№ 20 (213) май 1999

еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИКА

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

Пушкин и информатика

А.Г. ГЕЙН

Странное на первый взгляд сочетание вынесено в заголовок этой заметки. Никакой информатики, разумеется, Пушкин не знал, да и самой информатики в его времена еще не было. И все же перечитаем следующие с детства знакомые строки:

Сами шлют гонца другого
Вот с чем от слова до слова:
“Родила царица в ночь
Не то сына, не то дочь;
Не мышонка, не лягушку,
А неведому зверюшку”.
Как услышал царь-отец,
Что донес ему гонец,
В гневе начал он чудесить
И гонца хотел повесить...

Разве это не тема для обсуждения таких вопросов, прямо относящихся к информатике, как

— несет ли канал связи ответственность за содержание информации?

— всякая ли информация достоверна?

— какие есть средства, чтобы защитить информацию от преднамеренного (или непреднамеренного) искажения?

— верным ли было решение царя Салтана повесить гонца?

В последнем случае речь идет не об осуждении Салтана как антигуманиста, а о решении сменить плохо работающее средство связи. Ведь нередко мы так и поступаем с приборами, которые испортились. Я думаю, что учитель информатики найдет в этом фрагменте еще немало того, о чем можно побеседовать с учениками.

А мы пока почитаем дальше:

Но, смягчившись на сей раз,
Дал гонцу такой приказ:
“Ждать царева возвращенья
Для законного решенья”.

Распоряжение царя с точки зрения информатики вполне грамотное — принимать управленческое решение только после того, как во всем сам разберешься. И насколько низкую информационную культуру демонстрирует царь, доверяя свой приказ каналу связи, в правильной работе которого он только что усомнился. Впрочем, с информационной культурой у царя Салтана, по-видимому, совсем плохо. Что он предпринял, вернувшись с войны и видя, что его бояре сделали совсем не то, что он приказывал? Небось повесил-таки гонца, вместо того чтобы выяснить, в чем причина искажения информации.

А что же бояре? О, это совсем другая информатическая история! С одной стороны, получив приказ

И царицу, и приплод
Тайно бросить в бездну вод,

Окончание на с. 16

ДОРОГИЕ КОЛЛЕГИ!

В № 11/99 была опубликована анкета, которую мы обещали сделать регулярной. В ней предлагалось оценить первые десять номеров 1999 года, используя привычную пятибалльную систему, по трем критериям:

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

Например: 1. 4 2. 5 3. 4.

Мы благодарим всех приславших письма и просим вас оценить по тем же критериям номера с 11-го по 20-й. Результаты анализа анкет и розыгрыша призов среди приславших их в редакцию до 30.06.99 г. мы опубликуем в № 33/99.

Спасибо!

Редакция “Информатики”

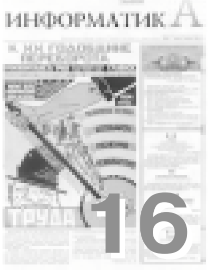
1. __ 2. __ 3. __

1. __ 2. __ 3. __

1. __ 2. __ 3. __

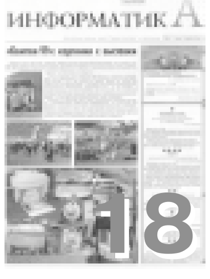
1. __ 2. __ 3. __

1. __ 2. __ 3. __



1. __ 2. __ 3. __

1. __ 2. __ 3. __



1. __ 2. __ 3. __

1. __ 2. __ 3. __

1. __ 2. __ 3. __

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

НАШИ ДЕТИ БУДУТ ЖИТЬ В XXI ВЕКЕ

Системы счисления и компьютерная арифметика

Е.В. АНДРЕЕВА, И.Н. ФАЛИНА

Окончание. Начало в № 14, 15, 16, 17, 18/99.

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

Ранее были опубликованы часть I — “Представление информации (базовый курс)” — и следующие разделы части II (“Системы счисления”):

Глава 1. “Позиционные системы счисления”.
Глава 2. “Арифметические операции в позиционных системах счисления”.
Глава 3. “Перевод чисел из одной позиционной системы счисления в другую”.
Глава 4. “Смешанные и нетрадиционные системы счисления”.

В этом выпуске помещены:
Глава 5. “Задачи для программирования”, а также
Часть III. “Компьютерная арифметика”:
Глава 6. “Целочисленная компьютерная арифметика”.
Глава 7. “Вещественные числа и компьютер”.
Глава 8. “«Длинная» арифметика”.

Кривая дракона

Д.М. ЗЛАТОПОЛЬСКИЙ

В ряде публикаций газеты "Информатика", посвященных компьютерной графике [1—3], были представлены программы, рисующие на экране красивые фигуры, орнаменты и даже изображения растений. Как правило, в них использовалась рекурсия [4—5]. Однако красивые линии с повторяющимся рисунком можно получить и с помощью простых итеративных алгоритмов. Пример такой линии — кривая, изображенная на *рис.* 1. Прежде чем описывать методику ее построения, рассмотрим кривые, которые можно получить следующим образом. Возьмем длинную полоску бумаги и сложим ее пополам, а затем развернем на 90^0. Если смотреть на полоску сбоку, то получится ломаная линия из двух перпендикулярных участков: см. *рис.* 2*а*. Теперь сложим полоску пополам дважды и также дважды развернем на 90^0 так, как это показано на *рис.* 2*б*. Получим ломаную линию уже из четырех отрезков, причем угол между смежными отрезками составляет 90^0. Наконец, если сложение и разворачивание полоски осуществить три раза, то в результате получится фигура, представленная на *рис.* 2*в*. Продолжая этот процесс, можно получить кривую, аналогичную той, которая представлена на *рис.* 1 (прямые углы у кривой на этом рисунке скруглены). Эту причудливую кривую называют кривой дракона. Способ построения подсказывает, что она не имеет самопересечений.

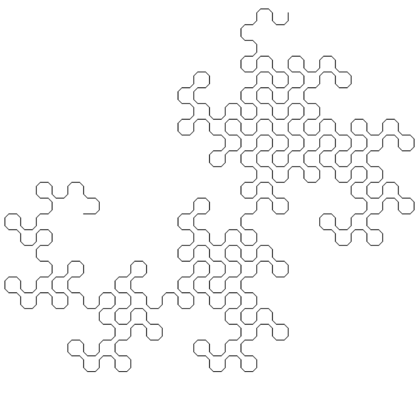
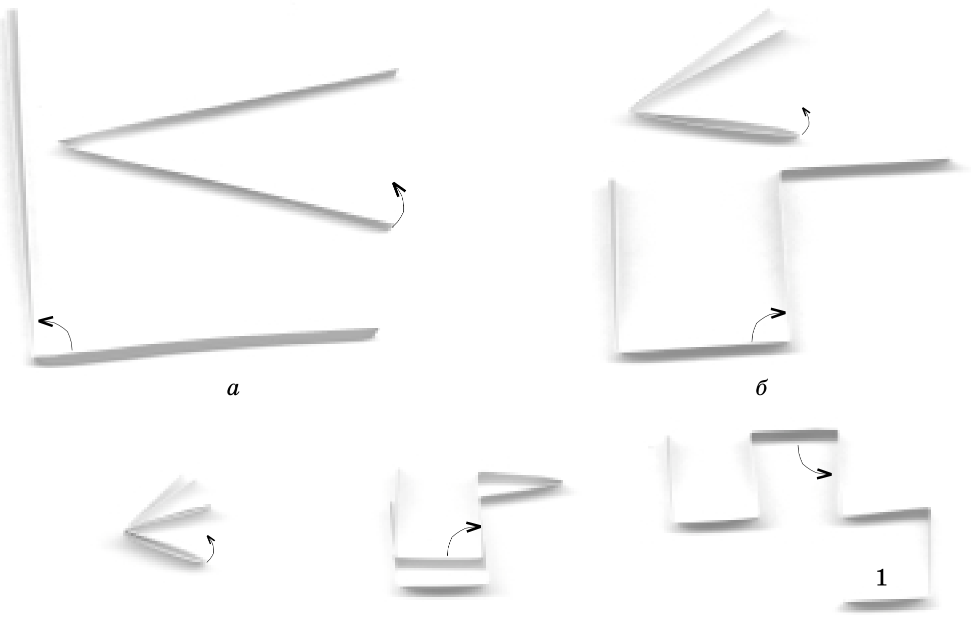


***Рис.* 1**



а *б* *в*

***Рис.* 2**

Приступим к разработке программы построения кривой дракона. Начнем с того, что будем наблюдать ход ломаных линий на *рис.* 2*в*, начиная с отрезка 1, и на каждом углу следить, поворачивается отрезок на 90^0 вправо (по часовой стрелке) или влево (против часовой стрелки). Присвоим код 1 повороту влево и код 3 повороту вправо и обозначим код для отрезка с номером `i` через `K(i)`. Из *рис.* 2*в* видно, что

```
K(1)=1; K(2)=1; K(3)=3; K(4)=1; K(5)=1; K(6)=3; K(7)=3.
```

Задача заключается в том, чтобы найти правило выбора направления поворота (влево или вправо) в конце отрезка с номером i (i =1, 2, ...).

Если рассмотреть ломаные, аналогичные представленной на *рис.* 2*в*, но состоящие из 16, 32 и т.д. отрезков, и проанализировать зависимость направления поворота от номера отрезка, то можно установить, что значение `K(i)` определяется следующим образом:

```
K(i)=K(i div 2),      если i четное;
K(i)=i mod 4,         если i нечетное,
```

где `div` — целочисленное деление, а `mod` — операция определения остатка.

Определение значений `K(i)` может быть проведено с помощью рекурсивной функции, на школьном алгоритмическом языке [4, 6] имеющей вид:

```
алг цел K(арг цел i)
нач
 | если mod(i, 2)=0
 |  | то
 |  |    знач:=K(div(i, 2))  | рекурсивный вызов функции
 |  | иначе
 |  |    знач:=mod(i, 4)
 | все
 |    | знач - значение функции K
кон
```

Построение очередного отрезка ломаной удобно выполнять с использованием команды, которая на школьном алгоритмическом языке называется ***вектор***. Эта команда проводит линию из текущей позиции в точку, заданную приращением ее координат. В Паскале и Си аналогичную роль выполняет команда `linerel`, а в Бейсике модификация команды `LINE: LINE -STEP (dx,dy)`, где `dx,dy` — приращения координат.

В процедуре `Step`, выполняющей построение, рассматриваются 4 варианта направления перемещения (2 направления по горизонтали и 2 по вертикали). Направление задается величиной `angle`. При перемещении вертикально вверх значение этой величины равно 0, горизонтально влево —90, вертикально вниз —180 и горизонтально вправо —270 (т.е. отсчет производится против часовой стрелки от направления вертикально вверх). Длина отрезка ломаной обозначена `len`.

```
алг Step(арг цел angle)
нач
 | выбор
 |  | при angle=0 : вектор(0, -len)
 |  | при angle=180 : вектор(0, len)
 |  | при angle=270 : вектор(len, 0)
 |  | при angle=90 : вектор(-len, 0)
 | все
кон
```

Определим закономерности изменения значения угла `angle` в ходе построения ломаной. Можно утверждать, что в конце каждого отрезка с номером `i` мы должны повернуть на `K(i)*90`^{0} влево, поскольку поворот на 90^0 по часовой стрелке эквивалентен повороту на 270^0 влево против часовой стрелки. Тогда с учетом принятой системы отсчета угла `angle` его очередное (после поворота) значение связано с предыдущим значением зависимостью:

```
angle:=mod ((angle+K(i)*90), 360),                    (1)
```

где `mod` — операция определения остатка, а `K(i)` — код направления поворота (см. выше).

С использованием функции `K` и процедуры `Step`, а также формулы (1) основная часть программы может быть оформлена в виде:

```
цел len              |величину len описываем как глобальную
алг Ломаная
нач цел n, angle, i
    видео(17)       |устанавливаем графический режим работы монитора
    поз(190, 276)   |начальная точка ломаной
    angle:=270      |угол, под которым рисуется первый отрезок
    len:=20         |длина отрезков ломаной
    вектор(len, 0)  |рисуем первый отрезок
    n:=418          |количество отрезков
    нц для i от 1 до n-1
       angle:=mod ((angle+K(i)*90), 360)
       нц 5000 раз  |пауза
       кц
       Step(angle)    |рисуем остальные отрезки линии
    кц
кон
```

Примечания.

1. Принятые значения координат начальной точки ломаной, а также длины отрезков обеспечивают размещение на экране всей линии из 418 отрезков.
2. "Пустой" цикл в программе создает небольшую паузу между вычерчиванием отрезков.

Поскольку функция `K(i)` рекурсивная, то полностью избежать рекурсии, о чем говорилось в начале статьи, не удалось. Между тем в данном случае рекурсия связана с многократным повторением одних и тех же операций. Действительно, для нахождения `K(8)` надо определить `K(4), K(2), K(1)`, а все эти величины уже были найдены и использованы ранее. Попробуем поступить так: будем хранить значения `K` в массиве `arrK[1:n]`. Для нечетных `i` значения `K` находятся непосредственно по формуле `K=i mod 4`. Поэтому сначала заполняем нечетные элементы указанного массива, после чего в цикле по четным `i` от 2 до последнего значения, не превышающего `n`, находим `K[i]` по соотношению: `K[i]=K[i div 2]`. Как обычно, повышение быстродействия потребовало дополнительных затрат памяти. Мы не будем приводить программную реализацию такого подхода к определению `K`, предоставив это заинтересованному читателю.

Как уже говорилось, замечательным свойством полученной линии является то, что в ней отсутствуют самопересечения (помните о складываемой полоске бумаги?). Чтобы наглядно продемонстрировать указанную особенность, можно скруглить все углы или, точнее, заменить их небольшими отрезками прямых линий (т.е. при этом во всех вершинах ломаной будем иметь углы 135^0 вместо 90^0). На *рис.* 1 как раз и представлена такая линия.

Для ее построения уточним приведенную программу. Основные изменения должны быть внесены в процедуру `Step`. Включим в нее также рисование "закругления", предшествующего очередному отрезку. Очевидно, что для вычерчивания этого "скругления" необходимо знать предшествующее направление перемещения. Информацию о нем будем хранить в переменной `old_angle`. Естественно, что длина вычерчиваемого отрезка в данном случае уменьшается до `len-2*a`, где `a` — длина катета "закругления". С учетом этого процедура `Step` оформляется следующим образом:

```
алг Step(арг цел angle, арг рез цел old_angle)
нач
 | выбор
 |  | при  angle=0:
 |  | если old_angle=90
 |  |  | то
 |  |  |    вектор(-a, -a)
 |  |  | иначе
 |  |  |    вектор(a, -a)
 |  | все
 |  | вектор(0, -(len-2*a))
 |  | при angle=180:
 |  | если old_angle=90
 |  |  | то
 |  |  |    вектор(-a, a)
 |  |  | иначе
 |  |  |    вектор(a, a)
 |  | все
 |  | вектор(0, len-2*a)
 |  | при angle=270:
 |  | если old_angle=0
 |  |  | то
 |  |  |    вектор(a, -a)
 |  |  | иначе
 |  |  |    вектор(a, a)
 |  | все
 |  | вектор(len-2*a, 0)
 |  | при angle=90:
 |  | если old_angle=0
 |  |  | то
 |  |  |    вектор(-a, -a)
 |  |  | иначе
 |  |  |    вектор(-a, a)
 |  | все
 |  | вектор(-(len-2*a), 0)
 | все
 | old_angle:=angle
кон
```

Основная часть программы меняется незначительно:

```
цел len, a  |о величине  a  - см. процедуру Step
алг Кривая дракона
нач цел n, angle, old_angle, i
 | видео(17)
 | поз(190+a, 276)
 | len:=20
 | a:=div(len, 5)
 | angle:=270
 | вектор(len-a, 0)
 | old_angle:=270
 | n:=418
 | нц для i от 1 до n-1
 |  | angle:=mod ((angle+K(i)*90), 360)
 |  | нц 10000 раз  |пауза
 |  | кц
 |  | Step(angle,old_angle)
 | кц
кон
```

Язык Паскаль

```
Uses graph,crt;
Const
len=20;{Длина отрезков ломаной}
a=len div 5;{Длина катета "закругления"}
path=''; {Файлы *.BGI находятся в текущем каталоге}
var
d,r,n:integer;
i,angle,old_angle:word;

{Функция, определяющая значение кода направления поворота в конце
 отрезка с номером i}
function K(i:word):byte;
begin if Odd(i) then K:=i mod 4 else K:=K(i div 2) end;

{Процедура рисования отрезка ломаной и предшествующего ему
 "закругления"}
Procedure Step(angle:word; var old_angle:word);
begin
{Рассматриваем 4 варианта направления перемещения. Вычерчиваем
 "закругление" и очередной отрезок }
Case angle of
   0: begin
      if old_angle=90 then LINEREL(-a,-a) else LINEREL(a,-a);
      LINEREL(0,-(len-2*a))
      end;
 180: begin
      if old_angle=90  then LINEREL(-a,a) else LINEREL(a,a);
      LINEREL(0,len-2*a)
      end;
  90: begin
      if old_angle=0 then LINEREL(-a,-a) else LINEREL(-a,a);
      LINEREL(-(len-2*a),0)
      end;
 270: begin
      if old_angle=0 then LINEREL(a,-a) else LINEREL(a,a);
      LINEREL(len-2*a,0)
      end;
end;{Case angle}
old_angle:=angle
end;
BEGIN
d:=DETECT;{режим автоопределения типа монитора}
INITGRAPH(d, r, path); {Переход в графический режим}
MOVETO(190+a, 276); {Начальная точка ломаной}
angle:=270; {Угол наклона первого отрезка}
LINEREL(len-a,0); {Рисуем первый отрезок}
old_angle:=270;
n:=418; {Количество отрезков}
for i:=1 to n-1 do
 begin
  angle:=(angle+K(i)*90) mod 360;
  DELAY(2000);{Задержка}
  Step(angle, old_angle) {рисуем остальные отрезки линии}
 end;
 repeat until keypressed;{Останов до нажатия любой клавиши}
END.
```

Язык Бейсик (вариант QuickBasic [7])

В программе на этом языке процедура рисования отрезка ломаной и предшествующего ему “закругления” имеет имя Ste, а величина длины отрезка ломаной — leng.

```
'Описываем используемые процедуру, функцию и величины
DECLARE FUNCTION K% (i AS INTEGER)
DECLARE SUB Ste (angle AS INTEGER, oldangle AS INTEGER)
DIM SHARED leng AS INTEGER  'Длина отрезков ломаной
DIM SHARED a AS INTEGER 'Длина катета "закругления"
DEFINT A-O
SCREEN (11)      'Переход в графический режим для монитора VGA
leng = 20        'Длина отрезков ломаной
a = leng \ 5     'Длина катета "закругления"
angle = 270      'Угол наклона первого отрезка
LINE (190 + a, 276)-STEP(leng - a, 0) 'Рисуем первый отрезок
oldangle = 270
n = 418   'Количество отрезков
FOR i = 1 TO n - 1
 angle = (angle + K%(i) * 90) MOD 360
 FOR j = 1 TO 32000: NEXT j 'Задержка
 CALL Ste(angle, oldangle)  'рисуем остальные отрезки линии
NEXT i
SLEEP 'Останов до нажатия любой клавиши
END

SUB Ste (angle AS INTEGER, oldangle AS INTEGER)
'Процедура рисования отрезка ломаной и предшествующего ему
'"закругления".Рассматриваем 4 варианта направления перемещения
'Вычерчиваем "закругление" и очередной отрезок
SELECT CASE angle
 CASE 0
  IF oldangle = 90 THEN LINE -STEP(-a, -a)
                   ELSE LINE -STEP(a, -a)
  END IF
  LINE -STEP(0, -(leng - 2 * a))
 CASE 180
  IF oldangle = 90 THEN LINE -STEP(-a, a)
                   ELSE LINE -STEP(a, a)
  END IF
  LINE -STEP(0, leng - 2 * a)
 CASE 270
  IF oldangle = 0 THEN LINE -STEP(a, -a)
                  ELSE LINE -STEP(a, a)
  END IF
  LINE -STEP((leng - 2 * a), 0)
  CASE 90
   IF oldangle = 0 THEN LINE -STEP(-a, -a)
                   ELSE LINE -STEP(-a, a)
   END IF
   LINE -STEP(-(leng - 2 * a), 0)
END SELECT
oldangle = angle
END SUB

FUNCTION K% (i AS INTEGER)
'Функция, определяющая значение кода направления
'поворота в конце отрезка с номером i
IF i MOD 2 = 0 THEN K% = K%(i \ 2) ELSE K% = i MOD 4
END FUNCTION
```

Язык Си

```
#include<graphics.h>
#include <conio.h>
#define len 20 /*Длина отрезков ломаной*/
#define PATH "" /*Файлы *.BGI находятся в текущем каталоге*/
int a;/*Длина катета "закругления"*/;
/*Функция, определяющая значение кода направления
 поворота в конце отрезка с номером i           */
int K(int i)
{if (i % 2 == 1) return i % 4; else  return K(i/2);}

/*Функция рисования отрезка ломаной и предшествующего ему "закругления" */
void Step(int angle, int *old_angle)
{/*Рассматриваем 4 варианта направления перемещения.
   Вычерчиваем "закругление" и очередной отрезок*/
 switch (angle)
  {case 0:
   if (*old_angle==90) linerel(-a,-a); else linerel(a,-a);
   linerel(0,-(len-2*a)); break;
   case 180:
   if (*old_angle==90) linerel(-a,a); else linerel(a,a);
   linerel(0,(len-2*a)); break;
   case 90:
   if (*old_angle==0) linerel(-a,-a); else linerel(-a,a);
   linerel(-(len-2*a),0); break;
   case 270:
   if (*old_angle==0) linerel(a,-a); else linerel(a,a);
   linerel(len-2*a,0); break;
  }
 *old_angle=angle;
}
void main()
{int d=DETECT,/*режим автоопределения типа монитора*/
 r,n=418/*Количество отрезков*/,i,
 angle=270,/*Угол наклона первого отрезка*/
 old_angle=270;
 a=len/5;
 initgraph(&d,&r,PATH);/*Переход в графический режим*/
 moveto(190+a, 276); /*Начальная точка ломаной*/
 linerel(len-a,0); /*Рисуем первый отрезок*/
 for (i=1; i<n; i++)
  {angle=(angle+K(i)*90) % 360;
   delay(2000);/*Задержка*/
   Step(angle, &old_angle);/*рисуем остальные отрезки линии*/
  }
 getch();/*Останов до нажатия любой клавиши*/
}
```

ЛИТЕРАТУРА

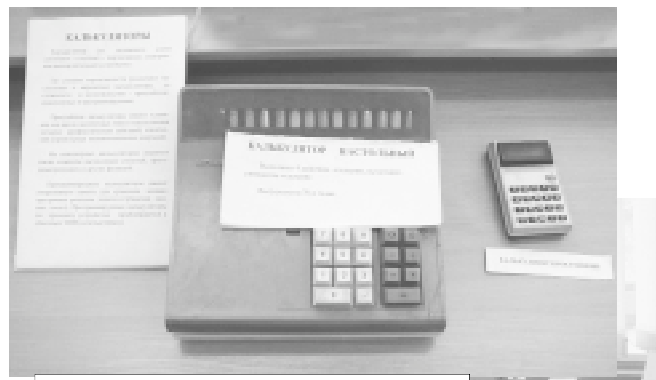
1. *Островский С.Л.* Фрактальные кривые. Информатика, 1995, № 23.
2. Елки, палки и прочие фракталы. Информатика, 1998, № 19.
3. *Островский С.Л., Соколинский Ю.А.* Задачи по теме “Компьютерная графика”. Выпуск 1. Информатика, 1998, № 30.
4. *Кушниренко А.Г., Лебедев Г.В., Сворень Р.А.* Основы информатики и вычислительной техники. М.: Просвещение, 1990.
5. *Златопольский Д.М.* Рекурсия. Информатика, 1996, № 7—8.
6. *Эпиктетов М.Г.* Почему школьный алгоритмический? Информатика, 1995, № 24.
7. *Зельднер Г.А.* QuickBasic 4.5. М.: ABF, 1994.

О кривой дракона читайте также на с. 24

Выставка
История вычислительной техники
в московской гимназии № 1530

В короткий перерыв между майскими праздниками в московской гимназии № 1530 (школа Ломоносова) состоялась выставка “История вычислительной техники”, организованная ребятами под руководством их учителя и нашего постоянного автора Д.М. Златопольского. На выставке было представлено множество интереснейших экспонатов, начиная с японских счетов “соробан” и кончая различными моделями персональных компьютеров отечественного производства, которыми оснащались школы в первые годы компьютеризации: “Агат”, “Искра” и др. Предлагаем вашему вниманию краткий фоторепортаж о выставке.

Большинство экспонатов для выставки ребята принесли из дома, некоторые были подарены друзьями, а счеты “соробан” предоставило китайское посольство в Москве.

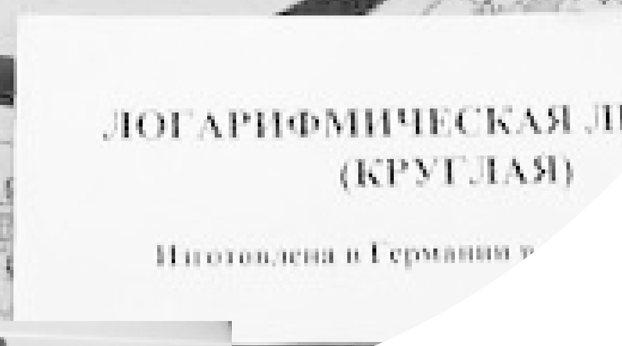


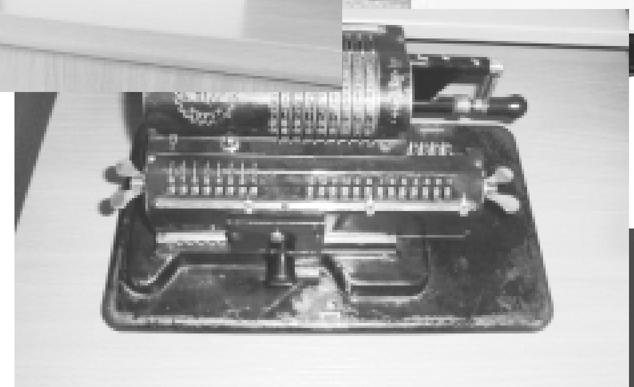
Один из первых калькуляторов — умеет выполнять четыре арифметических действия и не содержит движущихся частей.

