите, что происходит. Когда у нас есть какие-то действия и мы их часто используем, мы один раз задаем вспомогательный алгоритм с параметрами, а потом пишем только его вызов, только имя. Что это имя значит — его "определение", — записано в виде вспомогательного алгоритма. Другими словами, у нас есть средства конструирования *своих* действий. Аналогично в Паскале есть средства конструирования *своих* типов данных. В школьном языке у нас есть средства конструирования *своих* исполнителей. А как насчет *своих* управляющих конструкций и *своих* конструкций алгоритмов и тому подобных элементов языка программирования? Ведь до сих пор мы считали заданными и неизменными все эти **нц...пока...кц**, **если...то...иначе...все**, **алг...нач...кон** и пр. Как, впрочем, до Паскаля считали неизменными в языке типы данных.

Оказывается, можно предложить такие способы конструирования управляющих конструкций языка, что можно будет задавать *свои* конструкции, например, **выбор**, в котором варианты анализируются снизу вверх или в случайном порядке; циклы, в теле которых записываются варианты и выполняется какой-то один (эти две конструкции вы можете найти в книге [Дейкстра-1]), а также и любые другие. И задавать такие новые конструкции мы сможем примерно так, как задавали вспомогательные алгоритмы, а "условие окончания цикла" или

Аналогичным образом были реализованы исполнители Стековый калькулятор, Редактор слова и Резчик металла.

Появление Робота в его нынешнем виде.

Позже, когда мы начали адаптировать этот подход — управление внешними исполнителями — для школьников, в ходе написания промежуточного учебного пособия [Ершов—мы—Шень] А.Х. Шень предложил упростить Путника, оставив ему для движения команды Резчика металла (**вправо, влево, вверх, вниз**), убрав ориентацию и стороны света. В соответствии с принципом "ничего лишнего" этот подход был принят. Одновременно, ввиду развития техники, появления персональных компьютеров, экранов и пр., мы решили расположить препятствия (стены) между клетками (чего раньше сделать не могли из-за технических ограничений печати на АЦПУ).

Так родился Робот. И именно в этом виде, уже без всяких изменений, он перекочевал в наш школьный учебник [Инф] и в "Алгоритмику" [Шень].

Если сравнивать Робота с Путником, то я должен заметить, что и сам Робот, и алгоритмы, управляющие Роботом, выглядят проще. Путник — более сложный исполнитель, имеющий ориентацию и умеющий действовать относительно этой ориентации (например, узнавать, стена или свободно ***справа*** от Путника). С другой стороны (и как следствие), алгоритмы решения одних и тех же задач при использовании Путника записываются часто компактнее и красивее, чем при использовании Робота. Поэтому если Робот начинает казаться слишком простым и слишком игрушечным, то можно заменить его Путником и несколько повысить класс решаемых задач.

И в заключение — еще несколько слов о Путнике, Роботе, Резчике металла и других клеточных исполнителях. Я думаю, что любая попытка обучить основам алгоритмизации, не пользуясь математическим материалом и числами, с неизбежностью приводит к использованию того или иного исполнителя, обитающего в "клетчатом мире". До начала перестройки мы мало знали о том, что делается в преподавании информатики за рубежом, в частности, не были знакомы с роботом по имени Карел [Паттис]. По-видимому, все, кто пытался решить эту задачу, приходили примерно к одному и тому же результату.

E3. Как возник школьный алгоритмический язык

Сказать по правде, если бы я выбирал язык для преподавания основ информатики и если бы до этого никакого языка в школах не использовалось, то я бы выбрал "псевдокод" из нашего вузовского учебника [ПдМ]. Дело в том, что "псевдокод" возник так же, как и Путник. Начав писать на первых занятиях "сделать шаг" без всякого Фортрана, мы просто вынуждены были в момент появления управляющих конструкций написать по-русски:

```
если впереди свободно
  |  то сделать шаг
конец если
```

или

```
цикл пока впереди свободно
  |  выполнять сделать шаг
конец цикла
```

Другими словами, как только мы стали писать, "как надо", чтобы поменьше объяснять студентам и чтобы все было само понятно, у нас совершенно естественно возник русский "псевдокод". И лишь после этого мы реализовали сначала простой язык программирования без переменных "Мини" (это сделали — тогда второкурсники — братья А. и Ю. Прилипко под руководством А.А. Веденова), а потом и более содержательные системы программирования на основе "псевдокода".

Школьный алгоритмический язык родился по-другому. Когда в 1985 году было принято решение ввести курс информатики в школах, Андрей Петрович Ершов выбрал в качестве основы для преподавания широко распространенный в то время в научной литературе язык Алгол-60 (что и отразилось в названии: "алгоритмический язык" — это дословный перевод английского *Algorithmic Language*, сокращением от которого является Algol). Поскольку, однако, настоящий Алгол достаточно сложен и не приспособлен для школы, было выбрано очень небольшое подмножество языка (например, без понятия блочной структуры), которое, кроме того, было русифицировано (так появились **алг**, **нач**, **кон** и пр.). Получился достаточно простой и понятный, но несуществующий язык.

В первых школьных учебниках [Ершов] этот язык, впрочем, и не считался языком программирования. Он рассматривался как некоторая нотация для записи алгоритмов, причем сами алгоритмы могли быть совершенно неформальными, формулы можно было записывать как в математике (без всякой линейной записи) и пр. Считалось, что алгоритмический язык — это язык для записи алгоритмов (в первых учебниках под алгоритмом понималось существо вычислений) и что при работе на ЭВМ алгоритм надо будет перевести (переписать) на один из языков программирования — Бейсик, Паскаль, Рапира. Таким образом, школьный алгоритмический язык (как язык для записи алгоритмов) противопоставлялся языкам программирования. Правила записи алгоритмов были достаточно расплывчаты (ведь язык был неформальным) и требовали соблюдения формы только управляющих конструкций.

Летом 1985 года (напомню, что курс информатики был введен в школах с 1 сентября 1985 года) мы проводили на мехмате МГУ курсы подготовки учителей информатики, т.е. излагали содержание информатики (в том виде, в каком оно было в первых учебниках А.П. Ершова) учителям математики и физики, которых "призвали" в преподаватели информатики. Поскольку к этому времени мы уже, с одной стороны, не представляли себе интенсивного курса информатики без

E2. Как возник Робот

Это отступление будет посвящено истории возникновения школьного Робота в том виде, в котором он используется в учебнике. История эта началась в 1980 году, когда мы еще только-только приступали к преподаванию нового курса программирования на механико-математическом факультете МГУ (раньше в качестве курса программирования на мехмате преподавался ассемблер для ЕС ЭВМ (аналог IBM 360) и язык Фортран).

Надо сказать, что мы к тому времени уже осознали роль исполнителей (этим термином мы тогда обозначали пакет программ, работающих над общими данными, а вовсе не "игрушечных" внешних исполнителей типа Робота). Более того, мы понимали, что это понятие первично и более важно, чем понятие подпрограммы, величины (переменной, объекта) и пр.

С другой стороны, мы считали, что студенты должны работать — составлять программы. И мы поставили перед собой фантастическую по тем временам цель — студент должен уйти с ***первого*** занятия с выполненной программой. Напомню, что речь идет о времени, когда программы еще набивались на перфокартах, потом отдавались на обработку на ЭВМ в так называемом ***пакетном режиме***, а результаты распечатывались на АЦПУ. Обычно набитую на перфокартах программу студент получал на следующий день после сдачи текста, отдавал программу на выполнение, после чего на следующий день получал результаты, сдавал в набивку тексты перфокарт, в которые надо было внести изменения, и т.п. Над дверью перфораторной висел лозунг: "Если вам не то набили — обратитесь, вам добьют". На все эти технические сложности накладывались содержательные — ведь для составления даже самой простой программки на Фортране надо пройти массу материала, который в один урок никак не поместится.

