Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

Е. В. Касьянова, С. Н. Касьянова

ПРОГРАММИРОВАНИЕ ДЛЯ ШКОЛЬНИКОВ: СБОРНИК ЗАДАЧ ПОВЫШЕННОЙ СЛОЖНОСТИ С РЕШЕНИЯМИ

Препринт
95

Новосибирск 2002

Препринт содержит задачи повышенной сложности по программированию для школьников. Все задачи приводятся с решениями на языке Паскаль. Цель – научить школьников основным методам конструирования нетривиальных программ.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

E. V. Kasianova, S. N. Kasianova

PROGRAMMING FOR PUPILS: A COLLECTION OF RELATIVELY COMPLEX PROBLEMS WITH SOLUTIONS

Preprint
95

Novosibirsk 2002

The preprint contains relatively complex problems of programming for pupils. All problems are given with solutions in Pascal. The aim is to coach the pupils using the basic methods for design of nontrivial programs.

ПРЕДИСЛОВИЕ

Задачник базируется на опыте преподавания курса программирования в старших классах с математическим уклоном. В данном курсе большое внимание уделяется изучению и составлению алгоритмов. Преподавание алгоритмизации учит гибкости мышления и закладывает важную базу для дальнейшего изучения программирования. В технологии преподавания, в основном, используется проблемный ("задачный") подход в обучении. Почему выбран этот подход? Учитывается психология одаренных детей. Творческие дети относятся к составлению алгоритмов при решении задач, как к разгадыванию кроссвордов. Такие дети не любят решать однотипные задачи, поэтому очень важен подбор заданий. Им нужны интересные задачи, при решении которых они приобретают новые знания, знакомятся с новыми методами и новыми структурами данных. Набор задач должен, с одной стороны, предоставлять возможность ребенку проявлять смекалку, с другой стороны, требовать определенных знаний методов решения задач.

В данном сборнике представлены задачи, которые мы разбираем и решаем с детьми. При решении задач используется язык Паскаль. Для интересующихся школьников рекомендуем дополнительную литературу [1—16].

1. КРАТЧАЙШИЙ ПУТЬ В ЛАБИРИНТЕ

Задача. Задан лабиринт из n комнат, у каждой из которых имеется не менее одной и не более трех дверей, соединяющих между собой различные комнаты. Одна из комнат — вход, другая — выход. Найти кратчайший путь от входа к выходу.

```
Program labirint;

{При поиске используется стандартный алгоритм волны на графе. Вершинами графа являются комнаты лабиринта, ребрами — двери.}

uses crt;
const n=9;
type sm=array[1..n,1..n] of integer;
     wave=array[1..n] of integer;
     way=array[1..n] of 0..n;
var w:wave;                    {Для каждой вершины — шаг волны.}
    r:way;                                                {Путь.}
    m:sm;                                  {Матрица смежности.}
    i,j,l,step:integer;

Procedure setlab(var m1:sm);
{----------------------------Ввод лабиринта.-------------------------------------}
var y,c:integer;
begin
  for y:=1 to n do for c:=1 to n do m1[c,y]:=0;
  for y:=1 to n-1 do
    begin
      write ('С какими комнатами связана комната',
            y,'(0-конец):');
      repeat
         read(c);
         if (c<=n) and (c>y) then
            begin m1[y,c]:=1; m1[c,y]:=1 end;
      until c=0
    end
end;
```

```
begin
  clrscr; setlab(m);
{------------------------------------ Обнуление. ------------------------------------
                                    --}
  for i:=1 to n do w[i]:=0;
  for i:=1 to n do r[i]:=0;
{-----------------------------------Алгоритм волны-----------------------------------
                                    --}
  w[1]:=1;                                {Начало волны в комнате 1.}
  for step:=1 to n do
    for i:=1 to n do
      if w[i]=step then
        for j:=1 to n do
          if m[i,j]=1 then
            if w[j]=0 then w[j]:=step+1;
{-------------------------------------Поиск пути-------------------------------------
                                    --}
  if w[n]=0 then
     write ('нет пути')          {Волна не дошла до комнаты n.}
  else begin
     l:=w[n];                                          {Длина пути.}
     r[l]:=n;                                  {Последний шаг пути.}
     for i:=l-1 downto 1 do
       for j:=1 to n do                      {Ищем откуда пришли.}
         if (w[j]=i) and (m[r[i+1],j]=1) then r[i]:=j;
     write ('путь:');
     for i:=1 to l-1 do write (r[i],',');
     write(r[l],'.');
  end;
  repeat until keypressed;
end.
```