В некоторых учебниках информатики до сих пор можно встретить упоминание о таких экзотических носителях информации, как перфоленты и перфокарты. Хорошо, что хоть на таких выставках можно показать их ученикам “живьем”.

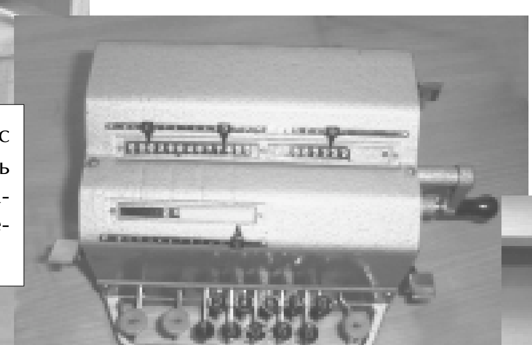
А это один из первых программирующих калькуляторов. Действующая модель! К сожалению, никто не знает, как на нем программировать.

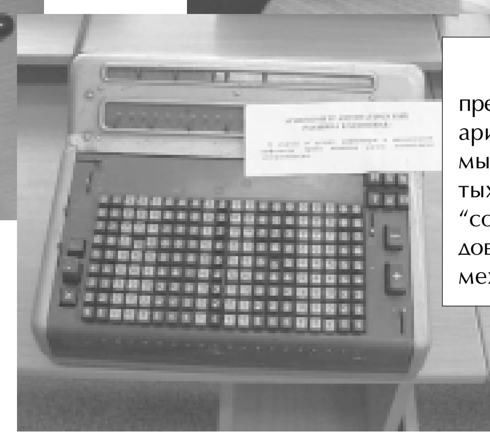
Все привыкли, что логарифмическая линейка похожа на обычную, потому она и линейка. А на выставке можно было увидеть логарифмические круги, самому старому из которых больше века.



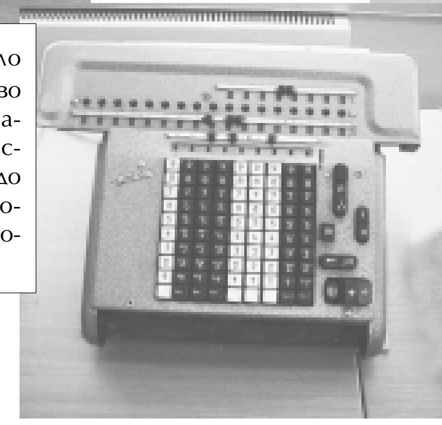


Специализированные расчетные линейки сейчас можно увидеть, пожалуй, только на выставках. А ведь еще не очень давно они верой и правдой служили работникам самых разных специальностей — инженерам-электротехникам, штурманам, гидротехникам.





На выставке было представлено множество арифмометров — от самых “древних” и простых — механических до “современных” (60-х годов выпуска) — электромеханических.



Выставка не является постоянно действующей, но организаторы планируют снова провести ее в начале следующего учебного года. Справки по телефону 268-50-59 (Дмитрий Михайлович Златопольский). Организаторы также будут благодарны всем, кто сможет предоставить для выставки новые экспонаты.

Некоторые из этих компьютеров, к сожалению, до сих пор можно встретить в школах. Но это сегодня “к сожалению”. А более десяти лет назад с этих “БКашек”, “Агатов”, “Искр”, “Электроник” начиналась массовая компьютеризация российских школ. Беда, что ими она и закончилась. Кстати, машины эти, хотя и выработали уже не один свой ресурс, по большей части до сих пор вполне работоспособны.

```
      begin
        write(0);
        k:=k div 10
      end;
    write(s[i]);
  end;
 Close(Output)
End.
```

Решение на языке С

```
void main()
{ALONG s;
 long L,n,i,k;
 FILE *f;
 scanf("%D",&n);
 s[1]=1; /* результат для нулевой степени */
 L=1;/* длина результата */
 while (n>16)/* умножаем на 1<<17=2^17 */
  { Mult(s,&L,(long)1<<17);
    n-=17;
  }
 if (n>0) /*умножаем на оставшуюся степень */
   Mult(s,&L,(long)1<<n);
 f = fopen("output.txt", "wt");
 fprintf(f, "%d", s[L]);
 for (i=L-1;i>=1;i—)
{k=1000;
   while (s[i]<k)
   /* недостающие цифры заменим 0 */
      {fprintf(f,"%c",'0');
        k/=10;
       }
    fprintf(f,"%d",s[i]);
   }
 fclose(f);
}
```

2. Дана последовательность из не более чем 1000 неотрицательных целых чисел, каждое из которых удовлетворяет ограничению $0 < m < 2^{31}$. Числа записаны в позиционной системе счисления с основанием *P*. Необходимо определить количество последних идущих подряд нулей в произведении этих чисел, записанном в той же системе счисления.

Один из возможных вариантов решения этой задачи заключается в прямом вычислении значения произведения этих чисел и определении количества хвостовых нулей в нем. Программа при этом фактически будет состоять лишь из процедуры умножения “длинного” *P*-ичного числа на “короткое”, аналогичной описанной в параграфе 8.3, и может оказаться короче, нежели программа, реализующая алгоритм разложения на *P*-ичные множители.

3. (*Всесоюзная олимпиада школьников по информатике. Харьков, 1990 г., автор — В.В. Прохоров*)

Вычислить с максимальной точностью значение выражения:

$$\sum_{k=1}^{100} \frac{k}{3^{[\log_2 k]}}$$

Результат записать в виде десятичной дроби.

Решение. Для того чтобы решить эту задачу точно, нужно понимать, на каких этапах вычисления значения суммы может возникать погрешность. Так, если производить последовательное суммирование 100 дробей, представленных в одном из вещественных компьютерных типов (пусть даже в 80-разрядном), погрешность суммы может в 100 раз превысить погрешность представления каждой дроби в отдельности. Кроме того, так как нас интересует лишь целая часть от логарифма натурального числа, то использовать операцию отбрасывания дробной части от значения логарифма, полученного в вещественной арифметике, также нельзя. В противном случае мы можем получить, например, для $\log_2 4$ вещественное значение 1,9999999999, целая часть которого равна единице, вместо требуемой двойки. Поэтому вычисление целой части логарифма числа *k* следует проводить в цикле, используя исключительно целочисленную арифметику.

Turbo Pascal

```
log:=0;
i:=1;
while i<k do
  begin
    i:=i*2;
    log:=log+1
  end;
```

C

```
log=0;
i=1;
while (i<k)
  {i*=2;
   log++;
  }
```

То же замечание касается и возведения тройки в целую степень. Заметим, что при $1 \le k \le 100$, $[\log_2 k]$ принимает значения от 1 до 6. Следовательно, сто обыкновенных дробей, которые требуется сложить, имеют всего лишь 6 различных знаменателей: 3, 3^2, 3^3, ..., 3^6, причем наименьшим общим знаменателем этих дробей является 3^6 (это число “короткое”). Поэтому несложно привести все 100 дробей к общему знаменателю и, используя лишь целочисленную арифметику, получить значение числителя и знаменателя результирующей дроби. Для перехода от обыкновенной дроби к десятичной можно воспользоваться алгоритмом деления “короткого” числа на “короткое”, изложенном в параграфе 8.5, и определить точное значение искомой суммы.

8.8. Контрольные вопросы и упражнения

1. Как определить, достаточно ли для точного решения задачи стандартных компьютерных типов данных или вычисления придется организовывать самому, с помощью алгоритмов “длинной” арифметики?

2. Назовите преимущества и недостатки различных способов представления “длинных” чисел.

3. Напишите программу для решения задачи 8.7.2.

4. Решите задачу 8.7.3 для случая, когда основанием логарифма является число 1,9, а в знаменателе вместо степеней тройки стоят степени числа 3,1. Значение суммы по-прежнему должно быть вычислено точно.

5. Реализуйте алгоритм сложения двух знаковых “длинных” чисел. Для этого замените *k*-разрядное отрицательное число *m* на его дополнение до числа 10^k, то есть на число $10^k - |m|$. Сделать это можно по алгоритму, аналогичному алгоритму получения дополнительного кода отрицательного двоичного числа (см. раздел 6.2.1). Обратным кодом десятичного числа является замена каждой его цифры *i* на $(9 - i)$. К обратному коду нужно прибавить единицу, чтобы получить дополнительный код. Если отрицательные слагаемые заменены на их дополнительные коды, то для сложения можно применить процедуру `Add`, приведенную в 8.2.

6. Реализуйте алгоритм целочисленного деления с остатком “длинного” числа на “длинное”.

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

Окончание. Начало см. в № 14, 15, 16, 17, 18/99

Глава 5. Задачи для программирования

Данная глава предназначена в основном для учебных заведений, в курс информатики которых включено программирование на современных алгоритмических языках. Однако часть задач может быть решена и с помощью электронных таблиц, например, Microsoft Excel (даже без использования встроенного языка Visual Basic, что также будет продемонстрировано.

Данная глава очень важна в курсе программирования. Решение предложенных задач закрепит умение школьников составлять алгоритмы с использованием циклов, а также осуществлять операции с целыми, вещественными и символьными типами данных.

При выполнении большинства заданий не нужно использовать массивы или строковый тип данных. Такая рекомендация заставляет учащихся продумать организацию ввода чисел в *P*-ичной системе. Можно предложить использовать схему Горнера для вычисления значений полиномов (развернутой формы записи чисел в позиционных системах счисления).

5.1. Задания

1. Идет *K*-я секунда суток. Определить, сколько целых часов и целых минут прошло с начала суток. Например, если *K*=13 257=3 · 3600+40 · 60+57, то *H*=3 и *M*=40. Вывести на экран фразу: “Это ... часов ... минут” (вместо многоточия программа должна выводить найденные значения *H* и *M*). Возможно некоторое усложнение, требующее правильного окончания слов. Например, “Это 3 часа 21 минута”.

2. Написать программу перевода целых чисел, не превосходящих 10^9, из десятичной системы счисления в *P*-ичную методом деления с остатком на основание системы *P*, где *P*=10 + <номер вашей фамилии в классном журнале>.

При этом цифры *P*-ичной системы счисления, соответствующие десятичным числам 10, 11, ..., *P*—1, заменяются при выводе на экран заглавными латинскими буквами *A*, *B*, ... и т.д.

В начале текста программы надо в комментариях указать номер вашей фамилии в классном журнале и привести тесты, на которых вы отлаживали свою программу. Например, для языка Pascal:

```
{Номер моей фамилии (Богданов) в классном журнале - 4.
      Тесты:  0(10)==>  0(14),
      5(10)==>  5(14),
      11(10)==>  B(14),
      14(10)==>10(14),
      30(10)==>22(14)}
```

3. Решить задачу 5.1.2 методом выделения максимальной степени основания системы счисления.

4. Написать программу перевода целых чисел из *P*-ичной системы счисления в десятичную. Основание *P* ($1<P\le36$) и само число, состоящее не более чем из 6 цифр, вводятся с клавиатуры. Обязательно сделать проверку правильности записи вводимого числа в *P*-ичной системе.

5. Решить задачу 5.1.4 для действительных *P*-ичных чисел с конечной дробной частью, количество цифр в которой не превосходит 6, с точностью до 8 десятичных цифр после запятой.

6. Написать программу перевода действительных чисел, целая часть которых не превосходит 10^9, а дробная отлична от 0, но конечна и состоит не более чем из 8 цифр, из десятичной системы счисления в двоичную. Для перевода дробной части использовать алгоритм умножения на 2 до получения заданной погрешности (количество требуемых цифр в дробной части результата). В начале текста программы в комментариях поместить тесты, на которых вы отлаживали свою программу. Например, для языка Pascal:

```
{0.0(10)  ==>   0.0(2),
 2.1(10)  ==>  10.0001100110011..(2),
 2.5(10)  ==>  11.1(2)}
```

7. Даны взаимно простые натуральные числа *m* и *n* ($m<n$). Найти периодическую и непериодическую части десятичной дроби, равной $\frac{m}{n}$.

8. Бесконечную десятичную периодическую дробь, заданную в виде непериодической части и периода, перевести в *P*-ичную систему счисления ($1<P<10$), выделив непериодическую часть и период у результата.

9. Бесконечную *P*-ичную периодическую дробь ($1<P<10$), заданную в виде непериодической части и периода, перевести в десятичную систему счисления, выделив непериодическую часть и период у результата.

10. Перевести число, записанное римскими цифрами (для их записи используются латинские буквы I — 1, V — 5, X — 10, L — 50, C — 100, D — 500, M — 1000), в его десятичное представление.

11. Перевести десятичное число в его представление римскими цифрами (см. предыдущую задачу).

12. Найти все двузначные и трехзначные десятичные числа, которые в другой системе счисления записаны теми же цифрами, что и в десятичной системе, но в обратном порядке (см. задачу 11 из параграфа 1.5).

13. Написать программу, которая по введенному натуральному десятичному числу (не более чем шестизначному) будет находить все системы счисления, в которых данное число будет представлено теми же цифрами, но записанными в обратном порядке.

14. Для числа, записанного в десятичной системе счисления, проверить, является ли оно палиндромом в системе счисления с основанием от 2 до 9. Число называется палиндромом, если оно одинаково читается как слева направо, так и справа налево.

15. $A_x = B_y$, где A, B — натуральные числа, состоящие не более чем из 6 цифр, которые записаны с помощью цифр от 0 до 9 в системах счисления с основаниями x и y соответственно. Числа A и B программа запрашивает. Количество цифр в числе A может отличаться от количества цифр в числе B. Найти такие основания их систем счисления x и y, в которых данные числа равны.

16. Вывести на экран таблицу умножения в P-ичной системе счисления ($1<P<20$), значение P запрашивается с клавиатуры.

17. *(Кировская областная олимпиада по информатике, 1998 г.).* Часы древних марсиан устроены следующим образом: из большой корзины каждую секунду выкатывается шарик и попадает в первую корзину. Как только в ней накапливается пять шариков, корзина переворачивается, и четыре шарика возвращаются в большую корзину, а пятый шарик попадает во вторую корзину. Как только во второй корзине накапливается шесть шариков, пять из них возвращаются во вторую корзину, а шестой — в третью корзину, и так далее (в третьей корзине не может быть больше 7 шариков, в четвертой — восьми, ...). Часы проработали T секунд. Требуется определить минимальное количество шариков в большой корзине, необходимое для того, чтобы часы успешно проработали еще P секунд ($T+P\leq 1\,000\,000\,000$). Входными данными служат числа T и P.

5.2. Указания и решения

Задача 5.1.1 — без усложнения

Решение на языке Turbo Pascal

```
Var
 k:longint;
  h,m:integer;
Begin
  write('Введите текущее время в секундах: ');
  readln(k);
  h:=k div 3600;
  m:=k mod 3600 div 60;
  writeln('Это ',h, ' часов ',m,' минут.');
  readln
End.
```

Решение на языке Basic

```
5 PRINT
10 INPUT "Введите текущее время в секундах: ", k%
20 h% = k% \ 3600
30 m% = k% MOD 3600 \ 60
40 PRINT "Это "; h%; " часов "; m%; " минут."
50 END
```

Решение на языке C

```
#include <stdio.h>
void main(void)
{long k;
 int h,m;
 printf("Введите текущее время в секундах: ");
 scanf("%ld",&k);
 h=k/3600;
 m=(k%3600)/60;
 printf("Это %d часов %d минут.\n",h,m);
}
```

Задача 5.1.3

Решение на языке Turbo Pascal

```
{Номер моей фамилии (Антонов) в классном
 журнале  - 2.
 Тесты:     10(10) ==> A(12)
            12(10) ==> 10(12)
           645(10) ==> 459(12)
          1001(10) ==> 6B5(12)
         12903(10) ==> 7573(12)}
Var
    c, n, i, P:byte;
   A,x:longint;
Begin
  write('Введите число в десятичной с/с: ');
  readln(x);
  P:=12;
  write('Число в ',P,'-ичной системе счисления: ');
  {Поиск наибольшей степени P: X>=P^N=A}
  A:=1;
  n:=0;
  while x div p>=A do
    begin
      A:=A*P;
      n:=n+1
    end;
  for i:=n downto 0 do
    begin
      c:=x div A;
      x:=x mod A;
      A:=A div P;
      case c of
        0..9:write(c);
        else write(chr(c+ord('A')-10))
      end {of case}
    end;
  readln
End.
```



```
     inc(c[0]);
     c[c[0]]:=p div b;
     p:=p mod b
   end
End;
Procedure WriteResult;
Var
  i:word;
Begin
 Assign(Output,'Output.txt');
 rewrite(Output);
 for i:=1 to c[0] do
 {печатаем целую часть частного от деления}
   write(c[i]);
 writeln;
 writeln(p);{печатаем остаток от деления}
 Close(Output)
End;
{Основная программа}
Begin
  ReadData;
  Count;
  WriteResult
End.
```

Решение на языке C

```
#include <stdio.h>
typedef int TLong[16000];
TLong a,c;
long  b,p;
void ReadData()
{char ch;
 FILE *f;
 f = fopen("INPUT.TXT", "rt");
 a[0]=0;/*длина числа a*/
 c[0]=0;/*длина числа c*/
 fscanf(f,"%c",&ch);
 while (!(ch=='\n'))
 /*пока не достигнут конец строки*/
   {a[0]++;
    a[a[0]]=ch-48;
   /*перевод символа в цифру*/
    fscanf(f,"%c",&ch);
   }
 fscanf(f,"\n%D",&b);
 fclose(f);
}
void Count()
{unsigned int ptr=1;
 p=0;
 while ((p<b)&&(ptr<=a[0]))
 {p=p*10+a[ptr];
  ptr++;
 }
 c[0]++;
 c[c[0]]=p/b;
 p%=b;
 while (ptr<=a[0])
 {p=p*10+a[ptr];
  ptr++;
  c[0]++;
  c[c[0]]=p/b;
  p%=b;
 }
}
void WriteResult()
{unsigned int i;
 FILE *f;
 f=fopen("Output.Txt","wt");
 for (i=1;i<=c[0];i++)
/*печатаем целую часть частного от деления*/
   fprintf(f,"%d",c[i]);
 fprintf(f,"\n%ld",p);
 fclose(f);
}
void main()
{ReadData();
 Count();
 WriteResult();
}
```

8.7. Задачи с решениями

1. По введенному натуральному числу $N < 30\,000$ найти значение выражения 2^N.

Решение. Известно, что существует эффективный по количеству выполняемых действий умножения алгоритм возведения произвольного числа в натуральную степень. Однако при его реализации для больших значений степени N придется выполнять операцию умножения "длинного" числа на "длинное", которая, как было показано в параграфе 8.4, требует порядка L^2 умножений, где L — количество цифр в каждом из "длинных" множителей.

Второй возможный подход к решению данной задачи — это перевод двоичного числа, состоящего из одной единицы и N нулей (а именно так в двоичной системе выглядит 2^N), в десятичную систему счисления методом деления в двоичной системе на $10=1010_2$. Но деление в двоичной системе придется организовывать самостоятельно (аналогично алгоритму деления длинного числа на короткое, разобранному в параграфе 8.6).

Наиболее же простым с точки зрения понимания и отладки программы является алгоритм простого последовательного умножения на некоторую степень двойки, являющуюся "коротким" числом. Приведем такое решение задачи, использующее процедуру `mult` из параграфа 8.3. Умножение здесь производится, пока это возможно, на 2^{17} (поскольку произведение 2^{17} на максимальную цифру 10 000-ичной системы счисления — 9999 еще меньше, чем максимальное число, представимое в целом типе). Затем результат домножается на оставшееся 2^n, где $n < 17$.

Решение на языке Turbo Pascal

```
Var
  s:along;{данный тип определен в 8.1}
  L,n,i,k:integer;
Begin
  readln(N);{вводим значение степени}
  s[1]:=1; {результат для нулевой степени}
  L:=1;{длина результата}
   while n>16 do
   {умножаем на 1 shl 17=2^17}
     begin
       mult(s,L,1 shl 17);
        n:=n-17
     end;
  if n>0 then{умножаем на оставшуюся степень}
    mult(s,L,1 shl n);
  assign(output,'output.txt');}
  rewrite(output);
  write(s[L]);
  {старший разряд просто печатаем}
  for i:=L-1 downto 1 do
    begin
      k:=1000;
      while s[i]<k do
      {недостающие цифры заменим 0}
```

При выводе дробной части результата надписи о наличии или отсутствии непериодической части можно опустить, а период напечатать внутри круглых скобок. Если же период отсутствует (в рамках решения задачи 5.1.7 это означает, что длина периода равна единице и состоит он из цифры ноль), то его лучше не печатать совсем.

Другой подход заключается просто в получении целой части и заранее фиксированного количества цифр дробной части, т.е. в решении задачи с желаемой точностью. Программа при этом оказывается несравненно более простой.

Решение на языке Turbo Pascal

```
Var
  m,n,i,k:longint;
Begin
  write('Введите делимое: ');
  readln(m);
  write('Введите делитель: ');
  readln(n);
  write('Введите количество цифр в дробной части: ');
  readln(k);
  write(m div n,',');
  m:=m mod n;
  for i:=1 to k do
    begin
      write(10*m div n);
      m:=10*m mod n;
      if m=0 then break
    end;
  writeln;
  readln
End.
```

Решение на языке C

```
#include <stdio.h>
#include <conio.h>
void main(void)
{long m,n,i,k;
 printf("Введите числитель дроби: ");
 scanf("%D",&m);
 printf("Введите знаменатель дроби: ");
 scanf("%D",&n);
 printf("Введите количество цифр в дробной части: ");
 scanf("%D",&k);
 printf("%ld,",m/n);
 m%=n;
 for(i=0;i<k;i++)
   {printf("%d",(m*=10)/n);
    m%=n;
    if (m==0) break;
   }
 printf("\n");
 getch();
}
```

Заметим, что более корректным было бы не просто печатать последнюю из k цифр дробной части, а в зависимости от значения $(k+1)$-й цифры округлять результат (т.е. увеличивать его на единицу в случае, когда $(k+1)$-я цифра больше четырех. Но тогда результат надо было бы предварительно запоминать, а потом использовать процедуру прибавления единицы к “длинному” числу. Согласитесь, что гораздо проще задать лишнюю пару цифр в дробной части.

8.6. Деление “длинного” числа на “короткое” нацело с остатком

Решение этой задачи мало отличается от предыдущей, поэтому дадим лишь некоторые пояснения к приведенным ниже программам.

Так как делить на этот раз мы будем “длинное” число, входные данные удобнее представлять в файле. Входным файлом в данной задаче является файл INPUT.TXT, причем в первой строке — делимое, а во второй — делитель, а выходным — OUTPUT.TXT. Делимое при вводе из файла в подпрограмме `ReadData` запоминается в массиве `a` (в первом элементе старшая цифра), делитель — в целой переменной `b`, а целую часть результата будем получать в массиве `c` (в первом элементе также находится старшая цифра).

Нулевые элементы массивов `a` и `c` содержат значения текущей длины этих массивов (`a[0]` — заполняется при считывании делимого из файла, а `c[0]` — при получении целой части результата. Так как делитель у нас является числом “коротким”, а остаток от деления всегда меньше делителя, то для его получения массив не нужен.

Решение на языке Turbo Pascal

```
Type
  AByte=array[0..32000] of byte;
Var
  a,c:AByte;
  b,p:longint;
Procedure ReadData;
Var
  ch:char;
Begin
  Assign(input,'input.txt');
  Reset(input);
  a[0]:=0;{длина числа A}
  c[0]:=0; {длина числа C}
  while not SeekEoln do
  {пока не достигнут конец строки}
  begin
    read(ch);
    inc(a[0]);
    a[a[0]]:=ord(ch)-ord('0')
    {перевод символа в цифру}
  end;
  readln;
  read(b);
  Close(input)
End;
Procedure Count;
Var
  ptr:word;
Begin
  p:=0;
  ptr:=1;
  while (p<b) and (ptr<=a[0]) do
  begin
    p:=p*10+a[ptr];
    inc(ptr)
  end;
  inc(c[0]);{c[0] — длина целой части частного}
 c[c[0]]:=p div b;{определили первую цифру частного}
  p:=p mod b;
  while ptr<=a[0] do
  {определяем остальные цифры частного}
  begin
    p:=p*10+a[ptr];
    inc(ptr);
```


Решение на языке Basic

```
'Номер моей фамилии в классном журнале
'(Антонов) — 2
'Тесты:      10(10) ==>  A(12)
'            12(10) ==> 10(12)
'           645(10) ==> 459(12)
'          1001(10) ==> 6B5(12)
'         12903(10) ==> 7573(12)
10  INPUT "Введите десятичное число: ", X%
20  P% = 12
30  PRINT "Число в "; P%; "-ичной системе
         счисления: ";
40  A% = 1
50  N% = 0
'Поиск наибольшей степени P: X>=P^N
60  IF X% < A% * P% THEN 80
70  A% = A% * P%: N% = N% + 1: GOTO 60
80  FOR i% = N% TO 0 STEP —1
90  C% = X% \ A%
100 X% = X% MOD A%
110 A% = A% \ P%
120 IF C% >= 0 AND C% <= 9 THEN 130 ELSE 140
130 PRINT CHR$(C% + ASC("0")); : GOTO 150
140 PRINT CHR$(C% + ASC("A") — 10);
150 NEXT i%
160 PRINT
170 END
```

Решение на языке C

```
#include <stdio.h>
#include <conio.h>
/* Номер моей фамилии в классном журнале
(Антонов) — 2
   Тесты:          10(10) ==> A(12)
                   12(10) ==> 10(12)
                  645(10) ==> 459(12)
                 1001(10) ==> 6B5(12)
                12903(10) ==> 7573(12) */
void main(void)
{unsigned char c,n,i,p;
 long a,x;
 printf("Введите число в десятичной системе
        счисления: ");
 scanf("%ld",&x);
 p=12;
 printf("Число в %d-ичной системе счисления: ",(int)p);
 a=1;
 n=0;
 while(x/p>=a)
 {a*=p;
  n++;
 }
 for(i=n;i!=255;i--)
 {c=x/a;
  x%=a;
  a/=p;
  if(c>=0&&c<=9) printf("%d",(int)c);
  else printf("%c",c+'A'-10);
 }
 getch();
 printf("\n");
}
```

Задача 5.1.5

Во входной строке целая часть P-ичной дроби от дробной части отделена запятой (при решении на языке Basic — точкой, так как запятая в нем является разделителем между входными данными). В решении сначала производится перевод в десятичную систему целой части, а затем дробной. При переводе в десятичную систему дробной части, для избежания накопления погрешности вычислений, сначала получаем обыкновенную дробь, а затем делим числитель на знаменатель, оставляя требуемые 8 цифр после запятой в результате.