Поэтому мы решили создать пакет (библиотеку) программ на Фортране, который можно будет быстро и легко объяснить, с тем чтобы студенты составили лишь основную программу, состоящую только из вызовов подпрограмм. В качестве такого пакета программ на Фортране — исключительно для простоты объяснения и постановки задач (по принципу "ничего лишнего") — мы придумали пакет программ "Путник", описание которого вы можете найти в [ПдМ] и который отличался от Робота тем, что имел ориентацию (шагал вперед, поворачивал налево и направо). Кроме того, препятствия у Путника, в отличие от стен на поле Робота, занимали клетку целиком (иначе их было невозможно напечатать на АЦПУ). Кроме того, мы решили заранее заготовить перфокарты с базовыми командами Путника (на Фортране), т.е. перфокарты вида

```
CALL STEP; Путник.сделать шаг
```

чтобы студенту достаточно было набрать из уже готовых, набитых перфокарт свою программу и отнести ее на ЭВМ. Заметьте, что текст справа — всего лишь комментарий для человека. ЭВМ будет понимать и выполнять вызов подпрограммы на Фортране. Но мы решили, что "существо" дела — как ЭВМ выполняет такие программы — мы объясним потом. Пока же главное — дать студентам возможность выполнить программу и унести домой распечатку с ответом.

Это все было осознанным построением курса. А вот дальше, как говорят, "не было бы счастья, да несчастье помогло". Многое в жизни происходит случайно или по каким-то техническим причинам. Чтобы пропустить большой поток студенческих программ, мы использовали в то время систему АСФОР, созданную студентами факультета под руководством В.Б. Бетелина, которая работала в сотни раз быстрее стандартного программного обеспечения (в рамках стандартного обеспечения наши цели были в принципе недостижимы). И то ли АСФОР оказался недостаточно хорош для введения новых библиотечных подпрограмм, то ли у меня в ту пору не хватило знаний и умений, но я как сейчас помню вечер накануне первого выхода студентов на ЭВМ, когда я — после двух суток борьбы с АСФОРом — с абсолютной ясностью понял, что создать библиотеку программ Путника до утра я не успею. А ведь в 9 часов утра первая группа придет на первый семинар — и что мы будем делать? Я думаю, вы очень живо можете представить себе мое состояние в тот момент. Апатия, отчаяние и лихорадочное "ну что же делать?".

Отход от Фортрана.

И именно тогда — под воздействием обстоятельств и понимая полную уже несовместимость наших целей и реального программного обеспечения — я вдруг понял, что для выполнения программ, которые наши студенты будут составлять на первом занятии, никакой Фортран не нужен — я могу написать программу-интерпретатор, и она все выполнит. Поскольку речь шла о линейных программах без управляющих конструкций, мне пришлось написать программу-интерпретатор типа алгоритма А91 из учебника, реализовать информационную модель исполнителя Путник и написать подпрограммы, реализующие ввод студенческой программы с перфокарт и распечатку программы и результата ее работы на АЦПУ (на последнее ушло больше всего времени, потому что надо было сделать так, чтобы на каждого студента тратился один и ровно один лист АЦПУ, чтобы он останавливался на сгибе бумаги, чтобы его было удобно отрывать и т.п.).

Сейчас, по прошествии времени, я думаю, что именно тогда впервые мы пожертвовали существующим программным обеспечением ради сохранения целей курса. Т.е. в ситуации выбора и ценности были выбраны и сохранены цели курса, а не Фортран.

Дальше ситуация развивалась стремительно и абсолютно естественно. Коль скоро программа перестала быть программой на Фортране, из текста перфокарт немедленно исчезли все "лишние" символы, все эти **`Call step`** и прочие фортрановские штучки. Осталось лишь самое необходимое — "Сделать шаг". Программа приобрела простой, читаемый русский вид, ***перестала быть набором вызовов библиотечных подпрограмм*** на Фортране, а превратилась просто в программу управления неким внешним устройством.

"тело цикла" при этом становятся параметрами, как аргументы или результаты для алгоритмов.

Это как если бы мы с вами строили разные здания из строительных блоков, а потом обнаружили, что и сами блоки можно собирать из более мелких "кирпичиков". Или как в физике от молекул переходят к атомам и кваркам. Оказывается, что и в программировании уже обнаружены свои "атомы", "кварки" и прочие частички, из которых можно собирать все эти **`если...то...иначе`**, циклы и пр.

Думаю, лет через 10 это все станет нормой, войдет во все языки, появятся литература, учебники и пр. Сейчас же это просто одна из зон развития. Но мне кажется полезным, чтобы вы посмотрели на конструкции школьного алгоритмического языка с этой точки зрения — просто как на удобные "блоки", собираемые из каких-то "кирпичиков". Чтобы вы подумали над возможностями программирования, если вы можете, например, определить такую конструкцию цикла:

```
нц для каждой пары (x, y),
|       удовлетворяющей условию u (x, y)
| тело цикла (x, y)
кц
```

— и потом использовать ее по мере надобности с разными условиями и разными "телами" так же, как вы использовали цикл **`пока`**. Чтобы вы подумали о том, какие новые возможности "накопления знаний" открывает такой подход.

Как вы видите, все три основных продолжения, развития нашего курса носят программистский, профессиональный или предпрофессиональный характер. Именно это я имел в виду, когда говорил, что наш курс как базовый общеобразовательный курс информатики в школе полон. Кроме него, в базовой школе можно использовать компьютеры в других предметах. Но сам курс закончен и полон.

D4. О программном обеспечении курса

Я надеюсь, что у вас уже сформировалось достаточно полное представление о курсе, о его целях, об учебнике и общей методике построения курса. Я бы хотел обратить ваше внимание, что учебник и курс построены без привязки к конкретным компьютерам и конкретному программному обеспечению. Учебник написан исходя из того, каким он должен быть. В момент его написания никакого программного обеспечения под этот учебник вообще не было.

С одной стороны, как мне кажется, замечательно, что учебник написан "как надо", без подгонки под какие-нибудь особенности программного обеспечения.

С другой стороны, когда учебник впервые вышел из печати, он оказался в школах без какого-либо программного обеспечения вообще. Это, конечно, проблема, поскольку между выходом учебника и выходом соответствующего программного обеспечения проходит некоторое время.

Если курс опирается на Бейсик, то такой проблемы, конечно, не возникает. Бейсик есть у всех — и еще до начала написания учебника. Но я еще раз хочу напомнить вам основную цель нашего курса — развитие алгоритмического стиля мышления. Мы не могли пожертвовать этой целью в угоду какому бы то ни было программному обеспечению.

Программное обеспечение, разработанное специально для поддержки нашего курса, называется КуМир (название это историческое и первоначально означало набор учебных исполнителей — Комплект Учебных Миров). Сейчас оно используется и для самой системы программирования на основе школьного алгоритмического языка.

Хотя система КуМир разрабатывалась специально как современная реализация школьного алгоритмического языка, эта система — "настоящая", в отличие от старого "Е-практикума", который являлся чисто учебной системой. Если в старом "Е-практикуме" имелась возможность лишь ввести и выполнить простейший алгоритм из 30—50 строк, то при разработке КуМира были поставлены (и достигнуты) совершенно другие цели. КуМир — это полноценная, полномасштабная система программирования, в которой можно создать программу и в тысячу строк, в которой — на школьном алгоритмическом языке — можно реализовать и экранный редактор текстов, и простенькую базу данных, и электронные таблицы, и учебный компилятор, и прикладные системы по другим школьным предметам (например, геометрии или биологии).

Методическая цель, которую мы при этом преследовали, — дать школьникам возможность не только поработать с подобного рода системами, но и познакомиться с текстом реализации соответствующей системы, посмотреть, "как" это делается. Чтобы им можно было задать задачу типа: добавьте в редактор текста команду **`вставить строку`**; измените так-то и так-то базу данных и пр. И чтобы при этом они работали в той среде, к которой привыкли, на обычном школьном алгоритмическом языке.

Наконец, в каждом классе есть лидеры, решающие какие-то свои (не учебные) задачи. КуМир — это система, позволяющая решить любую такую задачу (даже если понадобится на физическом уровне управлять какими-то устройствами). В КуМире можно сделать все, что можно сделать на Бейсике, и даже больше того. И при этом работать в КуМире намного удобнее и эффективнее, чем на Бейсике. На мой взгляд, чрезвычайно важно, чтобы лидеры класса предпочли КуМир другим системам программирования при решении конкретных задач.