2. СРАВНЕНИЕ СТРОК

Задача. Вводятся две строки символов в кодировке ASCII. Символы находятся в диапазоне [32..126]. Требуется определить, какие символы встречаются в обеих строчках, а какие только в одной, и вывести их в порядке возрастания номера. Проверки корректности входных данных не требуется.

```
Program comp_str;
```

{Алгоритм будет красивым при правильном выборе структур данных.}

```
var st1,st2:string;
    symbols:array[32..126]of byte;
    a:integer;
begin
   writeln(' Сравнение строк');
   writeln;
   writeln('Введите строки:');
   readln(st1);
   readln(st2);
   fillchar(symbols, sizeof(symbols),0);
   for a:=1 to length(st1)do
     symbols[ord(st1[a])]:=symbols[ord(st1[a])]or $F;
   for a:=1 to length(st2)do
     symbols[ord(st2[a])]:=symbols[ord(st2[a])]or $F0;
   write('В обеих:');
   for a:=32 to 126 do
     if symbols[a]=$FF then write(char(a));
   writeln;
   write('В одной:');
   for a:=32 to 126 do
      if symbols[a]in[1..$FE] then write(char(a));
   writeln;
end.
```

3. РАССТОЯНИЕ МЕЖДУ ПАРАМИ ВЕРШИН В ГРАФЕ

Задача. Задан ориентированный граф, у каждой дуги своя длина. Найти кратчайший путь от заданной вершины до другой заданной вершины.

```
Program graf;
```

{Используем матричный метод решения задачи о кратчайшем пути. Граф представляется в виде матрицы S[n, n], где n — количество вершин. Элемент матрицы S[i, j] равен длине дуги из i-й в j-ю вершину, если такая дуга существует, иначе равен бесконечности. Все элементы на главной диагонали равны 0. Определяем следующее “умножение” матриц: V = S * T; v[i, j] = min(s[i, k] + t[k, j]), k = 1, ..., n. Кратчайший путь между вершинами i, j в графе из n вершин может состоять не более, чем из n – 1 дуг. Элемен-

ты матрицы S в степени (n – 1) содержат все минимальные пути графа. Вместо вычисления степеней матрицы путем последовательного “умножения” на S, более экономично последовательно определить степени S кратные 2, т.е. S, S^2, S^4, S^8, … . Длины всех кратчайших путей определятся на r-м шаге, когда 2^r >= n – 1.}

```
uses crt;
const nn=20;                          {Количество вершин в графе.}
      inf=100000;                                 {Бесконечность.}
type node= array[1..nn,1..nn] of longint;
var n,a1,a2:integer;
    nodes,s:node;
    ch:char;

function Exp2(p:integer):longint;
{---------------------------------Вычисление степеней.-----------------------------}
var t,i:longint;
begin
  t:=1;
  for i:=1 to p do t:=t*2;
  exp2:=t;
end;

procedure Mult(var res,s1,s2:node);
{---------------------------------“Умножение” матриц.-----------------------------}
var i,j,k:integer;
    min,buf:longint;
begin
  for i:=1 to n do
     for j:=1 to n do
        begin
          min:=s1[i,1]+s2[1,j];
          for k:=2 to n do
            begin
              buf:= s1[i,k]+s2[k,j];
              if (buf<min) then
                 min:=buf;
            end;
          res[i,j]:=min;
        end;
end;

procedure Init;
```

```
{-----------------------------Обнуление матрицы графа.-----------------------------}
var i,j:integer;
begin
  clrscr;
  write('Введите число вершин:');
  readln(n);
  for i:=1 to n do
     for j:=1 to n do
        nodes[i,j]:=0;
end;

procedure GetNodes;
{-----------------------------Заполнение матрицы (весов) графа.-------------------
-}
var i,j:integer;
    ff:longint;
begin
  clrscr;
  writeln('Введите длины дуг между вершинами.',
          'Если дуга отсутствует, введите 0');
  for i:=1 to n do
    for j:=1 to n do
      if(i<>j) then
        begin
           write('Длина пути из ',i,
                 'вершины в ',j,':');
           readln(ff);
           if(ff=0) then
              nodes[i,j]:=inf
           else nodes[i,j]:=ff;
        end;
end;

procedure FindPath;
{---------------------------Получение матрицы в n – 1 степени.---------------------
}
var r:integer;
    a:node;
begin
  r:=1;
  s:=nodes;
  repeat
```

```
    inc(r);
    mult(a,s,s);
    s:=a;
  until (exp2(r)>n-1);
end;
begin
  repeat
    init;
    GetNodes;
    FindPath;
    repeat
      clrscr;
      writeln('Введите номера вершин, ',
        'между которыми хотите узнать расстояние:');
      readln(a1);
      readln(a2);
      writeln('Расстояние равно:',s[a1,a2]);
      writeln('Нажмите esc для ввода другой матрицы');
    until(ord(readkey)=27);
    writeln('Нажмите esc для выхода');
  until(ord(readkey)=27)
end.
```