Решение на языке Turbo Pascal

```
Sym:char;
  P, X0,X1,Y:integer;
Function Translate(s:char;P:word):integer;
var Temp:integer;
Begin
  case UpCase(s) of
   '0'..'9':Temp:=ord(s)-ord('0');
    'A'..'Z':Temp:=ord(upcase(s))-ord('A')+10;
    else
      begin
        writeln(s,' — не цифра');Halt
      end
   end; {of case}
   if Temp<P then Translate:=Temp
   else
     begin
       writeln(s,' — неверная цифра');Halt
     end
 End;{of Function Translate}
Begin {основная программа}
 write('Введите основание системы счисления: ');
 readln(P);
 write('Введите число: ');
 X0:=0; {целая часть результата}
 read(Sym);
 while not (Sym=',') do
 {пока не закончится целая часть}
   begin
  {формируем десятичное число по схеме Горнера}
     X0:=X0*P+Translate(Sym,P);
     read(Sym)
   end;
 X1:=0;{числитель дробной части}
 Y:=1;{знаменатель дроби}
 while not eoln do
 {пока не закончится входная строка}
   begin
     read(Sym);
     Y:=Y*P;
     X1:=X1*P+Translate(Sym,P)
   end;
 readln;
 writeln('=>',X0+X1/Y:0:8,'(10)');
  readln
End.
```

Решение на языке Basic

```
10 INPUT "Введите основание системы
         счисления: ", P%
20 INPUT "Введите число: ", sym$
30 X0% = 0 'целая часть в десятичной системе
40 IF MID$(sym$, 1, 1) = "." THEN 80
```

```
'целая часть еще не закончена
50 S$ = sym$: P1% = P%: GOSUB 200
60 X0% = X0% * P% + Translate%
70 sym$ = MID$(sym$, 2, LEN(sym$) — 1): GOTO 40
80 X1% = 0: Y% = 1
90  Q$ = MID$(sym$, 2, LEN(sym$) — 1)
'строка Q содержит дробную часть числа
100 FOR ii% = 1 TO LEN(Q$)
110 sym$ = MID$(Q$, ii%, 1)
120 Y% = Y% * P%: S$ = sym$
130 P1% = P%: GOSUB 200
135 X1% = X1% * P% + Translate%
140 NEXT ii%
150 S$ = STR$(X1% / Y%): D#= X0% + X1% / Y%
180 PRINT "=>", D# , "(10)"
190 END
200 IF ASC(S$) >= ASC("0")
   AND ASC(S$) <= ASC("9") THEN 240
210 IF ASC(S$) >= ASC("a")
   AND ASC(S$) <= ASC("z") THEN 250
220 IF ASC(S$) >= ASC("A")
   AND ASC(S$) <= ASC("Z") THEN 250
230 PRINT MID$(S$, 1, 1), " — это не цифра": END
240 Temp% = ASC(S$) — ASC("0"): GOTO 260
250 Temp% = ASC(S$) — ASC("A") + 10
260 IF Temp% < P1% THEN 280
270 PRINT MID$(S$, 1, 1), " — неверная цифра": END
280 Translate% = Temp%
290 RETURN
```

Решение на языке С

```
#include <stdio.h>
#include <conio.h>
char sym;
long p,x0,x1,y;
int translate(char s,unsigned char p)
{int temp;
 char supper=toupper(s);
 if(s>='0'&&s<='9') temp=s-'0';
 else
  if(supper>='A'&&supper<='Z') temp=supper-'A'+10;
  else
  {printf("\n%c — не цифра\n",s);exit(-1);
  }
 if(temp<p) return temp;
 printf("\n%c — неверная цифра\n",s);exit(-1);
}
void main(void)
{printf("\n, Введите основание системы счисления: ");
 scanf("%d",&p);
 printf("Введите число: ");
 x0=0;
 sym=getche();
 while(sym!=',')
 {x0=x0*p+translate(sym,p);
  sym=getche();
 }
 x1=0;
 y=1;
 for(;;)
 {sym=getche();
  if(sym==13) break;
  y*=p;
  x1=x1*p+translate(sym,p);
 }
 printf("\n=>%0.8f(10)\n",(double)x1/y+x0);
 getch();
 printf("\n");
}
```

Задача 5.1.7

Очевидно, что различных остатков при делении на n не может быть больше, чем n (их значения лежат в интервале от 0 до $n-1$). Если даже все первые n остатков при делении m на n различны, то $(n+1)$-й остаток обязательно совпадет с одним из уже полученных ранее, то есть он находится в периодической части. Поэтому для нахождения длины периода L запомним $(n+1)$-й остаток и будем генерировать остатки дальше, пока не получим совпадающий с ним. Количество сгенерированных при этом чисел и составит длину периода.

Зная длину периода L, найдем непериодическую часть следующим образом: начиная деление, сначала получим L-й остаток. Если он совпадает с m, то непериодическая часть нулевая, в противном случае — сравниваем первый и $(L+1)$-й остаток, второй и $(L+2)$-й и т.д. остатки, пока не найдем совпадающую пару. Количество сравнений до совпадения и является длиной непериодической части. Получаемые при этом в результате деления цифры можно выдавать на печать. Следующие же за непериодической частью L цифр результата деления составят период. Если дробь является конечной, то в качестве периода будет напечатано число 0.

Решение на языке Turbo Pascal

```
Var
  m,n,x,y,i,k:longint;
Begin
  write('Введите числитель дроби: ');
  readln(m);
  write('Введите знаменатель дроби: ');
  readln(n);
  x:=m;
  for i:=1 to n do
    x:=10*x mod n;
  {нашли (n+1)-й остаток при делении m на n
   столбиком, запомним его и будем дальше
   искать равный ему}
  y:=x;
  k:=0;
  repeat
    k:=k+1;
    x:=10*x mod n
  until x=y;
  x:=m;
  for i:=1 to k do
    x:=10*x mod n;
  y:=m;
  write(m,'/',n,'=0,');
  {сравнивая остатки, отстоящие друг от друга
   на длину периода, определим непериодическую
   часть}
while x<>y do
    begin
      write(10*y div n);
      {печать очередной цифры}
      y:=10*y mod n ;
      x:=10*x mod n;
    end;
  writeln('(');{печатаем период}
  for i:=1 to k do
    begin
      write(10*y div n);
```



```
Procedure ReadData;
Var
  c:char;
Begin
  Assign(input,'input.txt');
  Reset(input);
  aLen:=0;
  bLen:=0;
  while not SeekEoln do
  {пока не достигнут конец строки}
  begin
    read(c);
    inc(aLen);
   a[aLen]:=ord(c)-ord('0')
    {перевод символа в цифру}
  end;
  readln;
  while not SeekEoln do
    begin
      read(c);
      inc(bLen);
     b[bLen]:=ord(c)-ord('0')
      {перевод символа в цифру}
    end;
  Close(input)
End;
Procedure LongXLong;
Var
  i,j,p,q:word;
Begin
  p:=aLen+bLen;
  FillChar(c,p+1,0);
  {заполнение массива C нулями}
  for i:=aLen downto 1 do
    for j:=bLen downto 1 do
      begin
      {q — место в результате для
       произведения i-й и j-й цифры каждого
       из множителей}
       q:=p-i-j+1;
      c[q]:=c[q]+a[i]*b[j];
       if c[q]>9 then
       begin {перенос в следующий разряд}
        c[q+1]:=c[q+1]+c[q] div 10;
        c[q]:=c[q] mod 10
       end
      end;
{в c[0] поместим длину результата умножения}
  if c[p]<>0 then c[0]:=p else c[0]:=p-1
End;
Procedure WriteResult;
Var
  i:word;
Begin
  Assign(Output,'Output.txt');
  Rewrite(Output);
  for i:=c[0] downto 1 do
    write(c[i]);
  writeln;
  Close(Output)
End;
{Основная программа}
Begin
  ReadData;
  LongXLong;
  WriteResult
End.
```

Преимущество приведенных в данном параграфе программ заключается в том, что их текст является коротким и “прозрачным”, а следовательно, отладка подобной программы не составляет особого труда. Однако существует и более эффективный рекурсивный алгоритм умножения.

Пусть x и y — два n-разрядных десятичных числа и пусть n — некоторая натуральная степень числа 2. Разобьем цифры x и y на 2 равные группы, обозначив левые (старшие) цифры чисел x_1 и y_1 соответственно, а правые (младшие) — x_2 и y_2. Тогда

$$x{\cdot}y = (x_1{\cdot}10^{n/2} + x_2)(y_1{\cdot}10^{n/2} + y_2) =$$
$$= x_1{\cdot}y_1{\cdot}10^{n} + (x_1{\cdot}y_2 + x_2{\cdot}y_1)10^{n/2} + x_2{\cdot}y_2 =$$
$$= x_1{\cdot}y_1{\cdot}10^{n} + ((x_1 + x_2)(y_1 + y_2) - x_1{\cdot}y_1 - x_2{\cdot}y_2)10^{n/2} + x_2{\cdot}y_2.$$

То есть умножение двух чисел, каждое из которых имеет не более n разрядов, мы свели к трем действиям умножения чисел в два раза меньшей длины, сдвигу результатов такого умножения на соответствующее число разрядов влево и сложению длинных чисел. Однако числа $(x_1 + x_2)$ и $(y_1 + y_2)$, вообще говоря, имеют $n/2 + 1$ разряд, и поэтому их произведение нельзя вычислить непосредственным рекурсивным применением нашего алгоритма. Чтобы сделать алгоритм универсальным, перепишем его уже для произвольного числа n, обозначив буковой m — остаток от деления n на 2:

$$x{\cdot}y = (x_1{\cdot}10^{[n/2]+m} + x_2)(y_1{\cdot}10^{[n/2]+m} + y_2) =$$
$$= x_1{\cdot}y_1{\cdot}10^{n+m} + (x_1{\cdot}y_2 + x_2{\cdot}y_1)10^{[n/2]+m} + x_2{\cdot}y_2 =$$
$$= x_1{\cdot}y_1{\cdot}10^{n+m} + ((x_1 + x_2)(y_1 + y_2) - x_1{\cdot}y_1 -$$
$$- x_2{\cdot}y_2)10^{[n/2]+m} + x_2{\cdot}y_2.$$

Теперь возможна рекурсивная реализация данного алгоритма. Если же при некотором рекурсивном вызове n окажется меньше пяти, то умножение x и y уже можно произвести непосредственно в рамках четырехбайтного целого типа.

С ростом n такой алгоритм является асимптотически более быстрым, нежели умножение “столбиком”, однако его реализация более громоздкая, и в ней приходится использовать несколько дополнительных массивов, что уменьшает максимальную длину чисел, для которых этот алгоритм может быть применен.

8.5. Деление “короткого” числа на “короткое” столбиком

Несмотря на то, что исходные данные являются “короткими” числами, задача их деления относится к разряду алгоритмов длинной арифметики, так как в результате вполне может получиться периодическая дробь, т.е. число, состоящее из бесконечного количества цифр.

Возможен двоякий подход к решению поставленной задачи.

Можно задаться целью получить результат в алгебраической форме, т.е. с выделением непериодической части и периода. Почти такая же задача уже была решена в главе 5 (см. задачу 5.1.7). Единственным отличием новой задачи от уже решенной является то, что при делении произвольного натурального числа на другое натуральное число соответствующая данному действию обыкновенная дробь может оказаться неправильной и результат будет содержать ненулевую целую часть.

Поэтому после считывания значений делимого (m) и делителя (n) следует распечатать значение $m\ div\ n$ (в Turbo Pascal) или m/n (в С) и запятую, потом заменить m на остаток от такого деления ($m = m\ mod\ n$, или $m\% = n$ в Turbo Pascal и в С соответственно), а уж затем непосредственно использовать решение задачи 5.1.7.

```
  {r[i]=a[i]+b[i]+pr;
   pr=r[i]/10;
   r[i]%=10;
   }
 if (pr>0)
   {(*L)++;
    r[*L]=pr;
   }
}
```

8.3. Умножение "длинного" числа на "короткое"

Для выполнения такого умножения будем использовать один и тот же массив s описанного в примере 8.3 типа ALONG в качестве как входного параметра, так и результата умножения.

В первом элементе этого массива расположены четыре младшие цифры "длинного" числа. Переменная L будет содержать значение текущей длины этого массива. "Короткое" же число находится в целочисленной переменной n. Несмотря на то, что и число n, и элементы массива s можно представить в целом типе с максимальным числом разрядов (longint в Turbo Pascal и long в C), максимального значения, предусмотренного для этого типа, они достигать не могут, так как результат умножения любого ненулевого элемента массива s на n также не должен превышать максимального значения того же типа.

В приведенном ниже примере подобной подпрограммы каждый элемент входного и выходного массива s должен быть меньше 10 000, т.е. состоять не более чем из одной цифры 10 000-ичной системы счисления (четырех десятичных цифр). Значение n при этом может достигать 2^{31} : 10 000 ≈ 200 000. При уменьшении количества цифр у элементов s значение n может быть увеличено, и наоборот. Само же умножение производится последовательно от младшего разряда к старшему в 10 000-ичной системе счисления, каждая цифра которой записывается как число в десятичной системе, смешанной с ней (см. параграф 4.3).

Подпрограмма на языке Turbo Pascal

```
procedure Mult(var s:along;var
L:longint;n:longint);
{умножаем массив s длины L на число n}
Var
  pr,i:longint;
Begin
  pr:=0; {перенос в старший 10000-ичный разряд}
  for i:=1 to L do
    begin
      s[i]:=s[i]*n;
      s[i]:=s[i]+pr;
      pr:=s[i] div 10000;
      s[i]:=s[i] mod 10000;
    end;
  while pr>0 do
    begin
      inc(L);
      s[L]:=pr mod 10000;
      pr:=pr div 10000
    end
End;
```

Подпрограмма на языке C

```
void Mult(ALONG s,long *L,long n)
/* умножаем массив s длины *L на число n */
{long i,pr=0;
/* перенос в старший 10000-ичный разряд */
 for (i=1;i<=*L;i++)
   {s[i]*=n;
    s[i]+=pr;
    pr=s[i]/10000;
    s[i]%=10000;
   }
 while (pr>0)
   {(*L)++;
    s[*L]=pr%10000;
    pr/=10000;
   }
}
```

Заметим, что результат мы получили в 10 000-ичной системе счисления, и при его выводе следует дополнять каждый элемент результирующего массива до четырех десятичных цифр ведущими нулями. Пример такого вывода содержится, например, в решении задачи 8.7.1, использующем подпрограмму Mult.

8.4. Умножение "длинного" числа на "длинное"

Для умножения двух "длинных" чисел существует несколько различных алгоритмов. Один из них заключается в умножении каждой цифры первого множителя на каждую цифру второго и размещении произведения в соответствующем разряде результата.

Если в каком-то разряде результата оказывается число, большее, чем максимальная цифра соответствующей системы счисления (9 в десятичной системе), то производится перенос в старший разряд. Такой алгоритм соответствует умножению двух чисел столбиком.

Приведем тексты программ, реализующих подобный алгоритм. Считывание "длинных" чисел производится из файла INPUT.TXT, а результат умножения помещается в файл OUTPUT.TXT.

Входные данные в подобных задачах удобнее представлять в файле во избежание ошибки ввода длинного числа с клавиатуры. Результат также проще анализировать по текстовому файлу, а не по его выдаче на экран.

Подпрограмма ReadData является примером ввода из файла "длинного" числа в случае, когда заранее неизвестна его длина. Применение в языке Turbo Pascal логической функции SeekEoln вместо Eoln позволяет игнорировать пробелы в конце строки, при вводе "длинного" числа. В языке C такую ситуацию требуется отслеживать самостоятельно. После считывания числа из файла в первом элементе массива оказывается старшая цифра "длинного" числа. Результат же мы будем получать в массиве, первый элемент которого соответствует младшей цифре, а в нулевом элементе хранится текущая длина массива.

Программа на языке Turbo Pascal

```
Type
 AByte=array[1..10000] of byte;
 TDest=array[0..20000] of word;
Var
 aLen,bLen:word;
 a,b:AByte;
 c:TDest;
```



```
      y:=10*y mod n
    end;
  writeln(')');
  readln
End.
```

Решение на языке Basic

```
10 INPUT "Введите числитель дроби: ", M%
20 INPUT "Введите знаменатель дроби: ", N%
30 X% = M%
40 FOR I% = 1 TO N%
50 X% = 10 * X% MOD N%
60 NEXT I%
'нашли (N+1)-й остаток при делении M на N
'столбиком,запомним его и будем дальше
'искать равный ему
70 Y% = X%
80 L% = 0
90 L% = L% + 1: X% = 10 * X% MOD N%
100 IF X% = Y% THEN 110 ELSE 90
110 X% = M%
120 FOR I% = 1 TO L%
130 X% = 10 * X% MOD N%
140 NEXT I%
150 Y% = M%
'сравнивая остатки, отстоящие друг от друга
'на длину периода, определим непериодическую
'часть
160 IF X% = Y% THEN 170 ELSE 190
170 PRINT "Непериодическая часть отсутствует"
180 GOTO 260
190 PRINT "Непериодическая часть равна: ";
200 IF X% = Y% THEN 250
210 X% = 10 * X% MOD N%:
220 PRINT MID$(STR$(10 * Y% \ N%), 2, 1);
230 Y% = 10 * Y% MOD N%
240 GOTO 200
250 PRINT
260 PRINT "Период равен: ";
270 FOR I% = 1 TO L%
280 PRINT MID$(STR$(10 * Y% \ N%), 2, 1);
290 Y% = 10 * Y% MOD N%
300 NEXT I%
310 PRINT
320 END
```

Решение на языке C

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main(void)
{long m,n,x,y,i,k;
 printf("Введите числитель дроби: ");
 scanf("%ld",&m);
 printf("Введите знаменатель дроби: ");
 scanf("%ld",&n);
 x=m;
 for(i=0;i<n;i++)
  x=(10*x)%n;
 y=x;
 k=0;
 do
 {k++;
  x=(10*x)%n;
 } while(x!=y);
 x=m;
 for(i=0;i<k;i++)
  x=(10*x)%n;
 y=m;
 if(x!=y) printf("Непериодическая часть
равна: ");
 else printf("Непериодическая часть
отсутствует\n");
 while(x!=y)
 {x=(10*x)%n;
  printf("%d",(y*=10)/n);
   y%=n;
  }
 printf("\nПериод равен: \n");
 for(i=0;i<k;i++)
 {printf("%d",(y*=10)/n);
  y%=n;
 }
 printf("\n");
 getch();
}
```

Задача 5.1.9

Периодическую P-ичную дробь, целая часть которой равна 0, переведем в обыкновенную дробь, записанную в десятичной системе счисления, воспользовавшись третьим из алгоритмов, описанных в разделе 3.3. Перевод же обыкновенной дроби в десятичную произведен в решении задачи 5.1.7, поэтому здесь мы приводить его не будем, ограничившись печатью числителя и знаменателя полученной обыкновенной дроби. Сокращать эту дробь также не будем. Значение P-ичной дроби вводится в следующем формате: 0, $x\ldots x(x\ldots x)$, где вместо произвольного количества символов x должны стоять цифры P-ичной системы, не превосходящие 9 (значение же P может быть и больше 10), непериодическая часть может отсутствовать. Для решения описанной задачи в программе сначала ищутся числитель x и знаменатель z обыкновенной дроби для непериодической части, затем числитель y и знаменатель q для периода. Окончательный же результат равен $\frac{x}{z}+\frac{y}{zq}$.

Решение на языке Turbo Pascal:

```
Var
  s:string;
  i,P:byte;
  x,y,z,q:longint;
Begin
  write('Введите значение P: ');
  readln(P);
  write('Введите P-ичную дробь: ');
  readln(s);
  x:=0;{числитель для непериодической части}
  i:=3;{с этого символа строки начинается
        дробная часть}
   z:=1;{знаменатель для непериодической части}
  while s[i]<>'(' do {пока не начался период}
    begin
      x:=x*P+ord(s[i])-ord('0');
       z:=z*P;
       i:=i+1
    end;
```

```
  i:=i+1;
 y:=0;{числитель обыкновенной дроби для периода}
 q:=0;{знаменатель обыкновенной дроби для периода}
  while s[i]<>')' do {пока не кончился период}
   begin
     q:=q*P+(P-1);
    y:=y*P+ord(s[i])-ord('0');
     i:=i+1
   end;
 writeln('Числитель обыкновенной дроби
          равен: ',x*q+y);
 writeln('Знаменатель обыкновенной дроби
          равен: ',z*q);
  readln
End.
```

Решение на языке Basic

```
10 INPUT "Введите значение P: ", P%
20 INPUT "Введите P-ичную дробь: ", S$
30 X% = 0'числитель для непериодической части
40 I% = 3'с этого символа начинается дробная часть
50 Z% = 1'знаменатель для непериодической части
60 IF MID$(S$, I%, 1) = "(" THEN 100
'пока не начался период
70  X% = X% * P% + ASC(MID$(S$, I%, 1)) –
ASC("0")
80  Z% = Z% * P% : I% =I% + 1
90 GOTO 60
100 I% = I% + 1
110 Y% = 0
'числитель обыкновенной дроби для периода
120 Q% = 0
'знаменатель обыкновенной дроби для периода
130  IF MID$(S$, I%, 1) = ")" THEN 170
'пока не кончился период
140 Q% = Q% * P% + (P% – 1)
150 Y% = Y% * P% + ASC(MID$(S$, I%, 1)) –
ASC("0")
160 I% = I% + 1: GOTO 130
'преобразуем две полученные обыкновенные
'дроби в одну
170 PRINT "Числитель обыкн. дроби: "; X% * Q% + Y%
180 PRINT "Знаменатель обыкн. дроби: "; Z% * Q%
190 END
```

Решение на языке C

```
#include <stdio.h>
#include <conio.h>
char str[256],*s=str;
long p,x,y,z,q;
void main(void)
{printf("Введите значение P: ");
 scanf("%ld",&p);
 printf("Введите P-ичную дробь: ");
 scanf("%s",s);
 for(x=0,s+=2,z=1;*s!='(';z*=p,x=x*p+*s–'0',s++) ;
 for(s++,y,q;*s!=')';q=q*p+(p-1),y=y*p+*s–'0',s++) ;
 printf("Числитель обыкн. дроби равен:
        %ld\n",x*q+y);
 printf("Знаменатель обыкн. дроби равен:
        %ld\n",z*q);
 getch();
}
```

Задача 5.1.11

Решение на языке Turbo Pascal

```
Var
  x:integer;
Begin
  write('Введите десятичное число: ');
  readln(x);
  write(x,'=');
  while x>=1000 do
    begin write('M');x:=x–1000 end;
  if x>=900 then
    begin write('CM');x:=x–900 end;
  if x>=500 then
    begin write('D'); x:=x–500 end;
  if x>=400 then
    begin write('CD');x:=x–400 end;
  while x>=100  do
    begin write('C'); x:=x–100 end;
  if x>=90 then
    begin write('XC');x:=x–90  end;
  if x>=50  then
    begin write('L'); x:=x–50  end;
  if x>=40  then
    begin write('XL');x:=x–40  end;
  while x>=10   do
    begin write('X'); x:=x–10  end;
  if x>=9   then
    begin write('IX');x:=x–9   end;
  if x>=5   then
    begin write('V'); x:=x–5   end;
  if x>=4   then
    begin write('IV');x:=x–4   end;
  while x>=1    do
    begin write('I'); x:=x–1   end;
  readln
End.
```

Решение на языке Basic

```
10 INPUT "Введите  целое десятичное число: ", X%
20 IF X% < 1000 THEN 40
30 PRINT "M"; : X% = X% – 1000: GOTO 20
40 IF X% >= 900 THEN PRINT "CM"; : X% = X% – 900
50 IF X% >= 500 THEN PRINT "D"; : X% = X% – 500
60 IF X% >= 400 THEN PRINT "CD"; : X% = X% – 400
70 IF X% < 100 THEN 90
80 PRINT "C"; : X% = X% – 100: GOTO 70
90 IF X% >= 90 THEN PRINT "XC"; : X% = X% – 90
100 IF X% >= 50 THEN PRINT "L"; : X% = X% – 50
110 IF X% >= 40 THEN PRINT "XL"; : X% = X% – 40
120 IF X% < 10 THEN 140
130 PRINT "X"; : X% = X% – 10: GOTO 120
140 IF X% >= 9 THEN PRINT "IX"; : X% = X% – 9
150 IF X% >= 5 THEN PRINT "V"; : X% = X% – 5
160 IF X% >= 4 THEN PRINT "IV"; : X% = X% – 4
170 IF X% < 1 THEN 200
180 PRINT "I";
190 X% = X% – 1: GOTO 170
200 PRINT
210 END
```


Значения элементов массива a[2] и a[5] оказались однозначными числами, так как они содержат еще и значащий ноль в качестве второй цифры. У последнего же элемента массива ведущий ноль значимым не является. Текущая длина заполненного массива равна шести.

Таким образом, необходимая длина массива уменьшается практически в два раза по сравнению с предыдущим способом. Так как 100-ичная система счисления является смешанной с десятичной (см. главу 4), то, как это будет показано ниже, сами арифметические действия сложнее от такого представления не станут. Работу с "длинными" числами можно сделать еще более эффективной, если элементами ("цифрами") в их представлении будут служить числа, состоящие из четырех цифр. Именно такая величина элементов массива позволяет даже при выполнении операции умножения всегда оставаться в рамках целочисленной арифметики.

Пример 8.3. Покажем, как можно описать массив для записи 1000-значных "длинных" чисел:

Turbo Pascal