Вам, наверное, будет интересно узнать, что на механико-математическом факультете МГУ школьный алгоритмический язык используется весь первый курс. По расписанию на первом курсе лекций нет, а есть только практикум на ЭВМ, посвященный решению задач из алгебры, анализа, дифференциальных уравнений и других

предметов (например, “найти все делители нуля в кольце вычетов по модулю P”). Поэтому бывшим школьникам на первом же занятии говорят: “Решать будете на школьном алгоритмическом языке”. Примерно 20—30 процентов спрашивают: “Что это такое?” Мы им отвечаем: “Это вам должны были объяснить в школе”. И все — они успокаиваются (независимо от того, учили их или не учили): если в школе — значит, что-то простое.

Дальше они получают пособие [Валединский], где вначале приводятся конструкции алгоритмического языка, а потом просто идет список примерно из 200 задач. Все задачи не из информатики, а из курсов алгебры, анализа и пр. Обычно мы тратим 45 минут, чтобы написать на доске все конструкции алгоритмического языка, сказать, какими кнопками они вызываются, а также объяснить, как следует входить в систему, выходить из нее и т.д., после чего студенты просто весь семестр сидят за терминалами и решают задачи.

Это может быть вам интересно, потому что, как вы понимаете, мы это делаем не ради школьного языка. У нас совершенно другие цели — научить студентов решать задачи по алгебре, анализу и пр. Мы просто не можем тратить время на изучение языка — для этого надо какие-то лекции читать, еще что-то, а так мы говорим: “Школьному алгоритмическому языку вас должны были научить в школе. Не научили — берите в библиотеке школьный учебник или спрашивайте у товарищей. Ваше дело — уметь решать задачи, а на каком языке — никого не волнует”.

Некоторые при этом спрашивают: “А можно, я буду программировать на Си?” или: “А можно, я буду программировать на Бейсике?”. Я обычно говорю: “Можно, но вы просто по времени не успеете решить все задачи, потому что на запуск, поиск и исправление ошибки и следующий запуск вы потратите времени в десять раз больше, чем при работе на школьном языке в Е-практикуме”.

Стиль такой, что если они в течение года сдают, скажем, 20—30 задач по выбору преподавателя, то получают зачет-автомат. А если они зачет-автомат не получают, то на “настоящем” зачете они тянут билет с тремя задачами — и через час все задачи должны правильно работать на ЭВМ. Не успел пусть даже одну букву набрать или какую-то ошибку исправить — значит, не сдал, приходи в следующий раз, тяни новый билет и сдавай заново. В стандартных Бейсике, Паскале или Си мы просто не вправе были ставить такие требования.

Это немного в сторону, но я хочу еще раз подчеркнуть, что выбор языка и программной системы как ***цели обучения*** сильно отличается от выбора ***средств обучения***. Если цель — решать задачи по высшей алгебре, то средство должно быть удобным и не требовать времени на изучение. Если цель — изучить интересный язык программирования, то язык должен быть интересным и может быть сколь угодно сложным и неудобным.

Кроме того, уж коль скоро я заговорил о механико-математическом факультете МГУ, я хочу сказать, что наш курс “Программирование для математиков” [ПдМ] появился на мехмате до появления информатики в школах. Поэтому он стартовал с исполнителей Путник, Резчик металла, Редактор слова и простейших программ управления этими исполнителями. Сейчас этот материал спустился в школьный курс, и, конечно, теперь нецелесообразно повторять это на мехмате МГУ. Поэтому сейчас считается, что студенты уже умеют программировать, не важно, на каком языке, а в курсе программирования им излагаются структуры данных (стеки, деки, деревья и пр.), а также решаются сложные математизированные задачи — например, задача удаления невидимых линий при изображении конечно-элементных моделей [ПдМ, с. 167—188].

И в заключение.

Всеми поставками программного обеспечения к нашему курсу занимается “ИнфоМир” (тел./факс 931-17-86, e-mail: infomir@math.msu.su). Системы, как вы видели, называются КуМир, АльфаМир, ФортранМир, ПаскальМир, МикроМир и т.д. Окончание “Мир” (отражающее принцип непосредственного редактирования информации “Человека в Мире объектов” [Лебедев]) является торговой маркой, признаком всех наших систем.

Программное обеспечение для УКНЦ, Корвета и другие “старые” версии объявлены свободно копируемыми, т.е. любая школа, институт усовершенствования учителей и пр. могут бесплатно получить старые версии и свободно их копировать. Новые системы, в частности, КуМир на IBM PC, пока продаются за деньги.

КОНЕЦ КУРСА ЛЕКЦИЙ

Е. ПОСЛЕСЛОВИЕ (РАЗНЫЕ ЗАМЕЧАНИЯ, ОТСТУПЛЕНИЯ, РЕКОМЕНДАЦИИ И ПР.)

Е1. Рекомендуемая литература

Учитель должен знать больше того, что он преподает школьникам. Поэтому прежде всего я настоятельно рекомендую вам потратить время и силы на изучение нашего вузовского учебника [ПдМ]. Я думаю, что те из вас, у кого хватит на это сил и упорства, как минимум будут абсолютно полно представлять себе, что и для чего написано в школьном учебнике, какие цели преследуют те или иные параграфы и пр. Конечно, наш вузовский учебник достаточно сильно математизирован, недаром он называется “Программирование для математиков”. Но вы можете пропустить доказательства и математические схемы в разделах, посвященных методам алгоритмизации (индуктивное вычисление функций на пространстве последовательностей, проектирование цикла с помощью инварианта и пр.), а также пролистать решения математизированных задач (построение выпуклой оболочки последовательно поступающих точек плоскости, “удаление” невидимых линий при изображении конечноэлементной модели и пр.).

Зато все, что есть в школьном учебнике, вы найдете и в вузовском (напомню, что это старый учебник, написанный до введения информатики в школах), причем на более глубоком уровне. Кроме того, темы, затронутые в школьном учебнике бегло (как-то инвариант цикла или использование и придумывание **дано/надо** при составлении алгоритмов), а также не затронутые вовсе (как конструирование типов и структур данных) в вузовском учебнике изложены достаточно подробно и, на мой взгляд, достаточно просто.

Наконец, вы найдете там массу примеров других исполнителей (Путник, Стековый калькулятор, Резчик металла и др.) и массу примеров программ разной сложности, включая такие, где используются 3—4 уровня вспомогательных исполнителей, придуманных в ходе решения задачи (т.е. вполне содержательных задач по управлению простейшими исполнителями). Этот материал вы можете использовать и на обычных уроках в ваших классах.

Я так подробно и долго рекомендую вам [ПдМ], поскольку это тот же самый (по целям и по подходу) курс, что и в школьном учебнике, но изложенный на гораздо более глубоком и содержательном уровне. (Хотя чисто полиграфически учебник [ПдМ] издан ужасно.) Все остальные книги, как бы полезны и замечательны они ни были, посвященны чему-то другому. Их можно и нужно изучать. Из них можно и нужно заимствовать то, что вы сочтете возможным изложить в вашем классе. Но это другие книги, посвященные другим темам или основывающиеся на ином подходе к информатике.

Кроме школьного учебника [Инф], вузовского учебника [ПдМ] и настоящего пособия, непосредственно нашему курсу посвящены ряд статей в журнале “Информатика и образование” [И&О] и пособие для учителя [Авербух]. Как я уже говорил, в этом пособии хорошо дополнена глава 2 учебника (“Устройство ЭВМ”), а также почти полностью приведены решения упражнений из главы 1 учебника. В частности и в особенности я обращаю ваше внимание на решение упражнений 6—8 на с. 175—177 пособия и замечание о “подводных камнях” (с. 176). На мой взгляд, этот материал, безусловно, следует задействовать на уроках. Но что касается главы 3 учебника, то в этом отношении методическое пособие [Авербух] очень слабое и содержит в основном рекомендации, а не решения.