4. РЕКУРСИВНОЕ ОБРАЩЕНИЕ ПОДСТРОК

Задача. Задан текст — последовательность литер. Распечатать его, обращая каждую группу из 5 следующих друг за другом литер, т.е. печатая литеры этой группы в обратном порядке. Последнюю группу обращать, даже если она содержит менее 5 литер. В программе не должно быть ни одного оператора цикла.

```
program symbols;

{----------------------Используется рекурсивный метод.----------------------}

Uses Crt;
var n:byte;

procedure p5;
var a:char;
```

```
begin
     n:=n+1;
     if not eoln then
        begin
             read(a);
             if n<5 then p5;
             write(a);
        end;
     n:=0;
end;

procedure p1;
begin
     if not eoln then p5;
     if not eoln then p1;
end;

begin
     clrscr;
     writeln('введите последовательность литер.');
     p1;
     repeat until keypressed;
end.
```

5. РЕШЕНИЕ РЕБУСА

Задача. Дано равенство:

ТРАССА + ТРАССА = КОСМОС.

Здесь одинаковым буквам соответствуют одинаковые цифры, разным буквам – разные цифры. Расшифровать числа.

```
Program code;
```

{Для сокращения перебора проанализируем правила сложения столбиком. Это позволяет получить следующие соотношения: Т = 1, 2, 3, 4; А = 5, 6, 7, 8, 9; С= 2, 4; Р = С + 5; О = 2С + 1; М = 2С; К = 2Т + 1.}

```
var c,a,t,p,o,m,k,x,y,x1,y1,i,j:integer;
    u:boolean;
    r:array[1..7] of integer;
```

```
begin
  c:=2;
  while c<=4 do
  begin
    p:=c+5; o:=2*c+1; m:=2*c;
    r[1]:=c;  r[4]:=p; r[5]:=o; r[6]:=m;
    t:=1;
    while t<=4 do
    begin
      k:=2*t+1; r[3]:=t; r[7]:=k;
      a:=5;
      while a<= 9 do
        begin
          r[2]:=a; i:=1; u:=true;
          while (i<=6) and u do
            begin
              j:=i+1;
              while (j<=7) and u do
                if r[i]=r[j] then
                   u:=false
                else j:=j+1;
              i:=i+1
            end;
          if u then
            begin x:=100*t+10*p+a;
              x1:=110*c+a;
              y:=100*k+10*o+c;
              y1:=100*m+10*o+c;
              if(y=2*x) and (y1=2*x1) then
                begin
                  writeln (' ',x,x1); writeln ('+');
                  writeln (' ',x,x1);
                  writeln (' --- ');
                  writeln (' ',y,y1); writeln
                end
            end;
            a:=a+1
          end;
          t:=t+1
        end;
      c:=c+2
   end
end.
```

6. ЧЛЕН ДВОИЧНОЙ ПОСЛЕДОВАТЕЛЬНОСТИ

Задача. Последовательность 1001011001101001... строится так: сначала пишется 1, затем повторяется такое действие: уже написанную часть приписывают справа с заменой элемента 0 на 1 и наоборот, т.е. 1 -> 10 -> 1001 -> 10010110 -> ... . Составить программу, вычисляющую n-й член этой последовательности по заданному n.

```
Program member;
```

{При каждом действии длина последовательности увеличивается в 2 раза. Найдем такое i, для которого верны неравенства 2^i < n<= 2^(i + 1). Предположим, что на n-м месте стоит 1. Она получается по формуле к = 1 – к из 0, который стоит на n – 2^i месте. Далее можно получить, с какого места образуется этот 0. Каждый раз номер будет уменьшаться не менее чем в 2 раза.