```
Type
  along=array[1..250] of longint;
```

C

```
typedef long[250] ALONG;
```

Если в каждом элементе такого массива записать четырехзначное число, то "длинное" число окажется представленным в системе счисления с основанием 10 000, смешанной с десятичной. Описание такого массива на Turbo Pascal можно дополнить нулевым элементом (в C он уже существует), в котором будет храниться текущая длина числа (количество 10 000-ичных цифр).

Во всех приведенных выше способах представления "длинных" чисел мы заранее фиксируем максимально допустимую длину числа, реальная же его длина, которая не может превышать размерности массива, хранится либо в нулевом элементе массива, либо в отдельной переменной.

Длина числа будет подсчитываться автоматически, если для его представления использовать строковый тип данных (string в Turbo Pascal). Несмотря на то, что элементами этого типа служат символы, код каждого из символов является обычным десятичным числом. Поэтому, например, при записи числа из примера 8.2 в строковый тип мы получим строку, состоящую из 6 символов с кодами 78, 0, 56, 34, 2 и 1 соответственно. А все операции, включая печать результата, проводятся над кодами символов, из которых такая строка будет состоять.

Избежать излишнего расхода памяти при задании "длинных" чисел массивами максимально возможной длины можно с помощью динамических структур данных, таких, как списки из цифр (например, 10 000-ичных). Недостатком же этого представления будет отсутствие непосредственного доступа к произвольному элементу такой структуры данных, что, впрочем, особенно и не требуется при реализации арифметических операций. А запись "длинного" числа в список, организованный по правилам работы со стеком, при считывании данных позволяет разместить младшую цифру в первом элементе списка.

Однако подобный способ представления "длинных" чисел делает программу, оперирующую такими числами, менее прозрачной и усложняет ее отладку. Поэтому данный вариант мы рассматривать не будем.

Рассмотренные выше способы представления относятся лишь к "длинным" целым числам. Если же требуется получить точный результат при операциях с конечными дробными числами, то можно воспользоваться теми же самыми приемами, запомнив место для запятой в отдельной переменной, и учесть при печати результата. Такой способ соответствует операциям над вещественными числами, записанными в формате с фиксированной запятой.

Ниже мы рассмотрим арифметические операции с "длинными" целыми числами. Те же самые алгоритмы после минимальной модификации можно использовать и для операций над вещественными числами с фиксированной запятой.

8.2. Сложение двух "длинных" чисел

Напишем подпрограмму сложения двух "длинных" положительных чисел, представленных в виде массивов a и b типа ABYTE, описанного в примере 8.1. Будем считать, что каждый элемент такого массива содержит всего лишь одну десятичную цифру. Первый элемент массива содержит младшую цифру. Результат будем помещать в массив r того же типа.

Входным параметром такой подпрограммы является также максимальная из длин слагаемых L, на выходе же эта переменная равна количеству цифр в результате сложения, то есть может быть на единицу больше своего первоначального значения, поэтому она является как входным, так и выходным параметром.

Сложение производится с помощью стандартного алгоритма сложения столбиком от младшего разряда к старшему, с переносом единицы в старший разряд, если в результате сложения двух цифр и значения переноса из младшего разряда получается число, большее девяти.

Подпрограмма на языке Turbo Pascal

```
Procedure Add(var a,b,r:abyte;var L:word);
{складываем массивы a и b максимальной длины L}
Var
  pr:byte;
   i:word;
Begin
  pr:=0; {перенос}
  for i:=1 to L do
    begin
      r[i]:=a[i]+b[i]+pr;
      pr:=r[i] div 10;
      r[i]:=r[i] mod 10
    end;
  if pr>0 then
    begin
      inc(L);
      r[L]:=pr
    end
End;
```

Подпрограмма на языке C

```
void Add(ABYTE a,ABYTE b,ABYTE r,unsigned int *L)
/*складываем массивы a и b максимальной длины *L */
{char pr=0; /* перенос */
 unsigned int i;
 for (i=1;i<=*L;i++)
```

10. Произведите следующие арифметические действия над двоичными нормализованными числами согласно правилам вещественной компьютерной двухбайтной арифметики (где на мантиссу отводится (n) 10 бит, а на порядок — (k) 5 бит) и сравните полученные результаты с их истинными значениями:

1) $0{,}11111_2 \times 2^0 + 0{,}111111_2 \times 2^{-5}$;
2) $0{,}11111_2 \times 2^0 - 0{,}11111_2 \times 2^{-8}$;
3) $0{,}111111_2 \times 2^{16} \times 0{,}111111_2 \times 2^{15}$;
4) $0{,}1111_2 \times 2^2 : 0{,}11_2 \times 2^4$.

11. *(Государственная академия управления им. Серго Орджоникидзе)*

Выполнить действия над машинными кодами чисел:

а) с фиксированной точкой в 16-разрядном (целочисленном) формате:

1) $X = A + B$, где $A = -483$, $B = 216$;
2) $X = A + B$, где $A = 204$, $B = -231$;
3) $X = A + B$, где $A = 289$ и $B = -490$;

б) с плавающей точкой и смещенным порядком в 32-разрядном вещественном формате:

1) $Y = X + C$, где $X = -267$, $C = -21{,}25$;
2) $Y = Z + C$, где $Z = -25{,}5625$ и $C = 12\ 1/2$;
3) $Y = C + D$, где $C = -17{,}768$ и $D = -259{,}4$.

Результаты X и Y записать в разрядных сетках соответствующих форматов.

Глава 8. “Длинная” арифметика

8.1. Способы представления “длинных” чисел

Как было показано в предыдущих двух главах, арифметические действия, выполняемые компьютером в ограниченном числе разрядов, не всегда позволяют получить точный результат. Более того, для дробных десятичных чисел это практически невозможно, так как они, вообще говоря, не представимы конечными двоичными дробями, которыми оперирует компьютер в вещественной арифметике.

Мы можем гарантировать точный результат, если будем производить только операции сложения, вычитания и умножения над целыми числами, представленными в типе с максимальным количеством разрядов, отведенных под мантиссу (`Extended` или `Comp` в Turbo Pascal). При этом количество значащих цифр в результате выполнения таких действий не превышает точности этого типа (18 значащих цифр для приведенных типов).

Помимо перечисленных, точными являются операции целочисленного деления с остатком, которые реализованы в компьютере над целыми числами, если количество значащих цифр в общем случае не превышает девяти. В остальных же случаях для получения точного результата придется самостоятельно программировать те или иные алгоритмы. Если количества двоичных разрядов в стандартных компьютерных типах данных для представления числа недостаточно, то назовем такое число ***“длинным”*** и приведем примеры работы с ним.

Первый из возможных способов представления таких чисел — это их запись с помощью массива десятичных цифр. Количество элементов такого массива равно количеству значащих цифр в числе. Если же впоследствии мы будем производить с таким числом арифметические операции, то размер массива должен быть достаточным, чтобы разместить в нем и результат. Так, при выполнении умножения длина результирующего массива может быть в два раза больше, чем длина массивов для множителей.

Приведем пример описания массива для записи “длинных” чисел в языках программирования Turbo Pascal и C:

Пример 8.1

Turbo Pascal

```
Type
  abyte=array[1..1000] of byte;
```

C

```
typedef unsigned char[1000] ABYTE;
```

При этом в первом элементе массива может располагаться как старшая (самая левая), так и младшая (правая) значащая цифра. Это зависит от конкретной задачи и вкуса программиста. Скажем, при считывании “длинного” числа из файла или при вводе с клавиатуры проще разместить старшую значащую цифру в первом элементе массива, так как именно ее значение будет получено первым. Если же число генерируется программным образом или его длина известна до считывания, то в первый элемент можно помещать и младшую цифру.

Последний способ записи “длинных” чисел предпочтительнее для организации тех арифметических действий, в которых обрабатываются цифры от младшей к старшей, например, при сложении и вычитании столбиком, а сами числа имеют различную длину.

Но в любом случае для описания конкретного “длинного” числа требуется еще один параметр — длина текущего числа, расположенного в подобном массиве. Этот параметр можно хранить в отдельной целочисленной переменной или в нулевом элементе массива. Однако для массива, описанного в примере 8.1, вариант хранения в массиве невозможен, так как значение любого, в том числе и нулевого, элемента такого массива не превосходит 255, а текущая длина массива может достигать 1000.

Показанный в примере 8.1 способ записи “длинных” чисел наиболее нагляден, поскольку число представлено в привычной нам десятичной системе счисления, однако не очень эффективен. При организации арифметических операций над подобными массивами процессор будет складывать, вычитать, умножать или делить лишь однозначные числа, а количество самих действий окажется достаточно большим.

Поэтому один из способов повышения эффективности “длинной” арифметики — это хранение в каждом элементе массива не одной, а двух или более цифр. Если элементы массива, как и в примере 8.1, состоят из одного байта, то есть не должны превосходить 255, то в каждом из них вполне можно разместить двузначное число, что соответствует записи исходного числа в 100-ичной системе счисления.

Пример 8.2. Покажем, как можно разместить число 10 234 560 078 в массиве типа `abyte`:

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
78	0	56	34	2	1

Решение на языке C

```
#include <stdio.h>
#include <conio.h>
void main(void)
{int x;
 printf("Введите целое десятичное число: ");
 scanf("%d",&x);
 for(;x>=1000;x-=1000)
  printf("M");
 if(x>=900)
 {printf("CM");
  x-=900;
 }
 if(x>=500)
 {printf("D");
  x-=500;
 }
 if(x>=400)
 {printf("CD");
  x-=400;
 }
 for(;x>=100;x-=100)
  printf("C");
 if(x>=90)
 {printf("XC");
  x-=90;
 }
 if(x>=50)
 {printf("L");
  x-=50;
 }
 if(x>=40)
 {printf("XL");
  x-=40;
 }
 for(;x>=10;x-=10)
  printf("X");
 if(x>=9)
 {printf("IX");
  x-=9;
 }
 if(x>=5)
 {printf("V");
  x-=5;
 }
 if(x>=4)
 {printf("IV");
  x-=4;
 }
 for(;x;x-)
  printf("I");
 getch();
 printf("\n");
}
```

Задача 5.1.13

Решение задачи для двух- или трехзначных чисел проводится аналогично решению задачи 1.4.11. Для N-значных чисел мы получаем полиномиальное уравнение $(N-1)$-й степени относительно P. Для шестизначного числа $a=a_5a_4a_3a_2a_1a_0$ оно имеет вид:

$$a=a_0P^5+a_1P^4+a_2P^3+\ a_3P^2+\ a_4P+a_5.$$

Если возникшее уравнение имеет натуральные корни (корень), отличные от 1, то они и будут основаниями искомых систем счисления. Все такие корни полинома являются делителями свободного члена полинома (в нашем случае это a_5-a). Таким образом, для решения нашего полиномиального уравнения в натуральных числах следует найти все делители десятичного положительного числа $a-a_5$ и проверить путем подстановки, являются ли они корнями нашего уравнения.

Задача 5.1.15

Несмотря на то, что все цифры чисел A и B по условию не превосходят 9, основания их систем счисления x и y могут быть как угодно большими. Это делает не просто неэффективным, но и невозможным процесс простого перебора всех пар оснований систем счисления. Поэтому перебор следует организовать лишь по одному из оснований, например x, и для каждого x, определив значение a получившегося числа A_x в десятичной системе счисления, решить, как и в задаче 5.1.13, полиномиальное уравнение относительно y. Коэффициентами полинома в данном случае будут являться цифры числа B, а свободный член равен (b_0-a).

Задача 5.1.17

При решении воспользуемся тем, что состояние часов в любой момент времени легко описывается с помощью факториальной позиционной системы счисления с основанием, равным 5 (о факториальных системах счисления говорится в параграфе 1.2). Тогда остается только найти такое время в промежутке $(T, T+P]$, которому соответствует число с максимальной суммой цифр в этой системе счисления, и вычесть из этой суммы сумму “марсианских” цифр числа T.

Заметим, что для приведенных в задаче ограничений количество цифр в числе, записанном в “марсианской” факториальной системе, не будет превышать 10. В массивах такой длины мы и будем хранить числа, записанные в факториальной системе счисления с основанием 5.

Решение на языке Turbo Pascal

```
Type
  mas=array[1..10]of integer;
Var
  Q1,Q2:mas;
{массивы для записи начального и конечного
значения времени в "марсианской" системе
счисления}
  T,P:longint;
  i,Sum:integer;
Procedure Trans(num:longint;Var Q:mas);
{переводит десятичное число в "марсианское"}
Var
  i:integer;
begin
  for i:=1 to 10 do
    Q[i]:=0;
  i:=1;
  while num>0 do
    begin
      Q[i]:=num mod (i+4);
     {выделяем очередную "цифру"}
```

```
      num:=num div (i+4);
      i:=i+1;
    end;
end;
Begin
  readln(T,P);
  Trans(T,Q1);
  Trans(T+P,Q2);
  i:=10;
  while (Q1[i]=Q2[i])and(i>=1) do
    i:=i–1;
  Sum:=0;
  if i<>0 then Sum:=Sum+(Q2[i]–Q1[i]);
  i:=i–1;
  while i>=1 do
    begin
      Sum:=Sum+(i+3–Q1[i])
       {i+3 – максимальная цифра в i-м разряде}
       i:=i–1
    end;
  writeln(Sum)
End.
```

Решение на языке C

```
#include <stdio.h>
typedef int mas[10];
void Trans(long num, mas Q)
/* переводит десятичное число в "марсианское"*/
{int i;
 for (i=0;i<10;i++)
   Q[i]=0;
 i=0;
 while (num>0)
 {Q[i]=num%(i+5);/*выделяем очередную
"цифру"*/
  num/=i+5;
  i++;
  }
}
void main()
{mas Q1,Q2;
 /* массивы для записи начального и конечного
   значения времени в "марсианской" системе
   счисления */
 long T,P;
 int Sum=0,i=9;
 scanf("%D%D",&T,&P);
 Trans(T,Q1);
 Trans(T+P,Q2);
 while ((Q1[i]==Q2[i])&&(i>=0))
   i–;
 if (i>=0) Sum+=Q2[i]–Q1[i];
 i–;
 while (i>=0)
 {Sum+=i+4–Q1[i];
   /*i+4 – максимальная цифра в i-м разряде*/
  i–;
 }
 printf("%d\n",Sum);
}
```

5.3. Работа с различными системами счисления в электронной таблице (на примере Excel)

Реализуем алгоритм перевода целых чисел из десятичной системы счисления в *P*-ичную с помощью электронной таблицы Excel.

Наиболее просто осуществляется перевод из десятичной системы в системы с меньшим основанием, например, в двоичную, так как для записи чисел в таких системах счисления не нужны никакие символы, кроме десятичных цифр.

Как известно, первая ячейка таблицы имеет координаты **A1**.

1. В ячейку **A2** введем число, которое надо перевести в *P*-ичную систему счисления, а в ячейку **B2** введем значение $P<10$. Подпишем эти ячейки в строке **1**.

2. В ячейку **A3** запишем формулу **=целое(A2/$B$2)**, т.е. получаем целую часть от деления на *P*. Для значения основания системы счисления *P* нужно использовать абсолютную адресацию **($B$2)**, чтобы при копировании этой формулы деление по-прежнему производилось на значение ячейки **B2**.

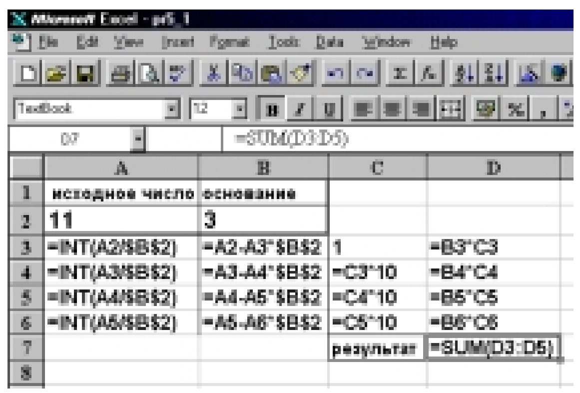
3. В ячейку **B3** запишем формулу **=A2–A3*$B$2**, т.е. получаем остаток от деления исходного числа на *P*, который является младшей цифрой результата.

4. Пометим ячейки **A3** и **B3** и скопируем их содержимое в ячейки **A4** и **B4** соответственно. В ячейке **B4** получим следующую (вторую справа) значащую цифру.

5. Если в последней ячейке столбца **A** получился 0, то перевод закончен, в противном случае повторяем действия, аналогичные п. 4, т.е. копируем формулы из последней заполненной строки в следующую, увеличивая количество строк в таблице.

Для получения *P*-ичной записи числа надо выписать все цифры из колонки **B** снизу вверх. Но этот процесс также можно автоматизировать и сразу получить искомое значение в привычной для нас форме. Для этого в столбце **C** на каждом шаге будем получать очередную степень 10, а в столбце **D** — результат умножения степени из столбца **C** на соответствующую цифру из столбца **B**. Окончательный результат перевода является суммой значений в столбце **D** и может быть получен с помощью стандартного средства таблицы Excel (суммирование значений столбца или строки).

Пример 5.1. Покажем, как может выглядеть электронная таблица в англоязычной версии Excel для перевода числа 12 в двоичную систему (в параметрах окна значком [✓] отмечены формулы).



	A	B	C	D
1	исходное число	основание		
2	11	3		
3	=INT(A2/$B$2)	=A2-A3*$B$2	1	=B3*C3
4	=INT(A3/$B$2)	=A3-A4*$B$2	=C3*10	=B4*C4
5	=INT(A4/$B$2)	=A4-A5*$B$2	=C4*10	=B5*C5
6	=INT(A5/$B$2)	=A5-A6*$B$2	=C5*10	=B6*C6
7			результат	=SUM(D3:D5)
8				

в) в результате вычитания возникают ненадежные значащие цифры, которые могут привести к серьезной потере точности и даже получению недостоверного результата;

г) добавление или вычитание малого числа может привести к результату, в точности равному большему операнду;

д) получение очень больших чисел может вызвать переполнение порядка, а очень малых — отрицательное переполнение или исчезновение числа (превращение в машинный нуль), что, как правило, заканчивается аварийным завершенем программы.

7.4. Задачи с решениями

1. Определить, сколько бит приходится на мантиссу в том или другом вещественном типе.

Решение на языке Turbo Pascal

```
Var
  a,b:double;{возможен любой вещественный тип}
  i:byte;{счетчик числа разрядов}
Begin
  a:=0.5;
  b:=0.0;
  i:=0;
  while b<>1 do
  begin
    b:=b+a;
    a:=a*0.5;
    i:=i+1;
  end;
  writeln('Количество разрядов в мантиссе равно ',i-1);
End.
```

Решение на языке C

```
#include <stdio.h>
void main()
{double a=0.5,b=0;
/*возможен любой вещественный тип*/
 int i=0; /* счетчик разрядов */
 while (b!=1)
   {i++;
    b+=a;
    a*=0.5;
   }
 printf("Количество разрядов в мантиссе равно %d\n",i-1);
}
```

2. Объяснить, почему в приведенной ниже программе на языках Turbo Pascal и C цикл не закончится при x=1, а будет выполняться до ошибки переполнения порядка у переменной x. (Разумеется, если вы запустите этот пример на компьютере, то дожидаться этого события придется, быть может, всю оставшуюся жизнь.)

Turbo Pascal

```
Var  x:real;
Begin
  x:=0;
  while x<>1 do
    begin
      x:=x+0.01;
      writeln(sin(x))
    end
End.
```

C