Существуют также многочисленные комментарии, пособия и пр. по отдельным темам нашего учебника, написанные разными учителями, из которых я, пожалуй, выделю только [Зайдельман]. Кроме того, появились и некоторые книги, связанные с учебником опосредованно либо раскрывающие отдельные его темы, например, [Дрофа 9, 10].

Наконец, очень близкий к нашему учебнику подход используется в курсе “Алгоритмика” [Шень] для учащихся младших классов, а также в замечательной книге [Шень-2]. Кстати, те из вас, у кого есть компьютер, подключенный к Интернету, последнюю книгу могут найти и бесплатно загрузить к себе из Интернета.

Все вышеназванные книги так или иначе связаны с нашим учебником или нашим курсом. Учитель, конечно же, должен быть знаком со всеми курсами. И уж как минимум с [Гейн], [Каймин]. Я уже неоднократно хвалил вам учебник [Гейн]. Про качество учебника [Каймин] я вам (как автор конкурирующего учебника) ничего говорить не буду. Но, на мой взгляд, вы все равно должны его прочесть и, где возможно, понять.

Кроме того, даже если вы уже знакомы со всеми учебниками, я рекомендую вам еще раз их все перечитать, задавая себе постоянно один и тот же вопрос: “Каковы цели и методы данного курса? Что должен усвоить ученик и как это достигается?”.

Ну и, конечно, учитель должен быть знаком не только с учебной литературой. Здесь я прежде всего рекомендую вам книги, указанные в конце нашего учебника: “Знакомьтесь: компьютер” [ЗК] и “Возможности вычислительных машин и человеческий разум” [Вейценбаум]. Первая из этих книг популярна и очень легка для чтения. Вторая, наоборот, имеет философский характер и достаточно трудно читается. О самих книгах я уже говорил.

Из других книг я назову еще раз [Вирт], [Грис], [Дейкстра]. Учителю также следует быть знакомым с основными понятиями современных языков программирования, таких, как C++, Ада, Smalltalk.

Еще я хочу обратить ваше внимание, что в середине 90-х годов в жизни человечества произошло событие, роль которого трудно переоценить. Мировая сеть Интернет перестала быть достоянием узкого круга ученых и инженеров и стала доступна сотням миллионов рядовых граждан развитых и развивающихся стран. Компьютерное сообщество мгновенно, за 1—2 года, признало ряд важных общемировых стандартов, из которых прежде всего следует отметить:

1) язык подготовки гипертекстов HTML,
2) объектно-ориентированный язык программирования Java и
3) простенький язык программирования гипертекстов, по сложности сравнимый с Бейсиком, — JavaScript.

По этой тематике в сети Интернет есть много свободно копируемых материалов, а к моменту выхода нашей книжки, я надеюсь, появятся уже и книги на русском языке. Поэтому желательно, чтобы, помимо традиционных языков, указанных выше, учитель познакомился также с:

— языком и системой программирования Visual Basic фирмы Microsoft;
— языком подготовки гипертекстов HTML (на уровне, достаточном для составления гипертекста из одной головной страницы, которая ссылается на пару других страниц);
— языком программирования гипертекстов JavaScript (на уровне, достаточном для составления контрольных вопросов к простейшему учебному гипертексту);
— языком программирования Java (основные понятия объектно-ориентированных языков и основные понятая параллельного программирования — потоки и процессы).

Цифры "губар"	1	2	3	4	5	6	7	8	9	0
XII век	1	2	3	4	5	6	7	8	9	0
1197 год	1	2	3	4	5	6	7	8	9	0
1275 год	1	2	3	4	5	6	7	8	9	0
ок. 1294 года	1	2	3	4	5	6	7	8	9	0
1303 год	1	2	3	4	5	6	7	8	9	0
1360 год	1	2	3	4	5	6	7	8	9	0
1442 год	1	2	3	4	5	6	7	8	9	0

Вопросы и задания

1. Какие виды систем счисления вы знаете?
2. С применением каких систем счисления вы встречались?
3. Какой системой счисления пользуется ребенок, когда он на пальцах показывает, сколько ему лет?
4. Являются ли слова "разрядная", "поместная" синонимами для понятия "позиционная система счисления"? Придумайте свою непозиционную систему счисления.
5. Какая система счисления вам ближе: вавилонская или древнеегипетская? И почему?
6. Из межпланетного путешествия астронавты привезли описание климата, природных условий и внешнего вида жителей планеты Z. Вот это описание: "На планете Z нет никакой наземной растительности, почва каменистая. Очень много рек, озер и других водоемов. Поверхность планеты покрыта туманом, часто идут дожди. Растительный же и животный мир водоемов очень разнообразен. Жители планеты Z физически очень сильны, но внешне чрезвычайно отличаются от землян. У каждого жителя планеты Z семь беспалых конечностей, каждая из которых заканчивается подобием присоски, а тело скорее похоже на шар, поэтому невозможно сказать, где верх, а где низ в нашем понимании. На теле есть два глаза". Сделайте предположения о том, какая может быть система счисления у жителей на планете Z.
7. Очевидно, что знание истории систем счисления необходимо историку и археологу. А в каких еще областях деятельности человека необходимо знание систем счисления?
8. Используя древнеегипетскую систему счисления, прочитайте запись числа
9. Используя римскую систему счисления, выпишите цифры от 100 до 110.
10. Используя римскую систему счисления, запишите год своего рождения и текущий год.
11. Запишите числа от 11 до 29 в старой русской системе счисления. Чем объясняется различный порядок написания цифр в числах 23 и 13?
12. Запишите число 15 во всех системах счисления, о которых было рассказано в этой теме.

Тема 3
Позиционные системы счисления

Более подробно материал этой темы изложен в главе 1 второй части.

3.1. Базис, алфавит, основание

В рамках этой темы мы будем рассматривать позиционные системы счисления. Напомним определения, которые мы дали в предыдущих темах.

Определение 1. ***Система счисления*** — способ записи (изображения) чисел.

Определение 2. Символы, при помощи которых записывается число, называются ***цифрами***.

Надо различать понятия "вид цифры" и "значение цифры". Например, в римской системе счисления числа записываются при помощи цифр I, V, X, L, C, D, M. Эти цифры имеют следующие значения: I — 1, V — 5, X — 10, L — 50, C — 100, D — 500 и M — 1000. В десятичной системе используются цифры от 0 до 9.

Десятичная система счисления, которую мы знаем с детства, является позиционной. Значение цифры, то есть ее вклад в величину числа, зависит от позиции (разряда) этой цифры в записи числа. Так, в числе 452 цифра 4 означает четыре сотни, цифра 5 — пять десятков, цифра 2 — две единицы. Это число можно представить в следующем виде: $452=4\cdot100+5\cdot10+2$.

Определение 3. ***Системы счисления,*** в которых вклад каждой цифры в величину числа зависит от ее положения (позиции) в последовательности цифр, изображающей число, называются ***позиционными.***

Определение 4. ***Системы счисления***, в которых каждой цифре соответствует величина, не зависящая от местонахождения этой цифры в записи числа, называются ***непозиционными.***

Определение 5. Совокупность различных цифр, используемых в позиционной системе счисления для записи чисел, называется ***алфавитом*** системы счисления.

При рассмотрении позиционных систем счисления чрезвычайно важно понятие ***базиса системы счисления***.

Определение 6. ***Базис*** позиционной системы счисления — это последовательность чисел, каждое из которых задает значение цифры "по месту" или "вес" каждого разряда.

Пример 11. Выпишем базисы нескольких систем счисления.

Базис десятичной системы счисления: 1, 10, 10^2, 10^3, 10^4, ..., 10^n, ...

Базис двоичной системы счисления: 1, 2, 2^2, 2^3, 2^4, ..., 2^n, ...

Базис восьмеричной системы счисления: 1, 8, 8^2, 8^3, 8^4, ..., 8^n, ...