Этот процесс будем продолжать, пока n не станет равно 1 или 2. Далее проверим равенство a[n] = k, где к равно числу, в которое преобразуется единица в указанных выше действиях, a[1] = 1, a[2] = 0 — первые два элемента последовательности. Если равенство выполняется, то наше предположение верно и на n-м месте стоит 1, а иначе там стоит 0.}

```
uses
 Crt ;

type
  ZerroOne = 0..1 ;                    {Тип ноликов и единичек.}

var
  n : LongInt ;                          {Вводимое число.}
  i : LongInt;                                {Счетчик.}
  O : Boolean;                          {Условие нолика.}
  Quit : Boolean;                       {Условие выхода.}
  IO : ZerroOne ;                        {Либо 0, либо 1.}

begin
 ClrScr;
 WriteLn( 'Enter n' ) ;
 ReadLn( n ) ;
```

```
 {--------------------------Условие выхода не выполняется.-------------------------
}
 Quit:= False ;
{-------------------------Делаем предположение, что символ — 1.------------------
}
 IO:= 1 ;
{--------------------------Отдельно рассмотрим n = 1.-------------------------------
-}
 if n = 1 then Write( '1' ) else
 begin
    n:= n — 1 ;
{----------------------Ищем ближайшую снизу степень двойки.----------------}
    repeat
     i:= 1;
     repeat
       i:= i * 2;
     until n/i < 1;
{--------------------------Смотрим, когда n либо 1, либо 2.----------------------}
     if (i = 2) and (Quit = False) then
      Begin
        Quit:= True;
        if n = 1 then
          begin
            if IO = 0 then O:= False else O:= True;
          end;
        if n = 2 then
          begin
            if IO = 1 then O:= True else O:= False;
          end;
      end
    else
{Если n отлично от 1 и 2, ищем последовательность на предыдущем шаге.}
     begin
      i:= i div 2; n:= n — i;
      if IO = 0 then
        IO:= 1
      else IO:= 0;
     end;
   until Quit;
   if O then
     Write( '0' )
   else Write( '1' );
```

```
 end;
 repeat until keypressed;
end.
```

7. ОПРЕДЕЛЕНИЕ СИСТЕМЫ СЧИСЛЕНИЯ

Задача. Какое наименьшее количество российских купюр необходимо, чтобы разменять банкноту "осьминога" планеты "Не знаю дробей" достоинством 1000 "вокадусов" из расчета 1 "вокадус" = 1 рубль? Банкнота "осьминога" имеет форму прямоугольника с периметром 101 "ого" и площадью 3003 "ого" в квадрате. В настоящее время в России в обращении находятся купюры достоинством в 100000, 50000, 10000, 5000,1000, 500, 200, 100, 50, 20, 10, 5, 2 и 1 рублей.

```
Program octopus;
```

{Вся сложность задачи в том, что на планете другая система счисления (не десятичная). Вначале определяем основание системы счисления, а затем необходимое количество купюр.}

```
uses crt;

const s: array[1..14] of
      longint=(100000,50000,10000,5000,1000,500,200,
               100,50,20,10,5,2,1);

var   a,b,n,k:longint;

begin
  clrscr;
  n:=5;
  repeat
    a:=(n*n+1) div 2;
    b:=3*(n*n*n+1);
    b:=sqr(a)-4*b;
    if b>=0 then
    begin
      k:=trunc(sqrt(b));
      if sqr(k)=b then
         begin
           b:=(a+k) div 2;
```

```
            a:=(a-b);
            writeln('a=',a,'b=',b);
            writeln('n=',n);
            writeln(n,'*','+1=',n*n+1);
            writeln('2*(a+b)=',2*(a+b));
            writeln('3*',n,'*',n,'*',n,
                    '+3=',3*(n*n*n+1));
            writeln('a*b   =',a*b);
            break;
          end;
    end;
    inc(n,2);
  until n<0;
  k:=0;
  a:=n*n*n;
  writeln('Banknota 1000 =',a:12,'rub');
  for n:=1 to 13 do
    begin
      b:=a div s[n];
      inc(k,b);
      if b>0 then
      writeln(s[n]:7,'rub*',b,'=',s[n]*b:12,'rub');
      a:=a mod s[n];
    end;
  inc(k,a);
  if a>0 then writeln(1:7,'rub*',a,'=',a:12,'rub');
  writeln('all=',k);
  repeat until keypressed;
end.
```