```
#include<stdio.h>
#include <math.h>
void main(void)
{float x=0;
 while (!(x==1))
   {x+=0.01;
    printf("%f\n",sin(x));
   }
}
```

Так как десятичное число 0,01 является бесконечной двоичной дробью, оно представляется в любом вещественном типе приближенно, а при многократном прибавлении такого числа ошибка только нарастает.

После стократного прибавления 0,01 к нулю точной единицы мы не получим. Следовательно, условие окончания цикла никогда не выполнится. Его можно заменить на следующее: $x < 1 - 0{,}001$. Вычитание 0,001 из 1 необходимо потому, что при представлении 0,01 в вещественном типе с недостатком после стократного выполнения цикла справедливо неравенство $1 - 0{,}001 < x < 1$, а уже $x + 0{,}01 > 1 - 0{,}001$. То есть при условии $x < 1$ цикл выполнится один лишний раз, а при предложенном условии ровно 100 раз.

7.5. Контрольные вопросы и упражнения

1. Каковы преимущества компьютерного представления чисел с плавающей точкой по сравнению с фиксированной, которую мы чаще всего используем в повседневной жизни?

2. Приведите к нормализованному виду следующие числа, используя в качестве *P* основания их систем счисления:

$-\ 0{,}000001011101_2$;
9876543210_{10};
$123456789{,}ABCD_{16}$.

3. Запишите в естественной форме с фиксированной запятой следующие нормализованные числа:

$0{,}1011_2 \times 16^1$; $0{,}1011_2 \times 2^1$;
$0{,}12345_{10} \times 10^{-3}$; $0{,}ABBA_{16} \times 16^{-2}$.

4. Почему современные языки программирования поддерживают одновременно несколько вещественных типов данных?

5. Для чего применяется сдвиг при записи порядка нормализованного числа и что такое машинный нуль?

6. Объясните, почему программа, приведенная в качестве решения задачи 7.4.1, действительно подсчитывает количество разрядов в мантиссе.

7. Исследуйте с помощью решения задачи 7.4.1 один из вещественных типов (кроме *real*) и запишите в нем нормализованное вами первое из чисел задачи 7.5.2.

8. Перечислите и объясните все ошибки, которые могут возникать при арифметических операциях с нормализованными числами в ограниченном числе разрядов.

9. Измените порядок приведенных ниже действий в десятичном калькуляторе с двумя разрядами для порядка так, чтобы переполнения порядка не происходило:

3.0E+60 × 4.0E+50 × 1.0E–30.

Так как при сдвиге мантиссы меньшего числа на 45 разрядов вправо она становится равной 0, то результат такого сложения окажется равным $0{,}1 \times 2^{25}$, то есть первому слагаемому.

2. Потеря крайней справа значащей цифры результата при сложении или вычитании мантисс (при сложении и вычитании двух чисел количество значащих цифр может увеличиться лишь на одну), это влияет на точность, но не на правильность результата.

Пример 7.12. Для 15-разрядных мантисс:
$0{,}111110000011111_2 + 0{,}1001001001001001_2 = 1{,}100010101000011_2$

То есть значащих цифр получилось уже 16 и после округления и нормализации результат будет выглядеть так:

$$0{,}110001010100010 \times 2^1.$$

Гораздо хуже обстоит дело, когда производится вычитание близких по модулю и имеющих одинаковый знак чисел (или сложение чисел разного знака). В этом случае достоверной может остаться всего лишь одна значащая цифра, а остальные цифры мантиссы (скорее всего нули) окажутся недостоверными.

Пример 7.13. Пусть требуется вычислить разность чисел $0{,}(0011)_2$ и $0{,}001100110011_2$.

Произведем вначале вычитание "на бумаге". Представим первую дробь (она периодическая) в виде $0{,}001100110011(0011)_2$. Перед периодом выписали 12 цифр, т.к. вторая дробь содержит 12 цифр. Произведем вычитание и получим $0{,}000000000000(0011)_2 = (1100)_2 \times 2^{-14}$. Переведем конечную периодическую дробь в десятичную систему счисления, воспользовавшись формулой суммы бесконечной убывающей геометрической прогрессии с $q = \dfrac{1}{2^4}$ и $b = \dfrac{1100_2}{2^4}$.

Получим
$0{,}(0011)_2 - 0{,}001100110011_2 = 0{,}(1100)_2 \times 2^{-14} = 0{,}8 \times 2^{-14}$.

При выполнении этой же операции вычитания в компьютере при условии, что под мантиссу отведено 12 разрядов, получим
$0{,}110011001101_2 \times 2^{-2} - 0{,}1100110011_2 \times 2^{-2} = 0{,}1_2 \times 2^{-13} = 1 \times 2^{-14}$,
то есть уже старшая значащая цифра результата оказалась неверной.

3. Выход за границу допустимого диапазона значений того или иного вещественного типа при нормализации результата. В данном случае получаемый порядок оказывается либо больше максимально возможного значения, либо меньше минимального. Данную ситуацию различные компиляторы и операционные системы обрабатывают по-разному, но чаще всего выполнение программы прерывается с сообщением об ошибке типа **"арифметическое переполнение"**.

Пример 7.14
$0{,}1 \times 2^{127} + 0{,}1 \times 2^{127} = 0{,}1 \times 2^{128}$,
и если максимальный представимый порядок у вещественного типа равен 127 (как в типе `real`), результат выходит за пределы допустимых чисел.

Рассмотрим теперь умножение и деление нормализованных чисел a и b в арифметике с ограниченным числом разрядов:

$$d = a \cdot b = (0, m_a \cdot 0, m_b) \times 2^{q_a + q_b};$$

$$f = a : b = (0, m_a : 0, m_b) \times 2^{q_a - q_b}.$$

При получении результата в данном случае производятся, как вы помните, следующие действия (не важно, какое из чисел — a или b больше):

1) мантиссы перемножаются (или одна делится на другую), при этом результат округляется до n значащих цифр, хотя при умножении их может получиться до 2n, а при делении — бесконечно много;

2) порядки при умножении складываются, а при делении вычитаются, и основная ошибка, которая может возникнуть в этот момент, — получение не представимого в используемом вещественном типе порядка, как положительного, так и отрицательного, т.е. уже описанное выше "арифметическое переполнение"; однако следует учитывать, что в десятибайтном вещественном типе диапазон допустимых порядков достаточно велик, чтобы производить практически любые вычислительные работы, и возникновение подобной ситуации скорее всего означает, что программа составлена неверно;

3) при необходимости мантисса результата нормализуется: для операции умножения возможен сдвиг мантиссы на один разряд влево (самый правый бит мантиссы при этом заполняется нулем, а не очередной значащей цифрой результата перемножения мантисс) с одновременным уменьшением порядка результата на единицу, а для операции деления может понадобиться сдвиг вправо вместе с увеличением порядка результата на единицу.

Таким образом, у вещественной арифметики есть несколько потенциально опасных свойств. Все они имеют общее происхождение, а именно тот факт, что мантисса и порядок в представлении с плавающей запятой занимают фиксированное число разрядов. Проиллюстрируем это еще раз на примере десятичного калькулятора с пятью значащими цифрами мантиссы и двумя цифрами порядка:

Реальные операции	Операции калькулятора	Погрешность
12,345 + 0,5432 = 12,8882	1.2345E+01+5.432E-01 = 1.2888E+01	0,0002
3,14159 − 3,1416 = − 0,00001	3.1416E+00 − 3.1416E+00 = 0.0E-00	0,00001
6,25·6,25 = 39,0625	6.25E+00·6.25E+00 = 3.9063E+01	− 0,0005
100,0 : 51,2 = 1,953125	1.0E+02 : 5.12E+01 = 1.9531E+00	0,000025

В приведенных примерах вычисления произведены неточно, а в случае с вычитанием ответ трудно считать даже приближенно верным.

Подведем итог всему сказанному о компьютерной вещественной арифметике:

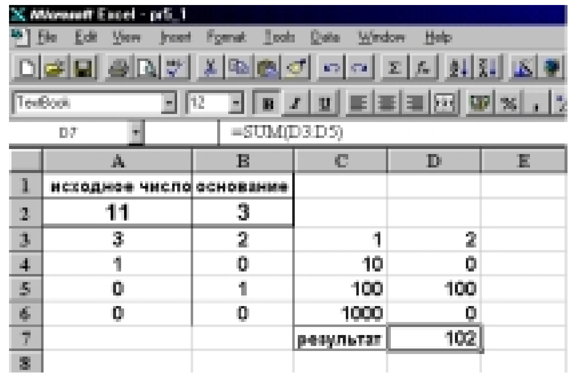
а) уже на стадии записи чисел в компьютер возникают ошибки округления, которые при выполнении арифметических действий только нарастают;

б) наличие погрешностей округления приводит к следующему правилу программирования: неразумно сравнивать в программе любую пару вещественных чисел на точное равенство;

Заметим, что получившуюся таблицу можно непосредственно использовать для перевода в P-ичную систему любых целых чисел, не превосходящих 15 (так как числа, меньшие шестнадцати, состоят не более чем из четырех двоичных цифр, а в системах счисления с большим основанием количество цифр может лишь уменьшиться). Для этого число 12 в ячейке **A2** нужно заменить на любое другое, не превосходящее 15, а число 2 в ячейке **B2** заменить на $P<10$.

В общем случае для перевода числа a наша таблица должна содержать m строк, где m удовлетворяет следующему условию: $P^{m-1} \le a < P^m$ (m — количество цифр в P-ичной записи числа), поэтому можно заранее заготовить достаточно длинную таблицу (по количеству строк), а при выписывании ответа отбрасывать все нижние строки, содержащие нули в обеих колонках.

Пример 5.2. Покажем, как будет выглядеть таблица из примера 5.1 при переводе числа 11 в троичную систему. При этом заменим формулы на их значения, использовав изменение вида окна отменой **[✓]** у формул в параметрах окна:


Microsoft Excel - p5_1
=SUM(D3:D5)

	A	B	C	D	E
1	исходное число	основание			
2	11	3			
3	3	2	1	2	
4	1	0	10	0	
5	0	1	100	100	
6	0	0	1000	0	
7			результат	102	
8					

Ответом в данном примере является число 102_3, а последняя строка оказалась лишней.

Покажем теперь, как можно переводить с помощью электронной таблицы P-ичное число в десятичную систему счисления.

1. В ячейку **A2** введем P-ичное число, которое надо перевести в десятичную систему счисления, а в ячейку **B2** введем значение $P<10$. Подпишем эти ячейки в строке **1**.

2. В ячейку **A3** запишем формулу **=целое(A2/10)**, т.е. получаем целую часть от деления на 10 исходного числа. Нижестоящие ячейки столбца **A**, который является вспомогательным для выделения цифр из исходного числа, будут получены копированием этой ячейки.

3. В ячейку **B3** запишем формулу **=A2-A3*10**, т.е. получаем последнюю цифру исходного числа, которая является коэффициентом при нулевой степени P в развернутом представлении целого P-ичного числа (1.7). В нижестоящих ячейках этого же столбца по скопированной из **B3** формуле будем получать следующие цифры исходного числа.

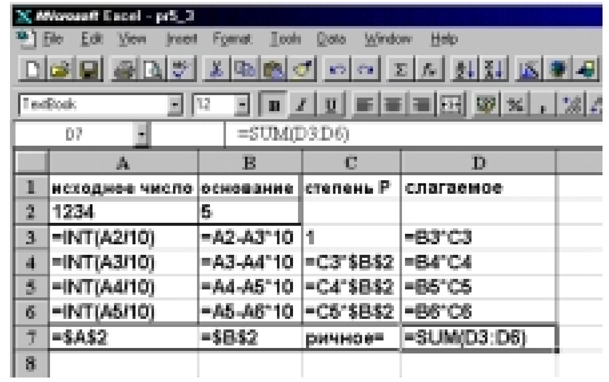
4. В ячейку **C3** поместим 1 — значение нулевой степени P. В ячейках столбца **C** (**C4, C5, C6** ...) будем получать очередные степени числа P, умножая значение вышестоящей ячейки на значение P. Для значения основания системы счисления P нужно использовать абсолютную адресацию (**$B$2**), чтобы при копировании этой формулы деление по-прежнему производилось на значение именно ячейки **B2**. Например, для **C4** формула будет выглядеть так: **=C3*$B$2**, а формулы во всех нижестоящих ячейках могут быть получены копированием.

5. В столбце **D** мы будем получать очередные слагаемые в представлении (1.7) исходного числа. Так, для **D3** формула будет иметь вид **=B3*C3**, а для нижестоящих ячеек столбца получится копированием формулы из **D3**.

6. Таким образом, четвертая строка таблицы (ячейки **A4–D4**) получится копированием третьей, пятая — копированием четвертой и т.д. Всего же для получения окончательного результата требуется столько строк, не считая первой, сколько значащих цифр имеет исходное число.

Результатом перевода является сумма значений в ячейках столбца **D**, которую можно получить с помощью стандартных средств Excel.

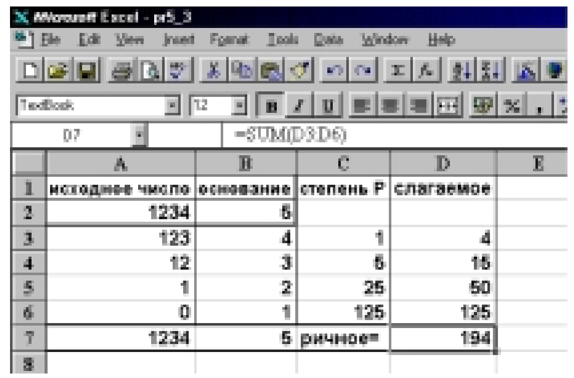
Пример 5.3. Покажем, как может выглядеть подобная таблица в англоязычной версии Excel для перевода пятеричного числа 1234 в десятичную систему (в параметрах окна значком **[✓]** отмечены формулы):


Microsoft Excel - p5_3
=SUM(D3:D6)

	A	B	C	D
1	исходное число	основание	степень P	слагаемое
2	1234	5		
3	=INT(A2/10)	=A2-A3*10	1	=B3*C3
4	=INT(A3/10)	=A3-A4*10	=C3*$B$2	=B4*C4
5	=INT(A4/10)	=A4-A5*10	=C4*$B$2	=B5*C5
6	=INT(A5/10)	=A5-A6*10	=C5*$B$2	=B6*C6
7	=$A$2	=$B$2	ричное=	=SUM(D3:D6)
8				

Получившуюся таблицу можно использовать для перевода любых четырехзначных P-ичных чисел, если P не превосходит 9.

Пример 5.4. Покажем вид таблицы из примера 5.3 в случае отмены формульного вида экрана:


Microsoft Excel - p5_3
=SUM(D3:D6)

	A	B	C	D	E
1	исходное число	основание	степень P	слагаемое	
2	1234	5			
3	123	4	1	4	
4	12	3	5	15	
5	1	2	25	50	
6	0	1	125	125	
7	1234	5	ричное=	194	
8					

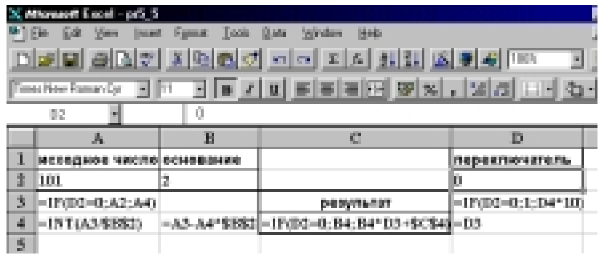
Результат вычислений находится в ячейке **D7**.

Приведенный в начале данного параграфа способ перевода чисел из десятичной системы счисления в P-ичную с помощью электронной таблицы демонстрирует наглядный, пошаговый метод перевода.

Ниже показан способ создания таблицы для перевода из десятичной системы в P-ичную с получением P-ичной записи результата. Как и ранее, $P<10$. В этой таблице используются более сложные конструкции, в частности, функция **ЕСЛИ** и циклические ссылки, которые позволяют автоматизировать перевод.

Пример 5.5. Данный пример показывает, как может выглядеть таблица для автоматического перевода чисел из десятичной системы в P-ичную. Она является аналогом компьютерной программы и вычисляет результат только после внесения определенного значения в ячейку **D2**, названную переключателем.

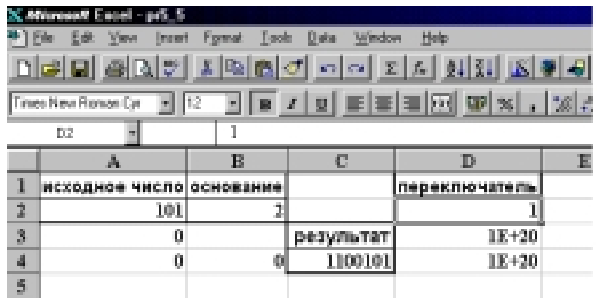
Не описывая принципы работы этой таблицы подробно, покажем, что именно вычисляет та или иная ячейка. Так, в ячейке **B4** на каждом шаге циклического алгоритма получается очередная цифра P-ичного представления исходного числа. В ячейке **D3** при значении переключателя, отличном от 0, вычисляется очередная степень 10, которая нужна для представления результата в привычной нам форме. Сам же результат формируется в ячейке **C4**, также при значении переключателя, не равном нулю.



	A	B	C	D
1	исходное число	основание		переключатель
2	101	2		0
3	=IF(D2=0;A2;A4)		результат	=IF(D2=0;1;D4*10)
4	=INT(A3/$B$2)	=A3-A4*$B$2	=IF(D2=0;B4;B4*D3+$C$4)	=D3
5				

Работать с этой таблицей следует так: в ячейки **A2** и **B2** вносятся значения исходного десятичного числа и основания новой системы счисления соответственно. Затем значение переключателя следует сделать равным сначала 0 (результаты предыдущих вычислений при этом обнулятся), а затем, например, 1. Тогда в ячейке **C4** можно прочитать значение результата. Если исходное число или основание системы счисления требуется изменить, то после изменения значений в **A2** или **B2** в переключатель **D2** опять следует занести сначала 0, а потом 1 и т.д.

Пример 5.6. Покажем, как будет выглядеть таблица из примера 5.5 после проведенных вычислений (формульный вид окна при этом отменен):



	A	B	C	D
1	исходное число	основание		переключатель
2	101	2		1
3	0		результат	1E+20
4	0	0	1100101	1E+20
5				

К сожалению, при работе с таблицей, показанной в примерах 5.5—5.6, количество итераций не определяется автоматически, оно равно фиксированному числу, устанавливаемому в меню электронной таблицы. То есть лишних действий, как и ранее в примере 5.2, избежать не удалось. Но таблица при любом количестве итераций остается компактной.

ЧАСТЬ III. Компьютерная арифметика

Глава 6. Целочисленная компьютерная арифметика

6.1. Компьютерное представление целых положительных чисел

Простейшими числовыми типами данных, с которыми оперируют ЭВМ, являются целые числа. Казалось бы, что мешает их рассматривать как вещественные с нулевой дробной частью? Тогда бы и любые вычисления сводились к арифметическим действиям над вещественными данными. Однако в целях:

1) эффективного расходования памяти,
2) повышения быстродействия,
3) введения операции деления нацело с остатком вместо приводящего к потере точности обычного деления вещественных чисел

— целые числа представляются в специальных форматах, которым в разных языках программирования соответствуют различные типы данных.

Введение специальных типов для целых чисел оправданно еще и потому, что для подавляющего большинства задач, решаемых с помощью ЭВМ, ничего и не требуется, кроме операций над целыми числами. Сюда относятся, например, задачи, данные в которых исключительно целочисленные: количество акций, сотрудников, деталей, транспортных средств и т.п. Целые числа используются для обозначения даты и времени, для нумерации различных объектов: элементов массивов, записей в базах данных, машинных адресов и т.п.

Для компьютерного представления целых чисел обычно используется несколько различных типов данных, отличающихся друг от друга количеством разрядов и наличием или отсутствием знакового бита. Чаще всего используется восьми-, шестнадцати- и тридцатидвухбитное представление таких чисел (один, два и четыре байта соответственно).

6.1.1. Представление положительных чисел в беззнаковых целых типах

Рассмотрим сначала представление положительных чисел в беззнаковых целых типах данных.

Все биты в таких типах отводятся для записи двоичного представления целого положительного числа (левый бит для старшего разряда, а правый — для млад-

Таким образом, 6-байтовый тип `real` хорош тем, что:

1) за счет “неполного” выписывания мантиссы (старшая значащая единица всегда “в уме”) достигается максимально возможная точность представления чисел;

2) за счет представления порядка с использованием сдвига упрощена операция сравнения произвольного вещественного числа с нулем.

В схеме для представления сопроцессорных вещественных типов мантисса и порядок меняются местами, оставляя при этом неизменным расположение знакового бита:

	q+t (k бит)	m (n бит)

Бит знака мантиссы

В этих типах порядок также представляется с использованием сдвига t.

Для представления чисел в этих типах данных (для типа `Extended` всегда, а для остальных сопроцессорных типов — в случае, когда $t + q = 0$) все числа предварительно преобразовываются к следующему формату:

$$a = \pm\, 0,m \times 2^q,\ 0 \le m < 2^n,$$
$$t = 2^{k-1} - 2\ ,\ -t \le q \le\ t + 3.$$

Величина сдвига вычисляется по принципу, описанному для типа `real`: максимально возможный порядок плюс сдвиг должны дать максимально возможное положительное беззнаковое число, записанное в k разрядах, а минимально возможный порядок плюс сдвиг должны дать ноль, т.е. k нулей.

Как легко видеть, в этих типах минимальная ненулевая мантисса состоит из $n - 1$ нуля после запятой и одной самой правой единицы. Отсюда и возникает несимметричность в допустимом диапазоне значений (на него теперь влияет не только порядок, но и мантисса).

Когда говорят о точности представления вещественных чисел, надо помнить следующее: десятичное число, имеющее даже всего лишь одну значащую цифру после запятой, не всегда можно записать точно в любом из вещественных типов. Объясняется это тем, что конечные десятичные дроби часто оказываются бесконечными периодическими двоичными дробями (см. параграфы 1.5 и 3.5).

Так, $0{,}1_{10} = 0{,}0(0011)_2$, а значит, и в нормализованном виде такое двоичное число будет иметь бесконечную мантиссу и не может быть представлено точно. При записи подобной мантиссы в ячейку компьютера число не усекается, а ***округляется***. Если под мантиссу отведено n бит и $n + 1$-я значащая цифра двоичной нормализованной мантиссы равна 0, то цифры, начиная с $n + 1$-й, просто отбрасываются, если же $n + 1$-я цифра равна единице, то к целому числу, составленному из первых n значащих цифр мантиссы, прибавляется единица.

Пример 7.9. Рассмотрим, как будет выглядеть запись мантиссы числа $a = 0{,}1_{10}$ при двоичной нормализации для различных значений n, где n — количество бит, отведенных под мантиссу:

$a = 0{,}1_{10} = 0{,}0(0011)_2 = 0{,}11(0011)_2 \times 2^{-3}$, то есть мантисса в нормализованном числе есть $0{,}11(0011)_2$; тогда при

$n = 10$, $m = 1100110011$ (остальные цифры мантиссы отброшены в результате округления);

$n = 12$, $m = 110011001101$ (последняя цифра изменилась с 0 на 1 при округлении);

$n = 13$, $m = 1100110011010$ (две последние цифры изменились при прибавлении 1).

7.3. Источники возникновения ошибок и потери точности при арифметических операциях

Как уже было показано в предыдущем параграфе на примере операций в ограниченном числе разрядов над десятичными нормализованными числами, даже если некоторые два таких числа были представлены в вещественном типе данных абсолютно точно, результат выполнения над ними арифметических операций часто может содержать погрешность, а иногда может быть и заведомо неверным.

Для того чтобы облегчить себе поиск подобных ошибок в программах, попробуем проанализировать алгоритмы выполнения арифметических операций с нормализованными числами в компьютере и понять природу возникновения таких ошибок.

Пусть $a = \pm\, 0,m_a \times 2^{q_a}$; $b = \pm\, 0,m_b \times 2^{q_b}$ — два нормализованных двоичных числа, и $q_a \ge q_b$ (в противном случае мы можем просто поменять их местами). Результатом их сложения или вычитания будет являться следующее выражение:

$$c = \left(0,m_a \pm 0,m_b \times 2^{q_b - q_a}\right) \times 2^{q_a}.$$

Как вы помните, порядок вычислений при этом таков:

1. Порядки чисел a и b выравниваются по большему из них (в нашем случае это q_a). Для этого мантисса числа b сдвигается на $q_a - q_b$ разрядов вправо (часть значащих цифр при этом оказываются утерянными), а его порядок становится равным q_a.

2. Выполняется операция сложения (вычитания) над мантиссами с округлением по значению $n + 1$-й значащей цифры результата.

Однако как при сложении, так и при вычитании мантисс может получиться как число, большее 1, так и меньшее $0{,}1_2$, т.е. нарушается нормализация.

Пример 7.10

$0{,}11_2 - (-\,0{,}101_2) = 1{,}011_2 = 0{,}1011_2 \times 2^1$;
$0{,}11_2 - 0{,}101_2 = 0{,}001_2 = 0{,}1_2 \times 2^{-2}$

В таком случае

3. Мантисса результата должна быть нормализована и получившийся после нормализации порядок может отличаться от q_a как в меньшую, так и в большую сторону.

Рассмотрим теперь ошибки, которые могут возникнуть при операциях сложения и вычитания (каждая возможная ошибка соответствует одному шагу алгоритма компьютерного сложения или вычитания вещественных чисел):

1. Потеря значащих цифр мантиссы у меньшего из чисел при выравнивании порядков. В худшем случае утерянными оказываются все значащие цифры и $a \pm b \equiv a$, что является абсурдным с точки зрения математики, но возможным в компьютерной арифметике с ограниченным числом разрядов.

Пример 7.11. Если в вещественном типе хранятся 40 значащих цифр, то

$$0{,}1_2 \times 2^{25} + 0{,}1_2 \times 2^{-20} = (0{,}1_2 + 0{,}1_2 \times 2^{-20-25}) \times 2^{25}.$$

робные схемы представления чисел некоторых из перечисленных типов.

Тип `real` не случайно приведен в таблице первым, хотя он и не обладает минимальным размером. Первые компиляторы с языка Turbo Pascal разрабатывались в те времена, когда персональные компьютеры сопроцессоров не имели, а система команд процессора содержала операции лишь над целыми числами. Реализацию арифметических операций над вещественными числами типа `real` разработчики компилятора вынуждены были проводить самостоятельно.

Они вынуждены были придумывать как можно более эффективные алгоритмы выполнения операций с этими числами, используя лишь операции основного процессора над целыми числами. В настоящее время работа с переменными типа `real` по-прежнему проводится с помощью этих алгоритмов, не использующих сопроцессор.

Поэтому схема для представления данных типа `real` отличается от схем для остальных вещественных типов, операции над которыми включены в систему команд сопроцессора. "Сопроцессорные" вещественные типы различаются между собой в основном лишь количеством бит, отводимых под мантиссу и порядок.

Как явствует из таблицы, любая переменная типа `real` занимает 6 байт памяти. Распределены они следующим образом:

47	46		8	7		0
		m (39 бит)			q+t (8 бит)	

Бит знака мантиссы

Здесь m — это цифры мантиссы в нормализованном двоичном представлении второго вида для вещественного числа, стоящие после единицы (после запятой в двоичной нормализации всегда стоит 1, и она в записи числа типа `real` в компьютере не участвует, но при реализации арифметических операций конечно же учитывается); q — это порядок нормализованного числа, а $t = 128$ — ***сдвиг*** порядка (подробнее о назначении сдвига и выборе его величины см. ниже). Любое число типа `real` имеет следующий нормализованный вид:

$$a = \pm\ 0{,}1m_2 \times 2^q,\ 0 \le m < 2^{39},\ -128 \le q \le 127.$$

Из приведенных неравенств следует, что при записи числа с минимально возможным порядком, равным 128, все разряды ячейки, отведенные под порядок, содержат нули ($q + t = -128 + 128 = 0$). При записи же числа с максимально допустимым порядком, равным 127, все разряды порядка в ячейке будут содержать 1 ($q + t = 127 + 128 = 255 = 256 - 1 = 2^8 - 1 = 11111111_2$).

Таким образом, в типе `real` порядок всегда представляется с помощью положительного беззнакового числа. Но если старший из k разрядов, отведенных под порядок, считать битом для знака порядка, то 1 в знаковом разряде порядка соответствует положительному и нулевому порядку, а 0 — отрицательному порядку (отличие от знакового разряда мантиссы). Действительно, если $q = 0$, то в k разрядах порядка будет записано число $128 = 10000000_2$.

В отличие от целых типов данных, для представления отрицательных действительных чисел дополнительный код не применяется, на месте мантиссы всегда размещается ее модуль, а ее знак, являющийся и знаком всего числа, определяется значением старшего бита: 0 для положительных чисел и 1 для отрицательных.

Минимальное значение мантиссы в приведенной выше схеме равно $0{,}1_2$ (все 39 разрядов, отведенных под мантиссу, при этом равны 0), а максимальное — $0{,}\underbrace{11\ldots1}_{39}{}_2 = 1 - 2^{-40}$.

Значит, минимальное представимое неотрицательное число типа *real* равно $0{,}1_2 \times 2^{-128}$. При его записи все 48 бит заполнены нулями. Такое число называется ***машинным нулем***, то есть минимальное представимое число считается равным нулю. Хотя, строго говоря, нулем оно не является, это довольно маленькое число и мы вполне можем считать его таковым.

Как было показано выше, у машинного нуля, т.е. у минимального числа, представимого подобным образом, порядок равен нулю. Обнуление порядка достигается за счет прибавления положительной константы t, называемой сдвигом порядка. Величина же сдвига зависит от количества разрядов, отводимых под порядок. При описанном представлении порядка ($q + t$) упрощается операция сравнения произвольного числа с нулем, которая аппаратно выполняется довольно часто.

Представить же ноль сам по себе согласно предложенной схеме мы не можем, т.к. в мантиссе всегда неявным образом присутствует единица. Кроме того, компьютерные числа $+0$ и -0 при таком представлении оказываются различными, а именно: у числа (-0) в самом старшем бите стоит знаковая единица, а остальные биты нулевые. Это может приводить к неправильной работе программы, поэтому сравнивать вещественные числа на точное равенство в большинстве случаев некорректно (вместо сравнения на равенство правильнее требовать, чтобы модуль разности сравниваемых чисел не превосходил некоторого числа, соответствующего значению погрешности вычислений).

Максимальное представимое число равно

$$(1 - 2^{-40})2^{127} \approx 2^{127} = 2^{(10\cdot 12{,}7)} \approx 1000^{12{,}7} = 10^{(3\cdot 12{,}7)} = 10^{38{,}1}.$$

Полученное число и обозначает правую границу допустимого диапазона значений, указанного в приведенной таблице. Аналогично определим левую границу диапазона допустимых значений:

$$0{,}1_2 \times 2^{-128} = 1 \times 2^{-129} = 2^{(-12{,}9\cdot 10)} \approx 1000^{-12{,}9} = 10^{(-12{,}9\cdot 3)} = 10^{-38{,}7} \approx 10^{-39}.$$

Следует помнить, что указанный диапазон говорит лишь о величине представимых чисел в том или ином вещественном типе и вовсе не означает, что мы можем точно представить, например в типе `real`, число, состоящее из 38 значащих десятичных цифр. На точность представления, как уже было сказано ранее, влияет размер мантиссы. В данном случае мантисса может состоять из 40 значащих двоичных цифр (39 из них записываются явно, а одна подразумевается). Это позволяет записать число, состоящее из 40 единиц, то есть

$$2^{40} - 1 \approx 2^{40} = 2^{10\cdot 4} \approx 10^{3\cdot 4} = 10^{12},$$

что и обеспечивает указанную в таблице точность 11—12 значащих десятичных цифр.

шего). Диапазон допустимых значений в каждом из таких типов зависит от количества разрядов, отведенных под представление числа.

Поясним это подробнее. Как вы знаете, числа в компьютере записываются при помощи 0 и 1, поэтому в ячейку из k бит мы можем записать набор нулей и единиц, рассматривающийся как двоичное число.

Минимальное число, которое можно записать в k битах, будет состоять из k нулей, т.е. это число 0. Максимальное же будет состоять из k единиц. Это число (k единиц в двоичной системе счисления) в десятичной системе счисления равно $2^k - 1$. Действительно, запишем это двоичное число в развернутой форме (1.7)

$$2^{k-1} + 2^{k-2} + \ldots + 2^2 + 2 + 1 = \frac{2^{k-1}\cdot 2 - 1}{2 - 1} = 2^k - 1$$

(здесь использована формула суммы конечной геометрической прогрессии).

Таким образом, для беззнаковых типов нижняя граница диапазона значений всегда равна нулю, а верхнюю границу диапазона допустимых значений можно подсчитать, зная количество бит, занимаемых элементами данного типа.

Приведем значения верхних границ диапазонов для различных беззнаковых типов, указывая число разрядов и названия соответствующих типов, реализованных в языке программирования Turbo Pascal:

Количество бит	Максимальное значение	Название в Turbo Pascal
8	255 ($2^8 - 1$)	`byte`
16	65535 ($2^{16} - 1$)	`word`
32	4294967295 ($2^{32} - 1$)	тип отсутствует

Заметим, что в языке программирования Object Pascal (среда Delphi 2.0 и выше) введен 4-байтный беззнаковый тип данных `Cardinal` с максимальным значением 2 147 483 647 (значения такого типа занимают 31 бит из 32 возможных), в языке С 32-битный беззнаковый тип обычно присутствует и обозначается как `unsigned long`, хотя размер того или иного типа может отличаться в различных компиляторах языка С.

6.1.2. Представление положительных чисел в знаковых целых типах

Целые положительные числа можно представлять и в знаковых типах данных. В них самый левый бит отводится на запись знака числа (0 — для положительных чисел и 1 — для отрицательных), остальные биты заняты его двоичным представлением (младший разряд в самом правом бите). Отрицательные числа при этом могут записываться и в дополнительном коде. Подробное описание представления отрицательных целых чисел в компьютере дано в параграфе 6.2.

При одинаковом числе разрядов в знаковых и беззнаковых типах допустимый диапазон для первых будет практически в два раза меньше, так как один бит у них отводится под знак числа. Например, максимальное значение для восьмибитных знаковых типов равно 127 (2^7—1), для 16-битных — 32 767 ((2^{15}—1) и т.д. В остальном же правила представления целых положительных чисел остаются теми же, что и для беззнаковых типов. Достаточное число примеров представления целых положительных чисел приведено в теме 8 части I.

Следует отметить, что ***разрядность процессора*** (число бит памяти, доступной для одновременной обработки процессором) на современных персональных компьютерах, не говоря уже о рабочих станциях и суперЭВМ, больше или равна четырем байтам. Поэтому использование целых типов меньшего размера оправдано лишь при работе с многомерными массивами и базами данных. Использование же четырехбайтных типов сделает программу, написанную для 32-разрядной операционной системы, более эффективной. Поэтому в языке Object Pascal наиболее употребимый целый тип `Integer` уже определен как четырехбайтный знаковый тип. Во многих компиляторах языка С тип `int` также реализован как четырехбайтный.

6.2. Целые отрицательные числа и дополнительный код

Очевидно, что отрицательные числа представимы лишь в знаковых типах данных. В большинстве компьютеров целые отрицательные числа представляются ***дополнительным кодом,*** который является записью в k разрядах положительного числа $2^k - |m|$, где m — целое отрицательное число, по модулю не превосходящее 2^{k-1}. Положительные же числа всегда записываются с помощью ***прямого кода***, соответствующего их двоичному представлению, так, как это было описано в предыдущем параграфе. Алгоритм получения дополнительного кода описан в параграфе 8.3 части I.

6.2.1. Диапазоны изменения значений для знаковых типов

Итак, целые числа со знаком для k-разрядных типов принадлежат диапазону $[-2^{k-1}, 2^{k-1} - 1]$, который не является симметричным относительно 0, что следует учитывать при программировании. Если, например, изменить знак у наибольшего по модулю отрицательного числа, то полученный результат окажется уже непредставимым в том же числе разрядов.

Выпишем значения границ диапазонов для знаковых типов с различной разрядностью и укажем их название в Turbo Pascal:

Разрядность	Минимум	Максимум	Название в Turbo Pascal
8	− 128	127	`shortint`
16	− 32768	32767	`integer`
32	− 2147483648	2147483647	`longint`

Заметим, что при решении некоторых задач даже четырехбайтный диапазон для представления целых чисел может оказаться недостаточным и придется использовать один из вещественных типов данных, или "длинную арифметику” (подробнее о которой см. главу 8). На некоторых ЭВМ возможна также поддержка целочисленных операций с 48 или 64-разрядными целыми числами.

Для полноты рассмотрения получим дополнительный 8-битный код для чисел – 128 (минимальное представимое число), – 127 (число, обратное максимально представимому) и – 0:

Число	– 128	– 127	– 0
Прямой код модуля	1000 0000	0111 1111	0000 0000
Обратный код	0111 1111	1000 0000	1111 1111
Дополнительный код	1000 0000	1000 0001	0000 0000

Отметим, что для числа – 128 прямой код совпадает с дополнительным, а дополнительный код числа (– 0) совпадает с обычным нулем. Однако при преобразовании обратного кода для числа (– 0) в его дополнительный код правила обычной двоичной арифметики оказываются нарушенными.

А именно:

$$1111\ 1111_2 + 1 = 1\ 0000\ 0000_2 = 2^k.$$

Но, так как описанные действия производятся в k двоичных разрядах, левая единица результата оказывается лишней и игнорируется, т.е. ***в k-разрядной целочисленной компьютерной арифметике*** $\mathbf{2^k \equiv 0}$.

6.3. Организация арифметических действий в ограниченном числе разрядов

Как уже было показано в предыдущем параграфе, целочисленная арифметика в ограниченном числе разрядов несколько отличается от обычной. А именно: левые цифры результата арифметических действий, выходящие за отведенное тому или иному типу количество разрядов, при вычислениях игнорируются, т.е. обычно они оказываются утерянными. Кроме того, при сложении, например, двух положительных чисел, представленных в знаковом типе данных, мы можем получить отрицательное число, если в результате сложения в левом знаковом бите окажется единица.

Пример 6.1. Выполним сложение 100 и 51 в восьмибитном знаковом типе.

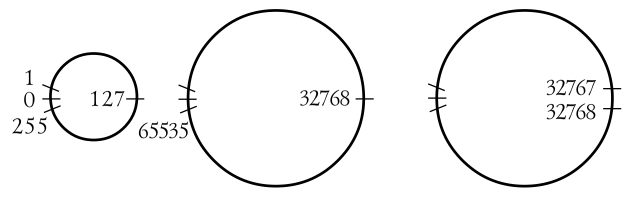
В восьмибитном компьютерном представлении эти числа имеют следующий вид: $100 = 0110\ 0100_2$ и $51 = 0011\ 0011_2$. При сложении этих чисел получим $1001\ 0111_2$. Самая левая единица (знаковый разряд) указывает на то, что в 8 разрядах получено отрицательное число. Так как все отрицательные числа в машине представляются дополнительным кодом, то для получения десятичного значения этого отрицательного числа надо выполнить следующее (алгоритм получения исходного числа по его дополнительному коду):