В более общем виде если для позиционной системы счисления базис можно записать в виде последовательных членов геометрической прогрессии: 1, P, P^2, P^3, ..., P^i, ..., — то такая система называется ***традиционной.***

Определение 7. Знаменатель P геометрической прогрессии, члены которой образуют базис традиционной системы счисления, называется ***основанием*** системы.

обозначает количество десятков и "вносит" в величину числа 30, а в числе 304 та же цифра 3 обозначает количество сотен и "вносит" в величину числа 300.

Системы счисления, в которых каждой цифре соответствует величина, не зависящая от ее места в записи числа, называются ***непозиционными.*** Примеры непозиционных систем счисления мы рассмотрим чуть позже.

Позиционные системы счисления — результат длительного исторического развития непозиционных систем счисления.

2.1. Единичная система

Потребность в записи чисел появилась в очень древние времена, как только люди начали считать. Количество предметов, например овец, изображалось нанесением черточек или засечек на какой-либо твердой поверхности: камне, глине, дереве (до изобретения бумаги было еще очень и очень далеко). Каждой овце в такой записи соответствовала одна черточка. Археологами найдены такие "записи" при раскопках культурных слоев, относящихся к периоду палеолита (10—11 тысяч лет до н.э.).

Ученые назвали этот способ записи чисел ***единичной ("палочной")*** системой счисления. В ней для записи чисел применялся только один вид знаков — "палочка". Каждое число в такой системе счисления обозначалось с помощью строки, составленной из палочек, количество которых и равнялось обозначаемому числу.

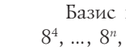
Неудобства такой системы записи чисел и ограниченность ее применения очевидны: чем большее число надо записать, тем длиннее строка из палочек. Да и при записи большого числа легко ошибиться, нанеся лишнее количество палочек или, наоборот, не дописав их.

Можно предположить, что для облегчения счета люди стали группировать предметы по 3, 5, 10 штук. И при записи использовали знаки, соответствующие группе из нескольких предметов. Естественно, что при подсчете использовались пальцы рук, поэтому первыми появились знаки для обозначения групп предметов из 5 и 10 штук (единиц). Таким образом, возникли уже более удобные системы записи чисел.

2.2. Древнеегипетская десятичная непозиционная система

В ***древнеегипетской*** системе счисления, которая возникла во второй половине третьего тысячелетия до н.э., использовались специальные цифры для обозначения чисел 1, 10, 10^2, 10^3, 10^4, 10^5 10^6, 10^7. Числа в египетской системе счисления записывались как комбинации этих цифр, в которых каждая из них повторялась не более девяти раз.

Пример 4. Число 345 древние египтяне записывали так:



где | — единицы, ∩ — десятки, 𓆼 — сотни.

В основе как палочной, так и древнеегипетской системы счисления лежал простой принцип сложения, согласно которому ***значение числа равно сумме значений цифр, участвующих в его записи.*** Ученые относят древнеегипетскую систему счисления к десятичной непозиционной.

2.3. Вавилонская шестидесятеричная система

Также далеко от наших дней, за две тысячи лет до н.э., в другой великой цивилизации — ***вавилонской*** — люди записывали цифры по-другому.

Числа в этой системе счисления составлялись из знаков двух видов: прямой клин 𒐕 служил для обозначения единиц, а лежачий клин — 𒌋 — для обозначения десятков. Число 32, например, записывали так: 𒌋𒌋𒐕𒐕. Знаки 𒐕 и 𒌋 служили цифрами в этой системе. Число 60 снова обозначалось тем же знаком 𒐕, что и 1, этим же знаком обозначались и числа $3600=60^2$, $216\,000=60^3$ и все другие степени 60. Поэтому вавилонская система счисления получила название ***шестидесятеричной***.

Для определения значения числа надо было изображение числа разбить на разряды справа налево. Новый разряд начинался с появления прямого клина после лежачего, если рассматривать число справа налево.

2-й разряд 1-й разряд

Значение числа определяли по значениям составляющих его цифр, но с учетом того, что цифры в каждом последующем разряде значили в 60 раз больше тех же цифр в предыдущем разряде.

Пример 5. Число $92=60+32$ записывали так: 𒌋𒌋𒐕𒐕, а число 444 в этой системе записи чисел имело вид 𒐕𒐕𒐕𒐕𒌋𒌋𒐕𒐕, т.к. $444=7\cdot60+24$.

Исключительно для наглядности мы разделили пробелом (которого не было у вавилонян) старший разряд (левый) и младший.

Все числа от 1 до 59 вавилоняне записывали в десятичной непозиционной системе, а число в целом — в позиционной системе с основанием 60.

Запись числа у вавилонян была неоднозначной, т.к. не существовало цифры для обозначения нуля. Запись числа 92, приведенная выше, могла обозначать не только $92=60+32$, но и, например, $3632=3600+32=60^2+32$. Для определения абсолютного значения числа требовались дополнительные сведения. Впоследствии вавилоняне ввели специальный символ для обозначения пропущенного шестидесятеричного разряда — 𒁹, что соответствует появлению цифры 0 в записи десятичного числа.

Пример 6. Число 3632 теперь нужно было записывать так: 𒐕𒁹𒌋𒌋𒐕𒐕. Но в конце числа этот символ обычно не ставился, т.е. этот символ все же не был цифрой "ноль" в нашем понимании, и опять же требовались дополнительные сведения для того, чтобы отличить 1 от 60, от 3600 и т.д.

Таблицу умножения вавилоняне никогда не запоминали, т.к. это было практически невозможно. При вычислениях использовались готовые таблицы умножения.

Шестидесятеричная вавилонская система — первая известная нам система счисления, частично основанная на позиционном принципе.

Система вавилонян сыграла большую роль в развитии математики и астрономии, ее следы сохранились и до наших дней. Так, мы до сих пор делим час на 60 минут, а минуту на 60 секунд. Следуя примеру вавилонян, мы и окружность делим на 360 частей (градусов).

2.4. Римская система

Знакомая нам ***римская*** система не слишком принципиально отличается от египетской. В ней для обозначения чисел 1, 5, 10, 50, 100, 500 и 1000 используются заглавные латинские буквы I, V, X, L, C, D и M соответственно, являющиеся цифрами этой системы счисления. Число в римской системе счисления обозначается набором стоящих подряд цифр. Значение числа равно:

1) сумме значений идущих подряд нескольких одинаковых цифр (назовем их группой первого вида);

2) разности значений двух цифр, если слева от большей цифры стоит меньшая. В этом случае от значения большей цифры отнимается значение меньшей цифры. Вместе они образуют группу второго вида. Заметим, что левая цифра может быть меньше правой максимум на один порядок: так, перед L (50) и C (100) из "младших" может стоять только X (10), перед D (500) и M (1000) — только C (100), перед V (5) — только I (1);

3) сумме значений групп и цифр, не вошедших в группы первого или второго вида.

Пример 7. Число 32 в римской системе счисления имеет вид XXXII=(X+X+X)+(I+I)=30+2 (две группы первого вида).

Пример 8. Число 444, имеющее в своей десятичной записи 3 одинаковые цифры, в римской системе счисления будет записано в виде CDXLIV=(D—C)+(L——X)+(V—I)= 400+40+4 (три группы второго вида).

Пример 9. Число 1974 в римской системе счисления будет иметь вид MCMLXXIV=M+(M—C)+L++(X+X)+(V—I)=1000+900+50+20+4 (наряду с группами обоих видов в формировании числа участвуют отдельные "цифры"). Наверняка вы видели подобные обозначения года выпуска в титрах голливудских фильмов.

2.5. Алфавитные системы

Более совершенными непозиционными системами счисления были ***алфавитные*** системы. К числу таких систем счисления относились ***славянская***, ***ионийская (греческая)***, ***финикийская*** и другие. В них числа от 1 до 9, целые количества десятков (от 10 до 90) и целые количества сотен (от 100 до 900) обозначались буквами алфавита. Алфавитная система была принята и в Древней Руси. Числа от 1 до 9 записывали так:

А̃ В̃ Г̃ Д̃ Е̃ S̃ З̃ И̃ Ѳ̃ І̃

Над буквами, обозначавшими числа, ставился специальный знак "~" — титло. Это делалось для того, чтобы отличить числа от обычных слов:

А̃І = 11, **В̃І** = 12, **Г̃І** = 13, ..., **Ѳ̃І** = 19, **К̃**=20, **КА̃** = 21.