8. СУММИРОВАНИЕ БОЛЬШИХ ЧИСЕЛ

Задача. Даны два очень больших (количество знаков < 200) натуральных числа. Найти и распечатать их сумму.

```
Program sum;
```

{Задача сводится к реализации обычного правила сложения столбиком, если числа представить в виде строк.}

```
uses crt;
```

```
var a,b,c:string[200];
    i,p,p1,p2,q:integer;

begin
  writeln('Введите первое число');
  readln(a);
  writeln('Введите второе число');
  readln(b);
  if length(a) < length(b) then
    begin c:=a; a:=b; b:=c; end;
  p:=0;
  c:='';
  for i:=length(b) downto 1 do
    begin
     p1:=ord(a[i])-ord('0');
     p2:=ord(b[i])-ord('0');
     p:=p+p1+p2;
     q:=p mod 10;
     p:=p div 10;
     c:= chr(q+ord('0'))+c
    end;
  for i:=length(a)-length(b) downto 1 do
    begin
     p1:=ord(a[i])-ord('0');
     p2:=ord(b[i])-ord('0');
     p:=p+p1;
     q:=p mod 10;
     p:=p div 10;
     c:= chr(q+ord('0'))+c
    end;
  if p>0 then c:='1'+c;
  writeln(' ',a);
  writeln('+');
  write(' ');
  for i:=1 to length(a)-length(b) do
  write(' ');
  writeln(b);
  if length(a)=length(c) then
     begin
       write(' ');
       for i:=1 to length(c) do write('-');
       writeln;
       write(' ');
```

```
       writeln(c);
     end
  else
    begin
      for i:=1 to length(c) do write('-');
      writeln;
      writeln(c);
    end;
  repeat until keypressed;
end.
```

9. ВЫКЛАДЫВАНИЕ РАЗНОЦВЕТНЫХ КРУЖКОВ В РЯД

Задача. Написать алгоритм, который находит все возможные расстановки одного белого, одного синего и четырех черных кружков в ряд.

```
Program regroup;
```

{Разместим четыре черных кружка в ряд. Белый кружок можно поместить на одно из следующих мест: перед первым черным, между первым и вторым, ..., после четвертого. Всего 5 вариантов. Аналогично существует 6 вариантов размещения синего кружка. Таким образом существует 30 (5*6) вариантов.}

```
var a,b:string;
    i,j:integer;
begin
  for i:=1 to 5 do
  begin
    a:='ччччч';
    case i of
      1:     a:='б'+a;
      2,3,4: a:=copy(a,1,i-1)+'б'+copy(a,i,5-i);
      5:     a:=a+'б';
    end;
    for j:=1 to 6 do
    begin
      case j of
        1:       b:='с'+a;
        2,3,4,5: b:=copy(a,1,j-1)+'с'+copy(a,j,6-j);
        6:       b:=a+'с';
```

```
      end;
    write(b,' ');
    end
  end
end.
```

10. УРАВНОВЕШИВАНИЕ ГРУЗА

Задача. На левой чашке весов лежит груз в n граммов, где n — натуральное число. Имеется по одной гире в 1, 3, 9, 27, 81, ... граммов. Указать, какие гири и на какую чашку весов надо поставить, чтобы уравновесить груз.

```
Program scales;
```