```
1) из дополнительного кода вычтем единицу
   10010111
 -        1
   10010110 (получили обратный код);
2) инвертируем полученный код: 01101001
(получили модуль отрицательного числа);
3) переведем полученное значение в десятичную систему:
```

$$01101001_2 = 2^6 + 2^5 + 2^3 + 1 = 64 + 32 + 8 + 1 = 105.$$

Таким образом, при сложении двух положительных чисел, используя восьмибитный знаковый тип, мы получили отрицательное число! То есть $100 + 51 = -105$ в *восьмиразрядной знаковой арифметике*.

Рассмотрим этот удивительный факт подробнее. Наглядным представлением для любого целого типа данных является кольцо, состоящее из расположенных по порядку констант этого типа, причем рядом с максимальным числом в том или ином типе находится минимальное, например:



8-битный беззнаковый 16-битный беззнаковый 16-битный знаковый

6.3.1. Прибавление и вычитание единицы

Результатом прибавления единицы в арифметике с фиксированным количеством разрядов является число, следующее по часовой стрелке за исходным в кольце, а результатом вычитания единицы – число, следующее против часовой стрелки. Данный факт вполне естественен, но только не тогда, когда происходит переход от максимально допустимого числа к минимальному при прибавлении единицы и от минимального к максимальному при вычитании. Что фактически и показано в примере 6.1.

Получается, что операции сложения, вычитания и умножения в k-битных беззнаковых типах соответствуют ***арифметическим операциям по модулю*** $\mathbf{2^k}$ (***сравнение по модулю*** $\mathbf{2^k}$), что, разумеется, вполне может привести к ошибкам вычислений.

6.3.2. Прибавление и вычитание произвольного числа

Прибавление или вычитание произвольного числа n соответствует n единичным шагам вдоль кольца от исходного числа в нужном направлении. Так, в восьмибитном беззнаковом типе $254 + 4 = 2$, $200 - 255 = 201$, а в шестнадцатибитном знаковом $32760 + 10 = -32766$, $-32760 - 10 = 32766$. Проверить это можно и непосредственно, выполняя действия сложения в двоичной k-разрядной арифметике. Вычитание же можно заменить на сложение с отрицательным числом, представляя последнее как дополнение его модуля до 2^k, вне зависимости от наличия знакового бита у типа. Только у беззнаковых типов дополнительный код для отрицательных чисел, по модулю превосходящих 2^{k-1}, будет иметь 0 в самом левом бите, что для знаковых типов недопустимо. Так, в 8-битном беззнаковом типе для приведенных примеров мы получим:


```
  10^3 ×0,23619
+ 10^-3×0,91824
```

Так как порядки у этих чисел различны, то требуется произвести предварительную операцию их выравнивания. После выполнения выравнивания порядков складываться будут следующие числа:

```
  10^3×0,23619
+ 10^3×0,00000091824
  10^3×0,23619091824
```

После сложения в мантиссе оказалось более пяти значащих цифр, и при записи в ячейку памяти произойдет округление результата до $10^3 \times 0{,}23619$. Получившееся число равно первому слагаемому, то есть при выравнивании порядков все значащие цифры мантиссы второго слагаемого потеряны. Таким образом, мы получили результат, невозможный с точки зрения обычной математики: $a + b = a$ при $b > 0$. Но в компьютерной арифметике с ограниченным числом разрядов такой результат вполне возможен, и об этом необходимо помнить при составлении алгоритмов решения задач.

Пример 7.7. Выполним умножение двух вещественных чисел:

$$10^3\times0{,}23000\cdot10^7\times0{,}95000.$$

При умножении двух вещественных чисел в представлении с плавающей запятой порядки складываются, а мантиссы перемножаются. В результате умножения получим следующий результат: $10^{10} \times 0{,}21850$. Это число не уместится в отведенный формат: в нашем калькуляторе под порядок отводится один разряд, а в получившемся числе порядок содержит две цифры. Выполнение операции умножения над этими числами приведет к прекращению выполнения программы в связи с ошибкой: ***переполнение порядка***.

Пример 7.8. Выполним деление двух вещественных чисел:

$$10^4\times0{,}92000:10^7\times0{,}30000.$$

При делении вещественных чисел в представлении с плавающей запятой порядки вычитаются, а мантиссы делятся одна на другую. В нашем примере при делении мантисс мы имеем бесконечную периодическую дробь: $0{,}92 : 0{,}3 = 3{,}0(6)$. Следовательно, при формировании мантиссы результата произойдет его округление. После нормализации получившееся число будет иметь следующий вид: $10^{-2} \times 0{,}30667$.

7.2.2. Математические сопроцессоры в современных персональных компьютерах

Важнейшей характеристикой любого компьютера является его быстродействие. Причем для ряда решаемых на компьютере задач одним из самых критичных параметров является скорость выполнения операций с плавающей запятой. Даже самые мощные универсальные процессоры тратят на такие вычисления достаточно много времени, т.к. они могут работать только с целыми числами, а для работы с вещественными числами (в формате с плавающей запятой) обычно используются специально написанные для конкретного транслятора подпрограммы. Поэтому вполне логичным было создание специального устройства – ***микросхемы математического сопроцессора***, предназначенного для работы с вещественными числами.

Математический сопроцессор представляет собой специализированную интегральную схему, работающую во взаимодействии с центральным микропроцессором. Помимо четырех действий арифметики, сопроцессор вычисляет значения тригонометрических функций (синуса, косинуса, тангенса и т.д.), что существенно ускоряет решение математических задач. Вычисления с плавающей запятой сопроцессор выполняет в 10–50 раз быстрее, чем обычный процессор. Кроме того, точность представления чисел в сопроцессоре много выше. Еще одним достоинством сопроцессора является его способность работать с числами в разных форматах: с целыми, с плавающей запятой и даже с двоично-десятичными.

В настоящее время на компьютерах решается громадное количество самых разнообразных задач. И если для решения одних вполне можно обойтись без математического сопроцессора, то для других его отсутствие будет крайне нежелательным.

Если не рассматривать задачи физического или математического моделирования, то можно однозначно сказать, что сопроцессор требуется для работы с трехмерной графикой, издательскими системами, электронными таблицами, САПР, математическими пакетами и т.д. При работе же с большинством баз данных или обычными текстовыми редакторами использование сопроцессора не дает никаких ощутимых результатов. Бесполезным окажется сопроцессор и при работе сетевых операционных систем.

Заметим, что все процессоры фирмы Intel, начиная с i486 (за исключением i486SX), имеют встроенные сопроцессоры.

7.2.3. Представление вещественных чисел в IBM-совместимых компьютерах для языка Turbo Pascal

В языке Turbo Pascal существует 5 стандартных типов данных, оперирующих с вещественными числами. В описании языка о них можно почерпнуть следующую информацию:

Название	Размер	Точность значений	Диапазон
REAL	6 байт	11–12 зн. цифр	$10^{-39}..10^{38}$
SINGLE	4 байта	7–8 зн. цифр	$10^{-45}..10^{38}$
DOUBLE	8 байт	15–16 зн. цифр	$10^{-324}..10^{308}$
EXTENDED	10 байт	19–20 зн. цифр	$10^{-4932}..10^{4932}$
COMP	8 байт	19–20 зн. цифр	$-9{,}2\cdot10^{18}..9{,}2\cdot10^{18}$

Тип comp, который мы подробно рассматривать не будем, служит для записи только целых чисел и упомянут нами только потому, что операции над переменными такого типа производятся с помощью команд сопроцессора (или их эмуляции при отсутствии сопроцессора), предназначенных для вещественных чисел, а целочисленное деление с остатком не реализовано.

Для того чтобы разобраться в такой таблице, а следовательно, научиться ею пользоваться, приведем под-

Самое же маленькое положительное число, которое можно ввести в нашем калькуляторе, +1.000E−99. Таким образом, выражая порядок лишь двумя десятичными цифрами, можно представлять числа очень широкого диапазона.

7.2. Схема представления вещественных чисел в компьютере

7.2.1. Общие сведения

Как и для целых чисел, для представления действительных чисел в компьютере используется чаще всего двоичная система счисления (иногда двоично-шестнадцатеричная). Десятичное число, таким образом, сначала переводится в двоичную систему, а уж затем представляется в нормализованном формате с $P = 2$ (для двоично-шестнадцатеричной системы $P = 16$).

Независимо от используемых систем счисления существует два основных типа представления чисел в компьютере, называемых представлениями с ***фиксированной*** и с ***плавающей запятой***.

В типах данных, использующих представление чисел с фиксированной запятой, все разряды ячейки, кроме знакового разряда, служат для изображения разрядов чисел. Причем каждому разряду ячейки соответствует всегда один и тот же разряд числа, что и фиксирует место запятой перед определенным разрядом.

Такая система упрощает выполнение арифметических действий, но сильно ограничивает диапазон представимых в таком виде чисел. Чаще всего это бывает диапазон $-1 < x < 1$. Этот случай соответствует расположению запятой непосредственно перед старшим цифровым разрядом. Иными словами, если количество цифровых разрядов в машине равно n, основание системы счисления, используемой в машине, равно p, то состоянию ячейки машины

$$\pm \quad a_1 \quad a_2 \quad \ldots \quad a_n$$

соответствует число $\pm\left(\frac{a}{p}+\frac{a_2}{p^2}+\ldots+\frac{a_n}{p^n}\right)$.

Наибольшее по абсолютной величине число, которое может быть представлено в компьютере таким образом, равно $1 - p^{-n}$, наименьшее, отличное от нуля, $-\left(p^{-n}\right)$. Для представления чисел, не укладывающихся в этот диапазон, программисту надо вводить ***масштабные множители***, т.е. заменять истинные величины, участвующие в решении задачи, их произведениями на специально подобранные коэффициенты, что существенно усложняет решение поставленной задачи.

Поэтому для представления вещественных чисел в современных компьютерах формат данных с фиксированной запятой не применяется. Однако описанные в главе 6 целые типы данных вполне можно отнести именно к этому классу, считая, что положение запятой в них зафиксировано после самого правого разряда.

Чтобы избавиться от недостатков представления действительных чисел с фиксированной запятой, в современных ЭВМ принят способ их представления с ***плавающей запятой***. Этот способ представления опирается на ***нормализованную (экспоненциальную) запись чисел***. Он подробно описан в параграфе 9.2 части I.

В большинстве современных языков программирования доступно несколько вещественных типов. Однако основное различие между ними состоит лишь в количестве байтов, отводимых под переменную того или иного типа (обычно от четырех до десяти), а стало быть, и на мантиссу, и на порядок, что влияет как на точность вычислений, так и на допустимый диапазон представимых чисел.

Чем большая точность нам требуется, тем более "длинный" тип данных придется использовать, рискуя при этом натолкнуться на ограничения оперативной памяти при использовании массивов.

Так, при размещении переменных в одном сегменте памяти (64 килобайта = 65 536 байтам), мы можем описать всего один двумерный массив 100 × 100 шестибайтных вещественных чисел (тип `real` в Turbo Pascal): 100 × 100 × 6 байт = 60 000 байт, или уже только 80 × 80 десятибайтных чисел: 80 × 80 × 10 байт = = 64 000 байт.

Различные схемы представления вещественных чисел будут рассмотрены ниже на примере Turbo Pascal, т.к. простейшие версии Basic ограничиваются всего лишь одним вещественным типом, а в языке C реализация того или иного типа существенно зависит от типа ЭВМ и ее операционной системы и стандартом языка не описывается.

Однако основные проблемы, возникающие при использовании вещественной арифметики, можно продемонстрировать и на примере обычного калькулятора.

Напомним, что как в компьютере, так и в калькуляторе представление чисел с плавающей запятой существенно усложняет схему арифметического устройства, а порядок выполнения операций с такими числами описан в параграфе 9.3 части I.

Здесь же мы более подробно поясним методику выполнения арифметических операций на примерах. Предварительно условимся, что при записи вещественного числа в представлении с плавающей запятой один разряд отводится под десятичный порядок и пять разрядов — под десятичную же мантиссу.

Пример 7.5. Выполним сложение двух вещественных чисел, представленных в формате с плавающей запятой нашего калькулятора:

$$\begin{array}{r} 10^8\times0,23619 \\ +\ 10^8\times0,91824 \\ \hline \end{array}$$

Так как порядки у этих чисел одинаковы, то производить операцию выравнивания порядков не требуется. Операция сложения сводится к сложению мантисс.

$$\begin{array}{r} 10^8\times0,23619 \\ +\ 10^8\times0,91824 \\ \hline 10^8\times1,15443 \end{array}$$

В результате получено ненормализованное число. Требуется выполнить нормализацию путем сдвига вправо на один разряд и округление результата, т.к. в мантиссе будет шесть цифр, а в ячейке памяти под мантиссу отведено их только пять.

Ответ: $10^9\times0{,}11544$.

Пример 7.6. Выполним сложение с помощью нашего калькулятора двух вещественных чисел, одно из которых достаточно большое по сравнению со вторым:

254	11111110	200	11001000
4	+ 00000100	− 255	+ 00000001
254 + 4	= 1|00000010	200 − 255	= 11001001
Результат:	2	Результат:	201

6.3.3. Ошибки, связанные с конечной разрядностью арифметики

Умножение и деление, как это будет показано ниже, сводится в целочисленной арифметике к сложению и вычитанию, поэтому мы не будем их отдельно рассматривать.

Просто покажем на примере вычисления факториала, что ошибки, связанные с конечной разрядностью арифметики, возникают и при выполнении операции умножения. Так, при вычислении значения факториала в наиболее распространенном знаковом 16-битном типе (integer в Turbo Pascal) значение

$$7! = 1\cdot2\cdot3\cdot4\cdot5\cdot6\cdot7 = 5250 = 0001010010000010_2$$

уместится в шестнадцати разрядах, а значение

$$8! = 7!\cdot8 = 1010010000010_2\cdot1000_2 = \\ = 1010010000010000_2$$

— уже нет. Для его записи требуется как минимум 17 разрядов в знаковом типе. Если же записать значение 8! в шестнадцать разрядов, то самая старшая (значащая) единица попадает в знаковый разряд и трактуется как знак отрицательного числа. Таким образом, получившееся значение соответствует отрицательному числу − 25216.

Следует заметить, что и 32-разрядного целого типа хватает для того, чтобы получить лишь значение 12!, а для 13! результат будет неверным, хотя полученное в 32-разрядной знаковой арифметике число окажется положительным: 1 932 053 504. Дело в том, что при умножении 12! на 13 результат содержит 33 значащие двоичные цифры, то есть самая левая единица в 32-разрядной арифметике оказывается утерянной, а вторая слева цифра равна 0, что и определяет положительный знак результата. Полученное положительное значение может ввести в заблуждение, чего бы не случилось, будь результат отрицательным.

В зависимости от конкретного компилятора и опций (или директив) его запуска программы, содержащие подобные ошибки, могут или быть прерваны в момент возникновения ошибки — выхода результата за границу допустимого диапазона, или благополучно завершить свою работу, получив в итоге неверные результаты. Так, компиляторы Turbo Pascal версии 6.0 и ниже подобные ошибки не проверяют (контроль должен самостоятельно осуществлять программист), в версии же 7.0 введена ***директива компилятора*** `{$Q+}`, которая обеспечивает соответствующий контроль (при `{$Q-}` контроль отсутствует).

Более того, при отсутствии подобного контроля поиск ошибки может быть затруднен тем, что промежуточные вычисления чаще всего производятся в максимальном целом типе (обычно 32-разрядном) и лишь при присваивании переменной другого типа лишние разряды отбрасываются. Как следствие, сообщение ***отладчика*** о значении арифметического выражения не всегда дает правильный результат. Рассмотрим это на примере следующей программы.

Пример 6.2

```
{$Q-}
Var
  a:integer;
Begin
  a:=1*2*3*4*5*6*7;
  writeln('7!=',a);
  a:=a*8;
  writeln('8!=',a)
End.
```

Если после получения переменной a своего первого значения, равного 7!, мы посмотрим с помощью отладчика значение выражения `a*8`, то оно будет равно 40 320, в результате же второго присваивания значение a окажется равным − 25 216.

В заключение приведем еще один возникающий при программировании случай, в котором неконтролируемый ***выход за границу диапазона*** приводит к зацикливанию программы:

Пример 6.3

```
{$Q-}
Uses Dos;
Const runtime=20;
{желаемое время выполнения программы}
Var
  h,m,s,hs,h1,m1,s1,hs1:word;
  {16-битный беззнаковый тип}
  t,t1:longint; {32-битный знаковый тип}
Begin
  gettime(h,m,s,hs);{процедура из модуля Dos}
  {запрашиваем время начала программы, здесь
   h — часы, m — минуты,
   s — секунды, hs — сотые доли секунды}
   t:=h*3600+m*60+s;{начальное время в секундах}
  repeat
    {собственно программа}
    gettime(h1,m1,s1,hs1);
    {запрашиваем текущее время}
    t1:=h1*3600+m1*60+s1
    {текущее время в секундах}
  until (t1-t)>=runtime
  {пока время выполнения программы
   не достигнет желаемой величины}
End.
```

Данный пример показывает, как ограничить время выполнения программы, написанной на языке Turbo Pascal.

Однако при выполнении программы из примера 6.3 могут возникнуть как минимум две различные ошибки. Первая возникает в том случае, когда программа запущена незадолго до полуночи (вернее, когда на внутренних часах компьютера — время, предшествующее полуночи не более чем на `runtime` секунд) и значения `t1` окажутся меньше, чем `t`, а следовательно, программа не завершится в силу отрицательности выражения `t1 - t`. В этом случае нам не хватает разрядов для представления времени в системе, основанной на секундах, минутах, часах, днях и т.д. (мы же ограничились лишь секундами, минутами и часами).

Вторая ошибка связана с тем, что 16 разрядов недостаточно для представления текущего времени в секундах (например, 20 часов — это 72 000 секунд. В беззнаковом 16-битном типе это число будет представлено как 6464).

Конечно, переменные `t1` и `t2` описаны как 32-битовые, однако в правой части соответствующих операторов присваивания расположены переменные типа `word`, что и приведет к вышеописанной ошибке.

Зацикливание здесь произойдет в том случае, если программа начала свою работу в начале 18-го часа суток, когда время, выраженное в секундах, еще нормально укладывается в тип `word`, а вот `t1` в пределах того же часа будет уже вычислено неверно. Стало быть, `(t1 - t)` будет отрицательным.

Более того, в данном случае не существует числа, представимого в типе `word`, и при этом большего, чем `t` на `runtime` секунд. Это приведет к тому, что основной цикл в программе не завершится никогда. Если же директиву компилятора заменить на `{$Q+}`, то, наоборот, программа будет прервана досрочно в результате ошибки "арифметическое переполнение" и желаемый результат все равно достигнут не будет.

6.4. Алгоритмы машинного сложения и умножения

6.4.1. Операция прибавления единицы

Наиболее простой компьютерной арифметической операцией над целыми числами, обычно реализованной аппаратно — в виде встроенной команды процессора, является прибавление единицы. В большинстве языков программирования для выполнения такой операции существует даже специальное обозначение, отличное от обычного сложения.

Так, в Turbo Pascal это функция `succ(a)`, результат которой может быть помещен в любую переменную того же типа, что и `a` (например, `b:=succ(a)`), и процедура `inc(a)`, в результате выполнения которой значение именно переменной `a` окажется увеличенным на единицу.

В языке C это арифметические операции `a++` и `++a`. В обоих случаях значение самой переменной `a` увеличивается на единицу, но в первом случае результатом выражения считается значение `a` до его увеличения, а во втором случае — после. Рассмотрим, например, следующий фрагмент:

```
a=0; b=0; c=0;
b=++a; c=a++;
```

В результате выполнения данного фрагмента программы, написанной на языке C, значение переменной `b` будет равно единице, `c` останется равной нулю, а `a` будет равно двум (ее дважды увеличивали на единицу).

Запишем **алгоритм реализации аппаратного прибавления единицы**.

Любое целое число, записанное в `k` битах, будем обозначать как `<начало><последний разряд>`, где `<начало>` — это первые слева `k - 1` бит, а `<последний разряд>` — самый правый бит в записи числа. Тогда число является четным, если `<последний разряд>=0`, и нечетным в противном случае. Заметим, что запись числа в дополнительном коде этого правила не нарушает.

Рекурсивный алгоритм `SUCC(a, k)` прибавления единицы к `k` левым разрядам числа `a`:

```
1) если k = 0, то конец;
2) если a - четное, то вносим единицу
           в <последний разряд>; конец;
3) если a - нечетное, то вносим ноль
           в <последний разряд>;
   SUCC(a,k) = SUCC(<начало>,k - 1); конец.
```

Дольше всего алгоритм будет работать в том случае, когда все `k` разрядов исходного числа — единицы. В этом случае потребуется `k` шагов для получения нуля.

6.4.2. Операция сложения двух целых чисел

Теперь перейдем к описанию **"быстрого" алгоритма сложения**, который в общем случае более эффективен, чем побитовое сложение с переносом разряда двух двоичных k-разрядных чисел, эквивалентное сложению столбиком. "Быстрый" алгоритм использует такие элементарные операции, как анализ четности числа, прибавление единицы, а также умножение и деление на 2.

Первые две из этих операций мы уже рассмотрели. Умножение двоичного числа на $2 = 10_2$ соответствует его сдвигу влево на один разряд, а целочисленное деление на 2 — сдвигу вправо. "Лишние" разряды при сдвигах (например, самый правый разряд при сдвиге вправо) оказываются утерянными. Побитовые сдвиги чисел в компьютере также реализованы аппаратно. Причем эти команды, как и прибавление единицы, — самые быстрые, реализующиеся всего лишь за один такт процессора.

В зависимости от четности двух чисел a и b справедливы следующие тождества:

$$a, b \text{ — четные: } a + b = \left(\frac{a}{2} + \frac{b}{2}\right) \cdot 2;$$

$$a, b \text{ — нечетные: } a + b = \left(\frac{a-1}{2} + \frac{b-1}{2} + 1\right) \cdot 2;$$

$$a \text{ — четное, } b \text{ — нечетное: } a + b = \left(\frac{a}{2} + \frac{b-1}{2}\right) \cdot 2;$$

$$a \text{ — нечетное, } b \text{ — четное: } a + b = \left(\frac{a-1}{2} + \frac{b}{2}\right) \cdot 2.$$

Заметим, что операция $\frac{a-1}{2}$, как и $\frac{a}{2}$, соответствует сдвигу двоичного представления числа a вправо на один разряд. Например, пусть $a = 101_2$. Сдвиг вправо a на один разряд даст результат 10_2. Сдвиг $a - 1 = 101_2 - 1 = 100_2$ на один разряд вправо также даст результат 10_2. Обозначим операции $\frac{a}{2}$ и $\frac{a-1}{2}$, как $a >> 1$, а умножение a на 2, как $a << 1$ (что соответствует терминологии языка C).

Тогда **рекурсивный алгоритм сложения двух чисел `ADD(a, b)`** можно записать следующим образом:

```
1) если a = 0, то ADD(a,b) = b; конец;
2) если b = 0, то ADD(a,b) = a; конец;
3) a, b - четные:
   ADD(a,b) = ADD(a >> 1, b >> 1) << 1; конец;
4) a, b - нечетные:
   ADD(a,b) = ADD(a >> 1, b >> 1 + 1) << 1; конец;
5) a и b имеют различную четность:
   ADD(a,b) = ADD(a >> 1, b >> 1) << 1 + 1; конец.
```


Пример 7.1. Если нам необходимо работать со следующими числами:

$- 6392000000 \qquad - 639{,}2 \qquad - 0{,}0000006392,$

то на дисплее нашего калькулятора отобразить удастся лишь второе из них (первое число занимает 11 знакомест, второе — 5 знакомест и третье — 12 знакомест).

Однако на реальных калькуляторах мы увидим следующее:

$- 6.392E{+}12 \qquad - 6.392E{+}02 \qquad - 6.392E{-}08,$

где знак E читается как "умножить на десять в степени". Такая запись отражает ***нормализованную форму записи чисел.***

Определение нормализованной записи числа дано в параграфе 9.1 части I.

В нормализованной форме все числа записываются одинаково в том смысле, что запятая у них ставится в одном и том же месте — перед первой (самой левой) значащей цифрой. Величина же числа (то есть ее порядок) указывается отдельно, с помощью соответствующей степени основания системы счисления, в которой это число было записано изначально (или смешанной с ней). В результате количество цифр в мантиссе может оказаться меньше, чем число значащих цифр в исходном числе. Кроме того, следует заметить, что мантисса P-ичного числа обычно записывается в P-ичной же системе счисления, а порядок и само число P — в десятичной.

Пример 7.2

1) $0 = 0{,}0 \times 10^0$ (возможная нормализация нуля);

2) $3{,}1415926 = 0{,}31415926 \times 10^1$ (количество значащих цифр не изменилось);

3) $1000 = 0{,}1 \times 10^4$ (количество значащих цифр уменьшилось с четырех до одной);

4) $0{,}123456789 = 0{,}123456789 \times 10^0$ (запятую передвигать не нужно);

5) $0{,}0000107_8 = 0{,}107_8 \times 8^{-4}$ (количество значащих цифр уменьшилось с семи до трех);

6) $1000{,}0001_2 = 0{,}10000001_2 \times 2^4$ (количество значащих цифр уменьшить невозможно);

7) $AB{,}CDEF_{16} = 0{,}ABCDEF_{16} \times 16^2$ (количество значащих цифр уменьшить невозможно).

Так как первая (левая) цифра мантиссы нормализованного числа отлична от 0 (за исключением самого числа ноль), в двоичной системе счисления она всегда равна 1. Поэтому общую форму записи нормализованного двоичного числа, отличного от 0, можно переписать в следующем виде:

$$a = 0{,}1m \times 2^q.$$

Нормализация числа, записанного в системе счисления с основанием Q, может быть проведена и по основанию P. При этом необходимо, чтобы $Q \le P$, и если $Q \ne P$, то обычно эти системы счисления являются смешанными (о смешанных системах счисления см. главу 4).

Приведем примеры нормализации двоичных чисел в смешанной двоично-шестнадцатеричной системе счисления. Мантисса будет представляться в двоичной системе, а основание P при нормализации равно 16. Само P и порядок нормализованного числа будем записывать в десятичной системе. При использовании этой смешанной системы запятую в записи исходного двоичного числа мы можем сдвигать только на 0, 4, 8, 12, … разрядов вправо или влево, так как $16 = 2^4$ и четыре двоичные цифры составляют одну шестнадцатеричную.

Пример 7.3

1) $1111_2 = 0{,}1111_2 \times 16^1$ ($1111_2 = F_{16}$);

2) $1 = 0{,}0001_2 \times 16^1$;

3) $0{,}0101\ 1_2 = 0{,}0101\ 1_2 \times 16^0$ ($101_2 = 5_{16}$);

4) $0{,}0000\ 0010\ 1_2 = 0{,}0010\ 1_2 \times 16^{-1}$ ($10_2 = 2_{16}$).

При записи нормализованного числа в компьютере или калькуляторе под мантиссу и порядок отводится заранее фиксированное количество знакомест.

Очевидно, что максимально допустимое количество цифр в мантиссе и определяет ***точность, с которой может быть представлено число.***

Поясним это на примере гипотетического калькулятора, который производит вычисления в десятичной системе счисления. Пусть это будет калькулятор с десятью знакоместами на дисплее, причем мантисса в нем имеет 4 цифры, а порядок — две.

В отличие от стандартной нормализации, у калькуляторов первая значащая цифра, с которой и начинается мантисса, изображается перед запятой. Порядок при этом соответственно уменьшается на единицу. Такая форма записи нормализованных чисел позволяет экономить одно знакоместо, так как вместо нуля в целой части мы помещаем значащую цифру.

Пример 7.4

1) 248,53786 в калькуляторе превращается в +2.485E+02. Переводя последнее число в привычное представление с фиксированной запятой, получим +248,5;

2) −2485367860,0 в калькуляторе превращается в −2.485E+09. Переводя последнее число в привычное представление с фиксированной запятой, получим −2485000000;

3) 0,00024853786 в калькуляторе имеет вид +2.485E−04, то есть равно числу 0,0002485.

Следовательно, всякое нормализованное десятичное число, состоящее не более чем из четырех значащих цифр, можно представить в таком калькуляторе точно, а остальные числа — лишь приближенно.

В первом из примеров 7.4 абсолютная погрешность калькуляторного представления исходного числа составила 248,53786 − 248,5 = 0,03786. Во втором примере погрешность равна 2485367860 − 2485000000 = = 367860, а в третьем − 0,00024853786 − 0,0002485 = = 0,00000003786. Если бы мантисса калькулятора имела больше цифр, погрешность была бы меньше.

Несмотря на то, что абсолютная погрешность может принимать самые различные значения, относительная погрешность во всех трех примерах одна и та же и равна

0,03768/248,53786 = 367860/248567860 = =0,00000003786/0,00024853786 = **0,0001516**.

Как количество цифр в мантиссе определяет точность, так количество цифр, отводимых под запись порядка в компьютерном представлении вещественных чисел с плавающей запятой, определяет ***наибольшее*** и ***наименьшее представимое положительное число.***

Покажем это на примере того же самого калькулятора. В его формате порядок имеет две цифры, и наибольшее число, которое может быть в нем представлено, — +9.999E+99. В целочисленном формате нам пришлось бы записать ровно 100 цифр (четыре девятки и 96 нулей), что конечно же не поддерживается никакими стандартными типами переменных в языках программирования.

```
  /* максимально возможное число элементов
     множества любого типа равно 30 */
 printf("Введите число элементов во множестве:");
 scanf("%d",&n);
 for (j=0;j<n;j++)
   a[j]=j+1;/* массив можно заполнить
               произвольно */
 q=(long)1<<n; /* помещаем в q число 2^n */
 for (j=1;j<q;j++)
   /* цикл по всем подмножествам */
   {for (k=0;k<n;k++)
    if ((((long)1<<k)&j)!=0) printf("%.2f ",a[k]);
     printf("\n");
   }
}
```

5. Во входном файле INPUT.TXT находится нечетное количество целых чисел, по модулю не превосходящих 32 000. Причем известно, что каждое из чисел встречается в файле дважды (но каждое из двух одинаковых чисел может находиться в произвольном месте файла) и лишь одно из них пары не имеет. Определите это число за один просмотр файла, не используя массивов для хранения чисел.

Решение этой задачи основано на следующем свойстве логической операции XOR (^^ в C):

B xor B ≡ 0 => A xor B xor B ≡ A и B xor A xor B ≡ A

(это несложно доказать, используя таблицу всех возможных значений логических значений A и B). Эти же тождества справедливы и в случае, когда операция является битовой (^ в C), а A и B — числами. Несложно убедиться, что если мы последовательно применим операцию XOR ко всем числам из исходного файла, то в результате получим искомое число, которое в данном файле уникально. Теперь легко составить программу, которая будет верно работать и в том случае, когда в исходном файле находится всего одно число (оно же тогда и является результатом).

Решение на языке Turbo Pascal

```
Var
  f:text;
  x,y:integer;
Begin
  assign(f,'input.txt');
  reset(f);
  read(f,x);
  while not SeekEof(f) do
   {пока не достигнут конец файла}
    begin
      read(f,y);
      x:=x xor y
    end;
  writeln('Не имеет пары число ',x)
End.
```

Данная программа является также примером организации ввода из файла чисел в случае, когда заранее неизвестно их количество. Применение в языке Turbo Pascal логической функции SeekEof вместо Eof позволяет игнорировать пробелы и символы перевода строки в конце файла после ввода последнего числа. В языке C такую ситуацию требуется обрабатывать самостоятельно (в приведенной же ниже программе считается, что непосредственно за последним числом расположен признак конца файла).

Решение на языке C

```
#include <stdio.h>
void main()
{int x,y;
 FILE* fd;
 char* fname="input.txt";
 fd=fopen(fname, "rt" );
 fscanf(fd, "%d",&x);
 while (!feof(fd))
   /* пока не достигнут конец файла */
   {fscanf(fd, "%d",&y);
    x^=y;
   }
 printf("Не имеет пары число %d\n",x);
}
```

6.6. Контрольные вопросы и упражнения

1. От чего зависят границы диапазона чисел, представимых в том или ином целом типе?

2. Как будут представлены в 8-битном знаковом типе числа − 1, − 10, − 120?

3. Даны десятичные числа a и b, не превосходящие 127 (каждому школьнику дается индивидуальный вариант, в котором a и b подбираются так, чтобы сумма чисел была больше, чем 127, а их разность была отрицательна, например: $a = 59$, $b = 106$). Найти их сумму и разность в 8-разрядном беззнаковом и знаковом представлении. Ответ записать в таблицу. Так, для приведенного примера:

Числа	Представление числа			
	Десятичные	8-битные	Беззнаковые	Знаковые
a	59	00111011	59	59
b	106	01101010	106	106
$a+b$	165	10100101	165	− 91
$a-b$	− 47	11010001	209	− 47

4. Какие ошибки возникают при сложении чисел в ограниченном числе разрядов? Как можно их избежать?

5. Может ли результат умножения двух положительных чисел, представимых в знаковом типе данных, оказаться отрицательным? Если да, то приведите пример.

Глава 7. Вещественные числа и компьютер

7.1. Нормализованный вид вещественных (действительных) чисел

Если при представлении целых чисел в компьютере ограничением может служить лишь величина записываемого числа, то при записи вещественного числа речь в первую очередь идет о точности его представления, то есть о количестве значащих цифр, которые удается сохранить в ограниченном числе разрядов.

Допустим, мы имеем калькулятор, в котором на экране дисплея для вывода чисел есть только 10 знакомест (включая знак числа и точку между целой и дробной частью действительного десятичного числа).

Проиллюстрируем работу данного алгоритма.

Пример 6.4. Сложим числа $11=1011_2$ и $14=1110_2$. Вначале выполняется рекурсивный спуск:
ADD(1011,1110) = ADD(101,111) << 1 + 1 (правило 5);
ADD(101,111) = ADD(10,11 + 1) << 1 = = ADD(10,100) << 1 (правило 4);
ADD(10,100) = ADD(1,10) << 1 (правило 3);
ADD(1,10) = ADD(0,1) << 1 + 1 (правило 5);
ADD(0,1) = 1 (правило 1).

Зная значение последнего выражения в построенной цепочке рекурсивных вызовов, мы можем в обратном порядке подсчитать значение исходного выражения:

ADD(1,10) = 1 << 1 + 1 = 10 + 1 = 11_2;
ADD(10,100) = 11 << 1 = 110_2;
ADD(101,111) = 110 << 1 = 1100_2;
ADD(1011,1110) = 1100 << 1 + 1 =
= 11000 + 1 = 11001_2 = $2^4 + 2^3 + 2^0$ =
= 16 + 8 + 1 = 25.

6.4.3. Операция вычитания

Операцию вычитания отдельно можно не описывать, т.к. любое вычитание можно заменить на сложение с числом противоположного знака, т.е. $a - b = a + (-b)$. Получаемые при этом отрицательные слагаемые следует перевести в дополнительный код. Если же, наоборот, $b < 0$ и $(-b) > 0$, исходный дополнительный код нужно перевести в прямой для $|b|$ и выполнять операцию $a + |b|$.

6.4.4. Операции сравнения двух целых чисел

Операции сравнения целых чисел обычно реализуются с помощью вычитания. Так, сравнение на равенство или неравенство двух чисел может быть сведено к сравнению их разности с машинным нулем (числом, все разряды которого нулевые). Сравнения на неравенство (<, >, ≤, ≥) аналогично могут быть сведены к проверке знака у разности этих чисел (т.е. к анализу значения самого левого бита разности).

6.4.5. Операция умножения

Объясним, как можно реализовать в двоичной арифметике умножение. “Быстрый” вариант обыкновенного умножения был известен еще в Древнем Египте, его также называют “русским”, или “крестьянским”, методом.

Пример 6.5.

```
 23 × 43 = 23+46+184+736=989
 46  21
 92  10 (четное)
184   5
368   2 (четное)
736   1
```

Здесь первый столбец состоит из результатов последовательного умножения первого слагаемого на два, а второй — последовательного целочисленного деления второго слагаемого на два. В результирующую же сумму вошли те из чисел первого столбца, рядом с которыми во втором столбце стоят нечетные числа.

Такое умножение основано на следующих тождествах:

a — четное: $a \times b = \frac{a}{2} \times (b \times 2) = \left(\frac{a}{2} \times b\right) \times 2$;

a — нечетное: $a \times b = \frac{a-1}{2} \times (b \times 2) + b = \left(\frac{a-1}{2} \times b\right) \times 2 + b$.

В результате получаем следующий **рекурсивный алгоритм умножения** целых двоичных чисел MULT(a, b):

```
1) если a = 0, то MULT(a, b) = 0; конец;
2) a - четное:
   MULT(a, b) = MULT(a >> 1, b) << 1; конец;
3) a - нечетное:
   MULT(a, b) = MULT(a >> 1, b) << 1 + b; конец.
```

При реализации такого алгоритма в ограниченном числе разрядов к неверному результату могут привести операции сдвига влево (если старший бит ненулевой) и сложения.

Пример 6.6. Умножим в 8-битном беззнаковом типе. $14 = 1110_2$ на $100 = 1100100_2$:
MULT(1110,1100100)=MULT(111,1100100)<<1;
MULT(111,1100100)=MULT(11,1100100)<<1+ +1100100;
MULT(11,1100100)=MULT(1,1100100)<<1 + 1100100;
MULT(1,1100100) = 1100100;

Проводя все вычисления в 8 битах, получим:
MULT(11,1100100) = 11001000 + 1100100 = 101100 (вместо 100101100, т.к. левая единица находится в 9-м разряде);
MULT(111,1100100) = 1011000 + 1100100 = 10111100;
MULT(1110,1100100) = 10111100 << 1 = 1111000_2 = 120_{10} (при сдвиге влево единица в старшем разряде исчезла).

Заметим, что полученное число равно остатку от деления $14 \times 100 = 1400$ на $256 = 2^8$, то есть умножение произведено по модулю $256 = 2^8$.

Определим также, в каком случае при умножении чисел в знаковых типах полученный результат является верным с точки зрения обычной арифметики.

• Если оба множителя положительны или отрицательны, то их произведение должно быть меньше, чем 2^{k-1}.

• Если оба числа a и b отрицательны, то с помощью дополнительного кода они записаны как $2^k - |a|$ и $2^k - |b|$, и их произведение равно

$$(2^k - |a|)(2^k - |b|) = 2^{2k} - 2^k(|a| + |b|) + |a|{\cdot}|b|,$$

что в k-разрядной арифметике соответствует просто $|a|{\cdot}|b|$, так как остальные слагаемые выходят за пределы k разрядов. Стало быть, надо, чтобы $|a|{\cdot}|b| \le 2^{k-1}$.

• Если же множители имеют разный знак, например, $a < 0$, $b > 0$, то их произведение можно представить как

$$(2^k - |a|){\cdot}|b| = 2^k|b| - |a|{\cdot}|b|,$$

но в k разрядах это просто $2^k - |a|{\cdot}|b|$ (то есть дополнительный код числа $-|a|{\cdot}|b|$) и для получения правильного результата умножения тоже достаточно, чтобы $|a|{\cdot}|b| \le 2^{k-1}$.

6.4.6. Операция целочисленного деления

Целочисленное деление с остатком в двоичной системе сводится к сравнению и вычитанию, и если как делимое, так и делитель представимы в k-разрядном типе, то и результат деления, и остаток от него будут получены правильно.

Однако если в делении участвуют отрицательные числа, то остаток компьютерного деления может не совпадать с математическим понятием остатка, и об этом следует помнить при программировании.

Пример 6.7. Операции mod и div в Turbo Pascal над отрицательными числами дадут следующие результаты:

```
(-5) mod   2 =-1;
(-5) div   2 =-2;
  5  mod (-2)= 1;
  5  div (-2)=-2.
```

6.5. Битовые операции и их применение (задачи с решениями)

Наряду с арифметическими операциями над k-разрядными целыми числами в компьютере определены также битовые операции.

1. NOT (в Turbo Pascal) или ~ (в C) — ***инверсия*** всех битов, отведенных под целое число, то есть все нули в представлении числа заменяются на единицы, а единицы на нули. Данная операция является ***унарной***, т.е. применяется к одному операнду (числу), например, в знаковых типах NOT(3) = – 4, и не зависит от количества разрядов для представления чисел в этом типе.

2. AND, OR, XOR (в Turbo Pascal) или **&, |, ^** (в C) — ***побитовые логические операции* "и", "или"** и **"исключающее или"**, последняя из которых означает сложение по модулю 2 или "не равно". Эти операции ***бинарны*** и производятся над каждой из пар битов операндов. Результаты операций для различных значений битов приведены в таблицах:

A	b	a and b	a or b	a xor b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Пример 6.8.

5 and 3 = 101_2 and 11_2 = 1_2 = 1;
5 or 3 = 101_2 or 11_2 = 111_2 = 7;
5 xor 3 = 101_2 xor 11_2 = 110_2 = 6.

3. Следующая группа операций уже упоминалась выше, она соответствует ***сдвигам числа в ячейке***:

сдвиг влево: **I SHL J** (в Turbo Pascal) или I << J (в C): сдвиг числа I на J бит влево, причем освободившиеся правые J бит заполняются нулями, что соответствует умножению на 2^J;

сдвиг вправо: **I SHR J** (в Turbo Pascal), или I >> J (в C): сдвиг числа I на J бит вправо, причем освободившиеся левые J бит заполняются нулями, что соответствует целочисленному делению на 2^J.

Покажем, как перечисленные операции можно применять при решении различных задач.

Задачи:

1. По введенному целому неотрицательному числу N получить значение 2^N (операция возведения в степень в языке программирования Turbo Pascal отсутствует, а в C реализована для произвольных чисел и в данном случае будет выполняться крайне неэффективно).

Решение для 0 ≤ N < 31 на языке Turbo Pascal

```
Var
  n,p:longint;
Begin
  write('Введите значение степени 2-x:');
  readln(n);
  p:=1 shl n;
  writeln('2 в степени ',n,'=',p);
  readln
End.
```

Решение для 0 ≤ N < 32 на языке C

```
#include <stdio.h>
void main()
{unsigned long p;
 int n;
 printf("Введите значение степени 2-x:");
 scanf("%d",&n);
 p=(unsigned long)1 << n;
 printf("2 в степени %d=%lu\n",n,p);
}
```

2. Перевести натуральное число, не превосходящее 10^9, в двоичную систему счисления.

Решение использует тот факт, что любое натуральное число хранится в компьютере в своем двоичном представлении. Поэтому достаточно узнать и напечатать значения всех его бит слева направо, опуская незначащие левые нули. Будем считать, что все биты в представлении двоичного числа n пронумерованы справа налево, начиная с нуля.

Идея заключается в выделении i-го бита с помощью логического умножения на двоичное число, содержащее единицу в этом самом разряде и нули — во всех остальных. Таким числом является 1 shl i, а соответствующая операция — (1 shl i) and n.

Решение на языке Turbo Pascal

```
Var
  i,k,n:longint;
Begin
  write('Введите десятичное число:');
  readln(n);
  write(n,'=');
  k:=30;{максимальный разряд, в котором может быть 1}
  while (k>0) and (n and (1 shl k)=0) do
    k:=k-1;{пропускаем незначащие ведущие нули}
  for i:=k downto 0 do {печатаем значащие цифры}
    if n and (1 shl i)=0 then write(0)
                         else write(1);
  writeln('(2)');
  readln
End.
```

Решение на языке C (обратите внимание на расстановку скобок в логических выражениях):

```
#include <stdio.h>
void main()
{unsigned long n;
 int i,k=30;
 printf("Введите десятичное число:");
 scanf("%D",&n);
 printf("%ld=",n);
 while ((k>0)&&((n & ((unsigned long)1 << k))==0))
   k--;/* пропускаем незначащие ведущие нули */
 for (i=k;i>=0;i--) /* печатаем значащие цифры */
  if ((n & ((unsigned long)1 << i))==0) printf("0");
  else printf("1");
 printf("(2)\n");
}
```

3. Подсчитать число единиц, входящих в двоичное представление данного натурального числа, не превосходящего 10^9, за число действий, пропорциональное числу этих единиц.

Решение этой задачи основано на следующем факте: если из положительного двоичного числа m вычесть единицу, то содержимое младших разрядов этого числа (до разряда с младшей единицей включительно) меняется на противоположное (цифра 0 заменяется на 1, а 1 — на 0), а старшие разряды остаются без изменения. Если, например, m = 1001 1000, то $r = m - 1$ = 1001 0111.

Применив побитовую операцию r and m, удалим младшую единицу из исходного числа. Так, в нашем примере r and m=1001 0000. Присвоим это новое значение переменной m и будем выполнять указанную процедуру до тех пор, пока очередная операция and не приведет к нулевому значению. Легко понять, что количество повторений этой процедуры и есть искомый результат.

Решение на языке Turbo Pascal