Интересно, что числа от 11 (один — над десять) до 19 (девять — над десять) записывали так же, как говорили, то есть цифру единиц ставили до цифры десятков. Если число не содержало десятков, то цифру десятков не писали.

Удобны ли алфавитные системы?

Пример 10. Запишем в славянской записи числа 444 и 32:

444 = **У̃МД̃**

32 = **К̃В̃**

Мы видим, что запись получилась не длиннее нашей десятичной. Это объясняется тем, что в алфавитных системах использовалось по крайней мере 27 цифр. Но эти системы были удобны только для записи целых чисел от 1 до 1000.

Правда, славяне, как и греки, умели записывать числа и превосходящие 1000. Для этого к алфавитной системе добавлялись новые обозначения. Так, например, числа 1000, 2000, 3000 ... записывали теми же цифрами, что и 1, 2, 3, ..., только перед цифрой слева снизу ставили специальный знак:

1000 = **͵А̃** 2000 = **͵В̃** 3000 = **͵Г̃**

Число 10 000 обозначалось той же буквой, что и 1, только без титла. Ее уже обводили кружком: 10 000 = Ⓐ. Называлось это число "тьмой". Отсюда и произошло выражение "тьма народу".

Таким образом, для обозначения "тем" (множественное число от слова "тьма") первые 9 цифр обводились кружками:

20 000 = Ⓑ 30 000 = Ⓖ 40 000 = Ⓓ

10 тем, или 100 000, было единицей высшего разряда. Ее называли "легион". 10 легионов составляли "леорд" и т.д. Самая большая из величин, имеющих свое обозначение, называлась "колода", и равнялась она 10^{50}. Считалось, что "боле сего несть человеческому уму разумевати".

Такой способ записи чисел можно рассматривать как зачатки позиционной системы, т.к. в нем для обозначения единиц разных разрядов применялись одни и те же символы, к которым лишь добавлялись специальные знаки для определения значения разряда.

Алфавитные системы счисления были мало пригодны для оперирования с большими числами. И, как уже говорилось, со временем они уступили место позиционным системам.

2.6. Индийская мультипликативная система

Системы счисления, основанные на позиционном принципе, возникли независимо друг от друга сразу в трех местах: в Древнем Междуречье (Вавилоне), у племени майя и, наконец, в Индии. И не случайно.

Каковы же были предпосылки для создания позиционной системы? Что привело людей к этому замечательному открытию?

Чтобы ответить на эти вопросы, мы снова обратимся к истории. В Древнем Китае, Индии и в некоторых других странах существовали системы записи, построенные на ***мультипликативном*** принципе.

Пусть, например, десятки обозначаются символом *X*, а сотни — *Y*. Тогда запись числа 323 схематично будет выглядеть так:

3*Y* 2*X* 3.

В таких системах для записи одинакового числа единиц, десятков, сотен или тысяч применяются одни и те же символы, но после каждого символа пишется название соответствующего разряда. С использованием введенных обозначений число 100 можно записать в виде 1*Y*.

Чуть позже перестали писать названия разрядов, и это стало следующей ступенью к позиционному принципу (подобно тому как мы пишем "320", а не "3 сотни, 2 десятка"). Но при записи чисел по такой системе очень часто требовался символ для обозначения отсутствующего разряда.

2.7. Появление нуля

Современная десятичная система счисления возникла в Индии приблизительно в V веке н.э. Возникновение этой системы стало возможным после величайшего изобретения — цифры 0 для обозначения отсутствующей величины.

Как же появился ноль?

Вспомним, что уже вавилоняне употребляли специальный символ для обозначения нулевого значения разряда. Примерно во II веке до н.э. с астрономическими наблюдениями вавилонян познакомились греческие ученые. Вместе с их вычислительными таблицами они переняли и вавилонскую систему счисления, но числа от 1 до 59 они записывали не с помощью клиньев, а в своей алфавитной нумерации. Но самым замечательным было то, что для обозначения нулевого значения разряда греческие астрономы стали использовать символ **О** (по первой букве греческого слова Ουδεν — ***ничто***). Этот знак, по-видимому, и был прообразом современного нуля.

Индийцы познакомились с греческой астрономией между II и VI вв. н.э., переняв общетеоретические положения этой науки и многие греческие термины. В это время в Индии уже использовалась мультипликативная система счисления. По утверждению историков, примерно в это же время там познакомились и с вавилонской системой счисления, и с греческим круглым нулем. Соединив свою десятичную мультипликативную систему с принципами нумерации чисел греческих астрономов, индийские ученые сделали завершающий шаг в создании всем известной десятичной системы счисления.

Приведем в заключение слова знаменитого математика и физика XVIII—XIX вв. П.Лапласа: "Мысль выражать все числа десятью знаками, придавая им, кроме значения по форме, еще значение по месту, настолько проста, что именно из-за этой простоты трудно понять, насколько она удивительна. Как нелегко было прийти к этому методу, мы видим на примере величайших гениев греческой учености Архимеда и Аполлония, от которых эта мысль осталась скрытой".

2.8. Цифры различных систем счисления

Мы рассмотрели довольно большое количество систем счисления, и в каждой из них использовались свои символы для записи чисел, которые, напомним, называются цифрами.

В палочной системе счисления использовался единственный символ — "палочка", то есть фактически единственная цифра — 1.

В древнеегипетской непозиционной десятичной системе счисления используются следующие цифры:

Единицы	𓏺
Десятки	𓎆
Сотни	𓍢
Тысячи	𓆼

В вавилонской шестидесятеричной системе счисления, частично основанной на позиционном принципе, в качестве цифр выступают два вида клиньев — 𒐕 и 𒐖.

В римской системе счисления в качестве цифр используются следующие заглавные латинские буквы:

I	V	X	L	C	D	M
1	5	10	50	100	500	1000

В алфавитной славянской системе счисления цифры — это 27 букв кириллицы. До конца XVII века (а конкретнее — до реформы Петра I) на Руси цифрами служили следующие буквы:

1 — **А** аз	10 — **І** и*	100 — **Р** рцы
2 — **В** веди	20 — **К** како	200 — **С** слово
3 — **Г** глаголь	30 — **Л** люди	300 — **Т** твердо
4 — **Д** добро	40 — **М** мыслете	400 — **У** ук**
5 — **Е** есть**	50 — **Н** наш**	500 — **Ф** ферт
6 — **S** зело*	60 — **Ѯ** кси**	600 — **Х** хер
7 — **З** земля**	70 — **О** он	700 — **Ѱ** пси*
8 — **И** иже**	80 — **П** покой	800 — **Ѡ** омега*
9 — **Ѳ** фита*	90 — **Ч** червь	900 — **Ц** цы

* Буквы, исключенные впоследствии из русского алфавита.

** Буквы, у которых изменилось начертание.

В современной десятичной системе счисления, которая является позиционной, используется 10 арабских цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Почему мы называем наши цифры арабскими? Дело в том, что с возникшей в Индии десятичной системой счисления первыми познакомились арабы. Они по достоинству ее оценили и начали использовать при подсчетах в торговых операциях. Именно арабы завезли эту систему счисления в Европу. И с начала XII века десятичная система получила распространение во всей Европе под названием арабской.

Будучи проще и удобнее остальных систем, она достаточно быстро вытеснила все другие способы записи чисел. Вот с тех пор цифры, используемые для записи чисел в десятичной системе счисления, и называют арабскими.

Приведем таблицу, отражающую постепенное видоизменение цифр, употреблявшихся арабами. Эти цифры называются цифрами "губар". Откуда произошли сами цифры "губар", до сих пор остается неясным.