{Пусть n = 3 * l + k, где k = 0, 1 или 2. Если k = 0, то гиря в один грамм не нужна и повторяем сначала алгоритм для n = l. Если k = 1, то ставим гирю в один грамм на правую чашку весов и повторяем алгоритм для n = l. Если k = 2, то ставим гирю в один грамм на левую чашку весов и повторяем алгоритм для n = l + 1. Алгоритм повторяем до тех пор, пока n не станет равным нулю, при этом при каждом повторении алгоритма рассматривается следующая гиря.}

```
uses crt;

var n,l,m:integer;

begin
  write ('Введите вес груза  ');
  readln(n);
  m:=1;
  while n>0 do
  begin
    l:=n mod 3;
    n:=n div 3;
    case l of
      0: writeln ('Гиря в ',m,' грамм не нужна.');
      1: writeln ('Гирю в ',m,
           ' грамм поставить на правую чашу весов.');
      2: begin
         writeln ('Гирю в ',m,
           ' грамм поставить на левую чашу весов.');
```

```
          n:=n+1
          end
    end;
    m:=m*3
  end;
  repeat until keypressed;
end.
```

11. ЧИСЛА С ТРЕМЯ НУЛЯМИ

Задача. Сколько натуральных чисел, не больше заданного N, имеют в своем двоичном разложении ровно три значащих нуля?

```
{----------------------------------------Метод 1.----------------------------------------
--}

Program binar1;

{Прямое вычисление цифр двоичного разложения и подсчет количества значащих нулей дает программу, которая работает достаточно долго.}

uses crt;

var i,k,l,n,s:integer;

begin
  writeln('Введите N'); readln(n); s:=0;
  for i:=8 to n do
    begin
      k:=i; l:=0;
      while k>0 do
        begin
          if(k mod 2=0) then l:=l+1;
          k:=k div 2;
        end;
      if l=3 then s:=s+1
    end;
  writeln(s);
  repeat until keypressed;
end.
```

```
{----------------------------------------Метод 2.----------------------------------------
--}

Program binar2;

{Разобьем рассматриваемый интервал натуральных чисел на подинтервалы:
левая граница — степень 2, правая — следующая степень 2 – 1. Для каждо-
го из подинтервалов требуемое количество чисел равно числу сочетаний по
три из степени (2). На последнем подинтервале, который может оказаться
урезанным, проводим явное вычисление требуемых чисел.}

uses crt;

var n0,                                   {Нижняя граница интервала.}
    i0,                                    {Степень нижней границы.}
    s                                    {Количество требуемых чисел.}
    i,n,nn,k1,k2,k3,l1,l2,l3:integer;

begin
  writeln('Введите N');
  readln(n);
  s:=0;
  i0:=1;n0:=2;
  while n>=2*n0 do
    begin i0:=i0+1; n0:=n0*2 end;
  for i:=3 to i0-1 do
    s:=s+i*(i-1)*(i-2) div 6;
  nn:=n0+n0 div 8-1;
  if n<16 then
    begin nn:=16; s:=0; end;
  k1:=i0-1; k2:=k1-1; k3:=k2-1;
  l1:=n0 div 2;
  l2:=l1 div 2;
  l3:=l2 div 2;
  if n < 2*n0-8 then
    while (nn<=n) do
      begin s:=s+1;
        if k3<>0 then
         begin
           k3:=k3-1; l3:=l3 div 2;
           nn:=nn+l3;
         end
```

```
        else if k2<>1 then
               begin
                 k2:=k2-1;
                 l2:=l2 div 2;
                 k3:=k2-1;
                 l3:=l2 div 2;
                 nn:=nn+l3+1
               end
             else if k1<>2 then
                    begin k1:=k1-1;
                          l1:=l1 div 2;
                          k2:=k1-1;
                          l2:=l1 div 2;
                          k3:=k2-1;
                          l3:=l2 div 2;
                          nn:=nn+l3+3;
                    end
     end
    else s:=s+i0*(i0-1)*(i0-2)div 6;
  writeln(s);
  repeat until keypressed;
end.
```

12. НАТУРАЛЬНОЕ ЧИСЛО ЗАДАННОГО ВИДА

Задача. В множестве, состоящем из натуральных чисел, в десятичной записи которых встречаются только цифры 1, 3, 7, все элементы занумерованы в порядке возрастания. По заданному N определить N-й элемент этого множества.

```
Program elemset;
```

{Построение необходимых чисел является модификацией способа получения троичного разложения числа.}

```
uses crt;
var a,b:string;
    n:integer;
begin
  writeln('Введите n');
  readln(n);
```

```
  a:='';
  b:='137';
  while n>0 do
    begin
      n:=n-1;
      a:=b[n mod 3+1]+a;
      n:=n div 3
    end;
  writeln('Искомый элемент= ',a);
  repeat until keypressed
end.
```