```
Var
  m:longint;{исходное число}
  n:byte;{результирующая переменная}
Begin
  write('Введите десятичное число:');
  readln(m);
  n:=0;
  while m<>0 do
    begin
      n:=n+1;
      m:=m and (m-1)
    end;
  writeln('В двоичном представлении исходного
числа ',  n:4,' единиц');
  readln
End.
```

Решение на языке C

```
#include <stdio.h>
void main()
{unsigned long m;/* исходное число */
 int n=0; /* результирующая переменная */
 printf("Введите десятичное число:");
 scanf("%D",&m);
 while (m!=0)
   {n++;
    m&=m-1;
   }
 printf("%d единицы в двоичном представлении
        исходного числа \n",n); }
```


4. Сгенерировать все подмножества исходного множества, состоящего из n (не более чем 30) элементов произвольного типа, исключая пустое множество.

Казалось бы, какое отношение имеет эта задача к рассматриваемым в данной главе темам, однако любое подмножество можно задать, поставив в соответствие каждому элементу исходного множества 0 (элемент не принадлежит подмножеству) или 1 (элемент подмножеству принадлежит).

То есть, каждому подмножеству соответствует набор из n цифр, каждая из которых принимает значение 0 или 1. Нетрудно понять, что полный перебор всех подмножеств данного множества соответствует перебору всех чисел в двоичной системе счисления от $\underbrace{0...01}_{n-1}$ до $\underbrace{1...1}_{n}$.

Последнее из таких чисел соответствует всему исходному множеству.

Теперь легко подсчитать и количество различных подмножеств данного множества. Оно равно $2^n - 1$ (или 2^n, с учетом пустого множества).

Прежде чем перейти к рассмотрению программ, соответствующих данному способу перебора, укажем, когда их применение является целесообразным. Во-первых, их удобно использовать, когда необходимо в любом случае перебрать все подмножества данного множества (например, требуется найти все решения, удовлетворяющие тому или иному условию), и во-вторых, когда с точки зрения условия задачи не имеет значения, сколько именно элементов должно входить в искомое подмножество.

Если же требуется найти подмножество, содержащее минимальное количество элементов и удовлетворяющее тому или иному условию, то порядок перебора должен быть другой: сначала генерируются все подмножества, состоящие из одного элемента, потом из двух и т.д.

Решение на языке Turbo Pascal

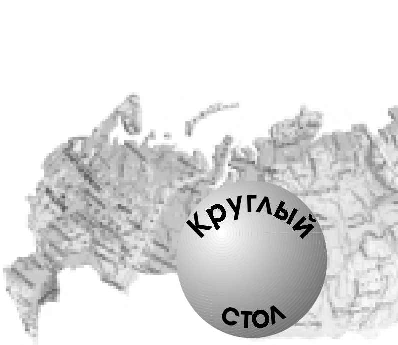
```
Const n:longint=10;{число может быть любым}
Var
  q,j,k:longint;
  a:array[1..]of real;
 {массив a содержит исходное множество,
  тип его элементов может быть любым}
Begin
  for j:=1 to n do
    a[j]:= j;{массив заполняем, например,
               натуральными числами }
  q:=1 shl n;{помещаем в q число 2^n}
  for j:=1 to q-1 do
             {цикл по всем подмножествам}
    begin
      for k:=1 to n do
      {выделение единиц в числе j и обработка
       соответствующего подмножества}
       if (1 shl(k-1))and j<>0 then write(a[k]:8:2);
      writeln
     end
End.
```

Решение на языке C

```
#include <stdio.h>
void main()
{unsigned long q,j,k,n;
   /* количество элементов n во множестве
      может быть и другим */
 float a[30];
```

Программно-методический комплект. Прошлое и будущее

А.И. Сенокосов



Осенью прошлого года на конференции “ИТО-98” наша газета провела “круглый стол” “Состав учебно-методического комплекта по информатике” (см. № 44/98). По итогам этого “круглого стола” некоторые его участники подготовили статьи, которые мы начинаем публиковать с этого номера. В прошлом номере мы опубликовали статью Ю.А. Шафрина “Ребята, давайте жить дружно!”. Сегодня слово другому участнику “круглого стола”, автору учебников и постоянному автору нашей газеты — А.И. Сенокосову. Статьи, подготовленные другими участниками “круглого стола”, будут опубликованы в осенних номерах газеты.

В стародавние времена, когда информатика только начинала свое житье-бытье в школьных стенах, вопрос о том, каким быть программно-методическому комплекту для преподавания информатики, имел вполне определенный ответ. Отцы-основатели допускали изучение нового предмета в так называемом “безмашинном” варианте, что ставило его в ряд физики и математики.

Естественно, для такого варианта обучения требовались лишь хороший учебник и отлаженная методика преподавания, материализованная в виде книги для учителя.

Появление в школах компьютеров добавило изрядной головной боли представителям всех уровней народного образования — от сотрудников министерства до учителей. Ситуация усугубилась тем, что оснащением школ компьютерами занимались все, кому не лень, в том числе и отечественные производители, которые меньше всего думали об унификации техники.

В итоге только “Корветов” насчитывалось 4 (!) различные модификации, каждая из которых с точки зрения аппаратной части была “вещью в себе”. А тут подоспели УКНЦ, “Ямахи”, “Поиски”, “Электроники”, ЕС, IBM PS/1...

Учителя оказались “законсервированными” в своих учебных заведениях, ибо о чем можно разговаривать с коллегой, у которого совершенно другая техника, не имеющая ничего общего с твоей?! Он, скажем, может позволить себе преподавать графический редактор, зато у него нет электронных таблиц. Правда, сеть у него работает безукоризненно, поэтому можно рассчитывать на разработку более или менее длинных программ на встроенном Бейсике...

Добавим сюда постоянные трудно решаемые проблемы с ремонтом халтурно состряпанной “вычислительной техники” и попробуем на этом фоне представить героический труд первых учителей информатики, который, впрочем, по большей части таковым и остался.

В не менее тяжелом положении оказались и авторы учебников, которые были вынуждены создавать свои курсы с оглядкой на этот разномастный “зоопарк”. Практически за каждым учебником стоял штат программистов, разрабатывающий на различных типах машин “учебные текстовые редакторы”, “учебные базы данных”, “учебные электронные таблицы”, поскольку никакого промышленного программного обеспечения такого сорта для большинства видов компьютеров попросту не существовало.

Понятно, что места, где существовали условия для создания учебников, были довольно далеки от сферы школьного образования, что и привело к изрядной оторванности программно-методических комплектов от реалий школьного преподавания.

Наследием этих времен и стало представление о программно-методическом комплекте как о самодостаточной системе, включающей в себя учебник, книгу для учителя и абсолютно все программные средства, необходимые при работе в кабинете информатики.

Времена, как им и положено, меняются. Все больше школ имеют в своих кабинетах исключительно “100%-IBM-совместимую” технику (допотопное, конечно, название, не очень-то отражающее действительность).

Те же школы, в которых находятся кабинеты с устаревшей техникой, просто пали жертвой... нет, отнюдь не экономических трудностей, которые хронически переживает страна. О каких трудностях можно вести речь, если стоимость пары вполне надежных и работоспособных компьютеров на базе 286-х процессоров в начале 1998 года была сравнима с месячной зарплатой одного учителя? Да, разумеется, есть и неблагополучные регионы, где и зарплату-то не платят... потому что администрация района закупила акции нефтяной компании! (Совершенно реальный случай!)

Так что основной причиной наличия старых компьютеров в школах можно считать лишь абсолютно безразличное, можно сказать, наплевательское отношение руководства народного образования на самых разных уровнях к вопросу компьютеризации школ. Оно и понятно. Гораздо выигрышнее смотрятся презентации, посвященные открытию суперсовременных Интернет-центров, чем покупка во все школы региона устаревших марок компьютеров, вполне подходящих, впрочем, для учебного процесса.

Вернемся к вопросу о ПМК. Следствием массового оснащения школ более или менее современной техникой явился другой взгляд на школьную информатику, рассматривающий ее лишь как узкотехнологическую дисциплину, призванную возместить нехватку в обществе “людей, знакомых с ПК”.

Впрочем, подобные сиюминутные веяния постоянно прокатывались по российской школе, начиная от массового изучения электротехники до организации особого, “политехнического” типа средних учебных заведений.

И если “безмашинный” вариант предусматривал стандартный набор “учебник — методичка” по причине отсутствия компьютеров, то изучение Windows и Microsoft Office не требует даже и “методички”, поскольку сам по себе учебник содержит подробное описание всех несложных технологических действий.

Правда, в силу достаточно высокого “интеллектуального” уровня самого технологического оборудования при таком подходе в перспективе отпадает нужда и в самом учебнике, чего не могло быть при изучении, скажем, токарного станка, который не мог подсказывать, как на нем работать.

Остается простой и в меру доступный минимум — Windows, Microsoft Office и программа, обучающая работе с ними. (О подобном программном обеспечении говорит, например, А.Ю. Уваров — “Чему и как учить на уроках информатики” — “Информатика”, 1999, № 1.)

Легко понять, что при этом варианте учитель превращается в простого надсмотрщика, следящего за сохранностью вычислительной техники. Ну а если учесть еще и стремительно набирающую обороты компьютеризацию, то становится вполне логичным полное исключение из этого процесса и учителя. А предмет можно исключить из школьного курса, предложив желающим освоить его прямо на дому, без лишних нервных затрат.

Если такое развитие событий вас почему-то не устраивает, а оно, судя по всему, не устраивает и Министерство образования, и ведущих специалистов в области школьной информатики, и свой предмет вы мыслите как предмет, стоящий в ряду остальных школьных предметов, развивающих интеллектуальный мыслительный потенциал школьников, то потребуется несколько более сложный набор, необходимый для работы.

Безусловно, кроме учебника и книги для учителя, необходим учебный исполнитель, служащий средством развития алгоритмического мышления. Называть такого исполнителя “архаичным, устаревшим и совершенно ненужным” может только человек, безумно далекий от школьного образования. Представляется, что дискуссия на эту тему на страницах “Информатики” абсолютно неуместна и может служить лишь показателем крайне низкого уровня профессионализма.

Впрочем, желающие могут найти исчерпывающую аргументацию в пользу такой точки зрения в “12 лекциях...” А.Г. Кушниренко (см.: “Информатика”, 1999, № 1).

Видимо, совсем не лишней будет и модель процессора, дающая возможность наглядно показать основы работы современной вычислительной техники, сформировав у учащихся ясное понимание самой природы компьютера.

Не будем забывать и о том, что общество ожидает от школьного курса информатики еще и так называемой “компьютерной грамотности”. Иначе говоря, любой выпускник школы должен совершенно спокойно ставить галочку в анкете о приеме на работу напротив графы “Знание ПК”. Нужно ли для этого создавать особые, “учебные” текстовые и графические редакторы, базы данных и электронные таблицы?

Один из “китов”, на котором стоит школьный курс информатики, заявленный в тех же “12 лекциях...”, предполагает “настоящесть” курса. С этой точки зрения вряд ли можно убедительно обосновать использование “искусственных” программных средств вместо фирменных локализованных (переведенных на русский язык) продуктов. Тем более что современные программные продукты, сделанные по принципу “дружественного интерфейса”, оснащенные интуитивно понятным инструментарием, с возможностью настройки панелей позволяют начинать работу с ними практически с нуля, плавно усложнять практические работы, добиваясь вполне сносного их освоения даже весьма средними учениками седьмого класса. А опыт показывает, что проблемы, связанные с необъятным количеством лишних, порой никому не нужных функций, которые будто бы пугают детей, во многом чисто умозрительные.

Итак, подводя итог вышесказанному, вполне разумным представляется следующий состав ПМК:

1) учебник;
2) книга для учителя;
3) учебный исполнитель типа “Робот”, “Черепашка” или “Паркетчик”;
4) действующая модель процессора типа “ToyCom” или “Малютки”;
5) стандартные локализованные версии графических, текстовых редакторов, электронных таблиц, разработанные для PC или “Макинтоша”;
6) специально разработанная, ориентированная на школьников база данных.

Ответ молодому педагогу

В апреле в группе новостей **relcom.education** был опубликован очень характерный вопрос. Ответ на который, предполагаю, будет интересен и вам, читатель...

```
>Подскажите, чем заняться
>со школьниками 10-го класса
>при изучении темы
>"Электронные таблицы Excel",
>Оборотные ведомости,
>различные сметы и т.п.
>Они считают, но без энтузиазма.
>Посоветуйте что-нибудь
>интересное...
>kit@iubnt.yar.ru
>для Карташевой Ольги. Спасибо.
```

Уважаемая Ольга! Вы поднимаете очень интересный вопрос, но он скорее не из области информатики, а из области общей педагогики. Решение проблемы — в правильной мотивации. Мотивация максимальна при обработке личностно значимой информации. Приведу несколько примеров из своего опыта.

Таблица 1. “Сладкая жизнь”

Цель образовательная: формирование навыков редактирования электронных таблиц, демонстрация возможностей автоматизации расчетов.

Цель воспитательная: анализ уровня потребностей и возможностей семьи.

1. Предлагается заполнить 5—6 строк в таблице и рассчитать, сколько стоит “твоя сладкая жизнь”. Цены на сладости дети знают прекрасно, их приводить не следует, но надо напомнить, что целесообразно использовать формат поля с двумя знаками после запятой. Формируется таблица следующего вида (она приведена не полностью):

A	B	C	D	E	F
Название продукта	Цена 1 шт. (руб.)	Кол-во в 1 день	Кол-во в месяц	Расходы в день (руб.)	Расходы в месяц (руб.)
Pepsi			=C2*31	=B2*C2	=E2*31
Мороженое					
			ИТОГО	=СУММ(E2:E7)	=СУММ(F2:F7)

2. Предлагается изменить количество сладостей так, чтобы “Расходы в месяц” составляли примерно половину зарплаты мамы или папы.

3. Предлагается изменить количество сладостей так, чтобы “Расходы в месяц” были как можно ближе к среднему заработку, или прожиточному минимуму, или бабушкиной пенсии.

4. Коллективно анализируем “наши” запросы и “наши” возможности.

Таблица 2. “Последний звонок”

Цели примерно те же. Только теперь рассчитываем, по скольку надо сброситься на организацию в классе вечеринки, посвященной последнему звонку.

Вопрос о мотивации вплотную подходит к проблеме организации структуры урока. Наиболее эффективно актуализируют внимание, мыслительную деятельность приемы, связанные с организацией проблемной ситуации и “обреченной на успех” разрешимостью конфликтной ситуации. Приведу пример из опыта работы с учениками 10-х классов.

Заканчивается тема “Текстовые редакторы”, обобщающий урок идет в виде зачета (практическая и теоретическая части). Имея при себе таблицу “Мой сравнительный анализ возможностей двух (трех) текстовых редакторов”, ученики подходят к преподавателю и за 3—4 минуты “переводят” эту таблицу в рассказ о текстовых... Дальше все очень скучно. Все при деле, кто-то выполняет N-ю часть практической работы, кто-то тоскливо рассказывает... Надоело слушать, 3-й урок одно и то же! Взял платок у отвечающей, завязал на свою лысеющую голову и произнес: “Я твоя деревенская тетка Фекла, работаю в сельсовете секретарем у председателя. Докажи мне, что компьютер нужнее печатной машинки. Доказать удастся, будет отлично, давай!” Вот тут и произошло маленькое чудо. “Ну и дура же ты у меня, тетка, набитая дура! Весь мир на компьютерах стучит, а ты все замазкой да копиркой пользуешься! И не лень одну и ту же справку сто раз печатать, сохранила файл, а потом только редактируй. Одну опечатку сделала — и пошла мазать-перемазать замазкою, а на компьютере одну клавишу нажала — и нет ошибки, да и ошибки допустить трудно будет, компьютер же их “покрасит”. А в райцентр мотаться, деньги тратить! Послала электронное письмо — и весь разговор...”. Хотелось прервать ребенка, за магнитофоном сбегать, да народ собрался, слушает, смеемся вместе. Пришлось огрызаться за тетку Феклу, спорить, вся группа аргументы подбрасывала, убедили, пошла Фекла председателя уговаривать компьютер покупать.

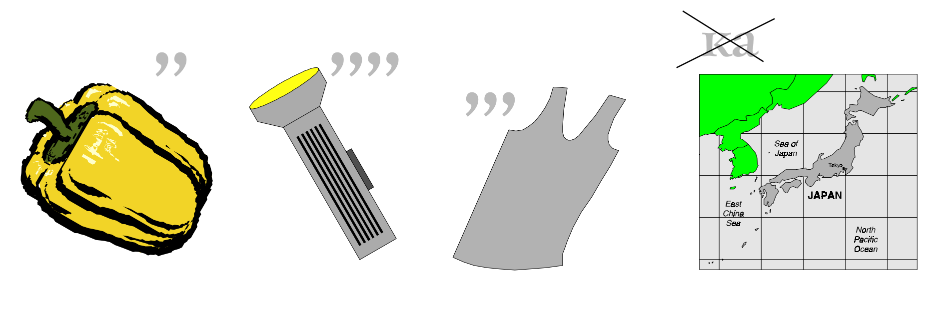
С урока и дети, и я ушли удовлетворенные. Сколько лет прошло, а помню эту историю. Может, просто самим собой с ребенком надо быть? Шутить чаще, играть, быть интересным человеком. И читать надо, например, **www.1september.ru**, там удобно выборку тематическую сделать. Приезжайте к нам в МДКЦ, там именно так с ребятами работают, весело и результативно, а пока загляните на наш сервер **www.botik.ru/ICCC**.

Всего доброго.

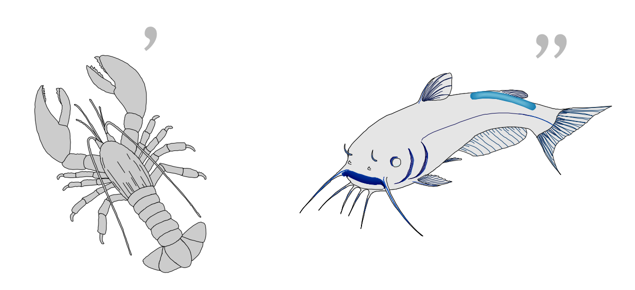
Методист Международного детского компьютерного центра ИПС РАН
СКОРОДУМОВ *Вячеслав Евгеньевич.*
E-mail: skorodum@mail.ru

ПРЕДЛАГАЮ КОЛЛЕГАМ

Ребусы по информатике



Перфокарта



Распечатка

Система

Материалы рубрики подготовлены В.Г. Федориновым

Пушкин и информатика

Окончание. Начало на с. 1

они поступают как истинно формальные исполнители. Чем формальный исполнитель отличается от неформального? Если формального исполнителя, имеющего часы, спрашивают: “Можете ли вы сказать, который час?” — он отвечает: “Могу”. А неформальный в ответ сообщит, какое время показывают его часы.

Так же и тут. Сказано: тайно — значит, надо закрыть так, чтобы никто не видел, т.е. в ящик или бочку. Сказано: бросить в воду — бросили. Убивать не приказано. Хотя, конечно, бояре вполне понимали смысл царского приказа — недаром сказано, что выполнять приказ они пошли, сначала “потужив о государе”.

С другой стороны, приказ “тайно бросить в бездну вод” принадлежит к числу так называемых плохо поставленных задач. Вариант, предложенный в сказке, только один из множества возможных вариантов уточнения постановки задачи. Ведь слову “тайно” можно придать смысл “без огласки”, “секретно”. “Бросить в бездну вод” можно было толковать так, как это сделано в русской сказке о сестрице Аленушке и братце Иванушке, — с камнем на шее. И т.д. Бояре нашли весьма нетривиальное решение этой задачи. Ай да Пушкин!

Вот строки из другой, не менее известной сказки Александра Сергеевича:

Свет мой, зеркальце! скажи
Да всю правду доложи:
Я ль на свете всех милее,
Всех румяней и белее?

Вам это ничего не напоминает? Ну конечно, запрос к информационно-поисковой системе. В запросе три признака, соединенных логической связкой И. А зеркальце — этакий выход в Интернет: ведь поиск надо провести по всему свету. Без Интернета тут не обойдешься. Царевич Елисей тоже ведет поиск с помощью информационных сетей. Только в Интернет у него, по-видимому, доступа нет, поэтому он обращается лишь к некоторым сетям: солнечной, лунной и ветряной.

Конечно, все, здесь рассказанное, — это шутка. Но шутка дидактическая — в устах учителя она способна оживить урок, заставить взглянуть на вещи с иной, иногда весьма неожиданной, стороны. А это так важно в учебном процессе.

И все же перечитайте еще раз сказки Пушкина. В большинстве из них движущей пружиной сюжета является именно информационный аспект. Вся “Сказка о царе Салтане” построена вокруг сначала подслушивающего царя, затем искажения информации завистливыми сестрами, потом на информации, получаемой Гвидоном во время его сказочных полетов к отцу. И как только информационный хаос рассеивается — все всё узнают, — сказка кончается. В “Сказке о мертвой царевне и о семи богатырях” все события тоже развиваются исключительно благодаря потокам информации. Что уж говорить про “Сказку о золотом петушке”, где заглавный персонаж имеет сугубо информационную функцию. Ни в русских народных сказках, ни в авторских сказках, созданных до А.С. Пушкина, вы не найдете такого. Гениальный Пушкин, видимо, ощущал, какая это мощная вещь — информация. И в его жизни именно информация сыграла роковую роль.

Ф. СП-1

Министерство связи
Российской Федерации
“Роспечать”

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

Информатика
(наименование издания) | Количество комплектов

на 1999 год по месяцам

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

Куда (почтовый индекс) (адрес)

Кому (фамилия, инициалы)

ДОСТАВОЧНАЯ КАРТОЧКА

ПВ | место | литер | на газету **32291** (индекс издания)

Информатика
(наименование издания)

Стоимость: подписки ______________ руб.; переадресовки ______________ руб. | Количество комплектов

на 1999 год по месяцам

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

Куда (почтовый индекс) (адрес)

Кому

Кривая дракона: послесловие

Кривая дракона, о которой рассказано в статье Д.М. Златопольского (см. с. 2, 3), впервые была описана в популярной литературе в журнале Scientific American в 1967 году. Заметка о ней появилась в колонке “Математические игры”, которую вел Мартин Гарднер. Первоначально использовалось полное название кривой — “дракон Хартера — Хейтуэя”, которое ей дал основатель компьютерной фрактальной геометрии Бенуа Мандельброт, именем которого названо знаменитое множество. В дальнейшем стали говорить просто о кривой дракона. В статье Д.М. Златопольского описан один из алгоритмов построения кривой. На наш взгляд, он несколько запутан (хотя и достаточно прост в реализации). В этой заметке мы приведем описание алгоритма построение кривой, близкое к тому, которое использовалось Мартином Гарднером.

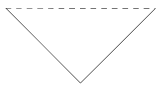


Рис. 1

Рассмотрим горизонтальный отрезок как кривую дракона нулевого порядка. Разделим отрезок пополам и построим на нем прямой угол, как показано на *рис.* 1. Получим кривую дракона первого порядка. На сторонах прямого угла снова построим прямые углы (*рис.* 2). При этом вершина первого угла всегда находится справа, если смотреть из точки А (начала кривой) вдоль первого отрезка кривой, а направления, в которых строятся вершины остальных углов, чередуются. На *рис.* 3 и 4 показаны кривые дракона третьего и четвертого порядков соответственно.

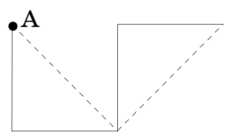


Рис. 2

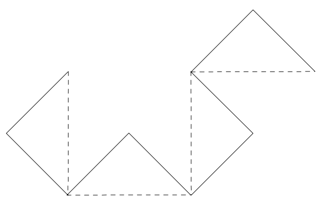


Рис. 3

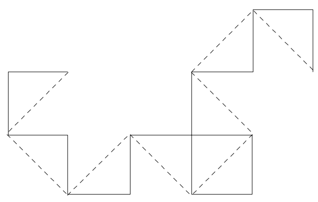


Рис. 4

Кривая дракона просто и изящно описывается в так называемой L-системе (предложенной Линденмауэром). Об этой системе, которую также называют черепашьей, вы можете прочитать в заметке “Елки, палки и прочие фракталы” (см. № 19/98). В L-системе дракон Хартера — Хейтуэя описывается с помощью одной “аксиомы” `FX` и двух “теорем”: `+FX--FY+` и `-FX++FY-`. Угол поворота (который, напомним, используется в командах `+` и `-`) равен 45°.

Кривая дракона — один из популярнейших объектов, который встречается практически во всех курсах фрактальной геометрии. Описание алгоритма построения кривой и множество другой интересной информации (в частности, ссылок на другие источники) находится, например, на **http://astsun.astro.virginia.edu/~eww6n/math/DragonCurve.html**. Краткое, но очень понятное описание можно найти на **http://www.math.okstate.edu/mathdept/dynamics/dragon.html**. Имеется также множество русскоязычных источников, например, **http://home.ural.ru/~shabun/fractals/fractals.htm**.

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

121165, Москва,
Киевская, 24
тел. 249 4896
Отдел рекламы
тел. 249 9870

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

Отпечатано с готовых диапозитивов редакции в ОАО ПО «Пресса-1», 125865, ГСП, Москва, ул. Правды, 24.

Тираж 7000 экз.
Заказ №

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

Тел.: (095) 249 3138, 249 3386; факс (095) 249 3184