Продолжение дискуссии

Нас, сотрудников Мурманского городского методического центра информационных технологий, очень заинтересовала дискуссия на страницах газеты “Информатика”. Поскольку наш центр оценил появившиеся учебники под редакцией профессора Н.В. Макаровой как очень интересные и полезные, мы сочли своим долгом выступить в защиту авторского коллектива. Мы давно ждем появления подобных работ и рекомендовали учителям города Мурманска приобрести в школах комплекты учебников “Информатика” издательства “Питер”. Конечно, это не единственное рекомендованное издание, но многие учителя нашего города уже пользуются этими книгами. В обсуждении приняли участие и пришли к общему мнению Пустоваченко Н.Н., директор ГМЦИТ, Попова Н.Н., зам. директора, методисты Мясникова О.К., Горденков С.В., а также Брылев С.Н., методист кафедры информатики политехнического лицея, Фролова В.И., учитель политехнического лицея, Зайцева В.П., учитель школы № 28, Короповская В.П., учитель школы № 43.

В статье “Информатика” и “Земля Информатика” А.Г. Гейна много критических замечаний, иногда верных, чаще подлежащих обсуждению. Но удивляет чрезвычайно раздраженный тон статьи, в котором проявляются скорее подозрения не то на плагиат из книги “Земля Информатика”, не то на недостаточное ее использование. Неудивительно, если базовые понятия школьного предмета объясняются несколько похожим образом. Удивил бы скорее обратный подход. Странным кажется раздражение автора статьи по поводу качественного оформления учебника (я имею в виду ссылки на определения, выделенные цветным шрифтом). И, по-видимому, автор задался целью представить в своей рецензии только отрицательные стороны учебника “Информатика. 6—7”, а о том, что учебник соответствует возрастному восприятию и учебных пособий для 6-го класса существует не много, не сказано ни слова. Мне как учителю работать интереснее именно по учебнику Н.В. Макаровой. Но после прочтения этой статьи читатель даже в руки не возьмет “хорошо задуманный” (определение А.Г. Гейна) учебник, “лицо” которого ему (читателю) представили и в прямом (фото обложки), и в переносном смысле. Это что, законы конкуренции? Кстати, автор постоянно делает сравнения почему-то со своей книгой “Земля Информатика”, хотя Министерство образования рекомендует к использованию вышедший в 1998 году учебник “Информатика. 7—9” (авторы *А.Г. Гейн, А.И. Сенокосов.* “Просвещение”, 1998). У нас в центре, к сожалению, нет этой книги, но разве там нет определений алгоритма или информатики?

Хочется ответить на спорные замечания автора. Называть ли информатику наукой или областью деятельности, вопрос дискуссионный. В журнале “Информатика и образование” (№ 6/98, стр. 4) С.А. Марков приводит определение науки. “*Наука — это сфера человеческой деятельности, функцией которой является выработка и теоретическая систематизация объективных знаний о действительности*”. Может, и спорить не о чем? Нам при обучении школьников очень редко и практически только в углубленных курсах доводилось упоминать о законах, открытых в науке информатике (кибернетике, теории информации, теории открытых систем). Вспомните науку физику. Сравнение не в пользу школьной информатики.

Теперь об измерениях. Автор сделал точное замечание по “измерению информации”. Но и сам автор допускает иногда «стилистическую “грязь”». Положим рядом книгу “Земля Информатика”. Заголовок п. 19 (№ 24 газеты “Информатика” за 1996 г.) “Знания и информация. Модельные аспекты измерения информации”. Это как? Нельзя же измерить информацию! Как и воду! Видимо, автору тоже не повезло с редактором. И еще по поводу различных измерений — цвет волос измерить можно при помощи кодирования цветовых таблиц (спросите у косметологов или художников).

Автор статьи иногда противоречит сам себе — сначала утверждает, что нельзя давать шестиклассникам понятия без их определения, после чего удивляется присутствию определения понятия “объект”. А в учебнике весь материал строится на этом понятии — идет ли речь о среде Windows, о программировании действий исполнителя Черепашка (язык Лого) или об информационных моделях.

А понятия “параметр” и “признак” действительно воспринимаются почти как синонимы, но в концепции коллектива авторов учебника для 6—7-х классов и в дальнейших изданиях для 8—11-х классов понятие “параметр” более приемлемо. Это связано со спецификой языка Лого. Мне это понятно и вопросов не вызывает. Хотя определение можно изменить, тавтология в нем присутствует. Сравните: “Информация — это сведения (данные)...”

О некоторых понятиях, предлагаемых прежде их объяснения. Существует стиль обучения, при котором учитель не вещает о непререкаемых законах, а размышляет вместе с учеником и помогает ученику сделать правильный вывод. Характерным признаком такого стиля является наличие большого количества вопросов, которые учитель задает ученикам перед тем, как новый материал представлен, объяснен. И в этом смысле мне больше нравится учебник, не совсем похожий на кроссворд, но позволяющий пытливой мысли за что-то зацепиться. Уверяю вас, моим ученикам такая книга интереснее и полезнее. Думаю, что автор статьи, который сам немало потрудился над созданием учебников по информатике и, безусловно, является специалистом в вопросах методики преподавания, знает о существовании разных стилей и методах изучения наук. И таких “опережающих” понятий в учебнике не так много, как, например, в учебном пособии “Общая информатика” С.Симоновича для 5—9-х классов.

Я считаю, что понятие “владеть информацией” в 6-м классе вполне можно преподносить, подразумевая качественное владение общечеловеческими знаниями, и направлять ученика не только на умение запомнить механически набор знаний, но и осмыслить, использовать их. Кстати, российская школа именно этим и отличается от западных вариантов обучения — знаний много, а что с ними дальше делать, никто не знает. А информационное право, вероятно, можно преподавать в дееспособном возрасте, в 8—9-х классах, то есть с 14 лет.

Критика, конечно, полезна, но хотелось бы видеть конструктивный подход, ведущий к общей цели — созданию грамотных, современных, интересных и красивых учебников для наших ребят. Просто неприятно превращение разбора недостатков нового и необходимого издания в попытку разгрома и “стирания в порошок”. Мне близок подход авторского коллектива обсуждаемого учебника еще и потому, что в нем реализуется важная методическая задача — информационные технологии рассматриваются не как кнопочные, по типу справочного издания, а с точки зрения возможностей решения задач и методов их решения. Это особенно хорошо видно, если заглянуть и в следующие учебники Н.В. Макаровой до 11-го класса. И называть это издание справочником по Windows просто неверно. По-моему, если продолжить “увлекательное” занятие — спорить о неустоявшейся терминологии и определениях в информатике, то алгоритмизация и программирование — это тоже информационная технология решения задач, ничем не лучше или хуже других технологий. Сравнительная мощь этого средства стремительно уменьшается со временем. Использование языка программирования Лого гармонично встраивается в полный курс профессора Н.В. Макаровой. Такое программирование действительно может показать экономными средствами, как создается программное обеспечение; реализует объектный подход; имеет визуальную направленность; не ограничено примитивным применением.

Известно, что во многих школах уроки ведутся не только по базовому курсу, но существует множество различных расширенных и углубленных вариантов программ. Рассмотрим наиболее известные издания для среднего звена школы и попробуем сравнить с обсуждаемым учебником (см. таблицу).

“Алгоритмика” — очень интересный учебник, но при разработанности методической поддержки он отличается от учебника Н.В. Макаровой отсутствием продолжения курса. Зная, какую серьезную работу проводят авторы “Алгоритмики” и их коллеги в Перми, можно ожидать, что будут и другие их работы. На практике учителя нашего региона широко используют курс Ю.А. Первина “Информационная культура”, но именно курсы пятого и шестого классов этой программы не всегда принимаются школьниками. Очень привлекательно пособие для учителя “Общая информатика”, но построить курс пятого-шестого классов с его помощью довольно сложно. Работа С. Симоновича недостаточно ориентирована на школьников этого возраста. Самым интересным изданием в сравнении с обсуждаемым учебником является пробный учебник А.Г. Щеголева, так как по целям имеет много общего с “Информатикой. 6—7”. Но качество издания учебника Н.В. Макаровой здесь вне конкуренции. Кроме того, работая с учебником “Информатика. 6—7”, легко почувствовать участие практикующих педагогов в создании этой книги. Это хороший помощник учителю.

Конечно, я не берусь за анализ всех существующих изданий этого направления. Но среди той методической литературы для школьника и учителя, которая популярна в нашем регионе, работа авторского коллектива под руководством Н.В. Макаровой уже занимает одно из первых мест.

О.В. МАРТЫНЕНКО,
методист ГМЦИТ,
Мурманск,
MCIT@dionis.mels.ru

№	Название, автор	Классы	Год издания	Программное обеспечение
1	“Общая информатика”, С. Симонович, уч. пособие	5—9	М., 1998, АСТпресс	Windows 95—98, MS Office
2	“Информатика и моделирование процессов”, А.Г. Щеголев и задачник	6—9	М., 1992	Пролог (детский)
3	“Алгоритмика”, А.К. Звонкин, С.К. Ландо, учебник	5—7	М., Дрофа, 1996	Пакет “Алгоритмика”
4	“Информационная культура”, Ю.А. Первин	1—11	М., Дрофа, 1995	Пакет “КуМир”, “Роботландия”
5	“Информатика”, Н.В. Макарова	6—11	СПб., Питер, 1998	Windows, MS Office, Лого

Задача 11-й Кировской областной олимпиады

Мотоциклисты

С.М. ОКУЛОВ

Среди мотоциклистов г. Кирова популярно следующее соревнование: на одной из улиц города выбирается прямой участок, на котором установлено N ($1 \le N \le 100$) светофоров, имеющих только красный и зеленый цвета. Мотоциклист проезжает выбранный участок на постоянной скорости, причем нарушать правила дорожного движения, т.е. проезжать перекресток на красный свет, категорически запрещается, кроме случаев, когда мотоциклист пересекает перекресток в момент переключения светофоров. Мотоциклист, изменивший скорость движения, либо проехавший на красный свет, либо выбравший скорость, меньшую 5 м/с, дисквалифицируется. Побеждает мотоциклист, проехавший трассу за минимальное время. Известный мотоциклист Федя обратился к представителям науки, чтобы они помогли ему выбрать оптимальную скорость движения по трассе.

Входные данные. В первой строке файла INPUT.TXT записано N — число светофоров. В следующей строке $N+1$ число: первое — расстояние от точки старта до первого светофора, второе — расстояние между первым и вторым светофорами, ..., последнее число — расстояние от последнего светофора до точки финиша (числа из интервала от 1 до 100 000). В третьей строке записаны N чисел — время работы каждого светофора в красном и зеленом режимах (оно одинаково). Время — число из интервала от 1 до 300.

Выходные данные

В файле OUTPUT.TXT должно быть записано одно число (с точностью до четырех знаков) — искомая скорость или сообщение "NO", если решения нет.

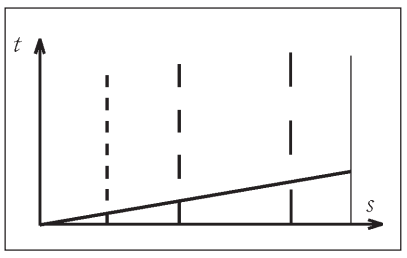
Пример

Входные данные

```
3
200 100 200 200
2 1 9
```

Выходные данные

```
55.5556
```



Идея решения. На рисунке по горизонтальной оси откладываем расстояния, а по вертикальной — время. Время работы каждого светофора в красном режиме выделено на соответствующих расстояниях "черным отрезком". Каждой скорости движения мотоциклиста соответствует наклонная прямая. Для решения задачи необходимо провести прямую из начала координат с наименьшим угловым коэффициентом, проходящую через "светлые" участки рисунка. Обратим внимание на то, что "лучшая" прямая касается хотя бы одного отрезка в верхней точке. Действительно, если она не касается ни одного отрезка в верхней точке, то ее можно сдвинуть ("повернуть") вниз, уменьшая при этом угол, т.е. увеличивая скорость, до соприкосновения с первым отрезком. Из вышесказанного следует решение: необходимо проверить все возможные "верхние" точки "темных отрезков" на предмет их принадлежности искомым прямым и выбрать лучший.

Определим данные.

```
Type  Real=Extended;
Const InputFile='INPUT.TXT';
      OutputFile='OUTPUT.TXT';
      Eps=1e-5;
      MaxN=100;
Var   N:Integer;
      Rs, WhRd: Array[0..MaxN+1] Of Real;
      Can: Boolean;
      ResMax: Real;
```

Процедуры ввода и вывода данных и основная программа представлены ниже.

```
Procedure Init;
Var i: Integer;
Begin
  ResMax:=-MaxLongInt;
  Assign(Input, InputFile);Reset(Input);
  Read(N);
  For i:=1 To N+1 Do Begin
  Read(Rs[i]);
   If i>1 Then Rs[i]:=Rs[i] + Rs[i-1];
     {Считаем расстояния от начальной точки}
  End;
  For i:=1 To N Do Read(WhRd[i]);
     {Время работы каждого светофора}
 Close(Input);
End;
Procedure Done;
 Begin
 Assign(Output, OutputFile);
 Rewrite(Output);
 If Can Then WriteLn(ResMax:0:4)
  Else WriteLn('NO');
 Close(Output);
 End;
{Основная программа}
Begin
 Init; Solve; Done;
End.
```

Если значение переменной `Can` равно `true`, то решение найдено. Вспомогательные процедуры для работы с вещественным типом данных.

```
Function More(a, b:Real):Boolean;
Begin
  More:=(a-b)>Eps;
End;
Function Eq(a, b:Real):Boolean;
  Begin
    Eq:=Abs(a-b)<Eps;
  End;
```

Пусть дана скорость движения мотоциклиста `R`. Необходимо определить, можно ли с этой скоростью проехать все светофоры. Эта подзадача решается с помощью двух функций — `IfMay` и `IfPoss`.

```
Function IfPoss(Const k: Integer;Const Tm: Real): Boolean;
  Var d: Real;
   Begin
    d:=(Tm/WhRd[k]);
    IfPoss:=Eq(d, Round(d)) And Eq(Frac(Int(d)/2), 0.5);
     {значение d кратно времени работы светофора и нечетно}
   End;
Function IfMay(Const R: Real): Boolean;
  Var i: Integer;
   Begin
    IfMay:=False;
     For i:=1 To N Do Begin {i — номер светофора}
      If Not IfPoss(i, Rs[i]/R) Then Exit;
       {R[i]/R - время, в которое мотоциклист доедет со
        скоростью R до светофора с номером i}
       {если для каждого значения i IfPoss возвращает
         значение true, т.е. мотоциклист может проехать
         через данный светофор, то скорость R допустима и
         IfMay присваивается значение true}
     End;
    IfMay:=True;
   End;
```

Последняя, ключевая процедура — `Solve`, в которой производится перебор всех верхних точек отрезков.

```
Procedure Solve;
Var   i: Integer;
  Begin
    Can:=False;
    For i:=1 To N Do TryFind(i);
   {А почему не сделать еще одну процедуру, которая будет
    осуществлять проверку прямых, проходящих через
    "верхние" точки работы светофора с номером i?}
  End;
Procedure TryFind(k: Integer);
Var   i:Integer;
      R: Real;
      Begin
        i:=1;
       R:=Rs[k]/(WhRd[k]*i);
       While (Not More(5, R)) And More(R, ResMax) Do Begin
           If IfMay(R) Then Begin
           {если мотоциклист может проехать на этой скорости все
           светофоры, то одно из решений найдено, следующие
           "верхние" точки этого светофора проверять нет смысла —
           для них скорость мотоциклиста будет меньше}
           ResMax:=R;Can:=True;Exit;
                              End;
        Inc(i,2);{переходим к следующему переключению светофора
                  с красного режима на зеленый}
       R:=Rs[k]/(WhRd[k]*i);{вычисляем скорость}
                                                    End;
End;
```

ОЛИМПИАДЫ

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

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

Тираж 7000 экз.
Заказ №

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

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

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