13. ПРОСТОЕ ЧИСЛО С НАИБОЛЬШИМ ЧИСЛОМ ЕДИНИЦ

Задача. Среди простых чисел, не превосходящих n, найти такое, в двоичной записи которого максимальное число единиц.

```
Program max1;

uses crt;
var n,i,r,max,m:integer;

function pr(a:integer):boolean;
{-------------------------Проверка числа на простоту.-------------------------------
-}
var b,i:integer;
    j:boolean;
begin
  pr:=true;
  j:=true;
  b:=2;
  i:=a div 2+1;
  while b<i do
    begin
      if not j then exit;
      if a mod b = 0 then
        begin pr:=false;j:=false end;
      b:=b+1
    end
end;
```

```
function numb1(a:integer):integer;
{--------------------Подсчет числа единиц в двоичной записи числа.------------}
var e,e1:integer;
begin
  numb1:=0; e:=a; e1:=0;
  while e<>1 do
    begin e:=e div 2;
      if e mod 2 = 1 then
        begin numb1:=e1+1; e1:=e1+1 end
    end
end;

begin
  clrscr;
  writeln('Введите натуральное число');
  readln(n);
  max:=0;
  m:=0;
  r:=0;
  for i:=1 to n do
  if pr(i) then
    begin
      r:=numb1(i);
      if max<r then
        begin
          max:=r;
          m:=i
        end;
     end;
  writeln('Простое число, не больше ',n,',');
  writeln('имеющее наибольшее число ');
  writeln('единиц в двоичной записи: ',m);
  repeat until keypressed;
end.
```

14. БАЛАНС СКОБОК

Задача. Вводится строка символов, признаком конца которой является точка. В строке могут содержаться круглые, квадратные и фигурные скобки — как открывающие, так и закрывающие. Требуется проверить правильность расстановки скобок.

```
Program brackets;

{Заводим стек. Просматриваем строку символов, если встречается открывающая скобка, заносим ее в стек, если — закрывающая, выбираем из стека скобку и проверяем их на соответствие. Скобки будут расставлены правильно, если для каждой очередной закрывающей скобки в строке из стека будет выбрана соответствующая открывающая скобка, стек не будет пуст к началу обработки очередной закрывающей скобки в строке, и по окончании обработки последней из закрывающих скобок в строке стек будет пуст.}

uses crt;
type typelem = char;
     sv = ^zvenst;
     zvenst = record next: sv;
                     elem: typelem
              end;
     stec =sv;
var sym: char; s: stec; b:boolean;

procedure instec(var st:stec; novel:typelem);
  var q:sv;
  begin
    new(q); q^.elem:=novel;
    q^.next:=st; st:=q
  end;

procedure outstec(var st:stec;var a: typelem);
  begin
    a:=st^.elem;
    st:=st^.next;
  end;

function sootv:boolean;
  var r:char;
  begin
    outstec(s,r);
    case sym of
      ')': sootv:= r='(';
      ']': sootv:= r='[';
      '}': sootv:= r='{';
    end
```

```
  end;

begin
  s:=nil;
  writeln('Введите строку символов');
  read(sym);
  b:=true;
  while (sym <> '.') and b do
    begin
      write(sym);
      if sym in [ '(','[','{']
       then instec(s,sym)
       else
         if sym in [')',']','}']
          then
          begin
            if (s=nil) or (not sootv)
            then b:=false
          end;
      read(sym)
    end;
    writeln;
    if not b or (s<>nil) then
       writeln('Скобки расставлены неправильно')
    else writeln('Скобки расставлены  правильно');
  repeat until  keypressed;
end.
```

15. СРАВНЕНИЕ МЕТОДОВ СОРТИРОВКИ

Задача. Коллективное творчество. Сравнить время исполнения различных методов сортировки на массиве случайных чисел

```
program comparison;

uses crt,dos;

const n=10000;                          {Размер сортируемого массива.}
     k=-10;              {Расширение массива для метода ShellSort.}
     fname='rand';                                  {Имя файла.}
```

```
type intar=array[k..n] of integer;

var ar1:intar;
    s : string[6];
    h,m,c,ce,h1,m1,c1,ce1: word;
    i : integer;
    far : text;                    {Текстовый файл для хранения
                                    массива случайных чисел.}

procedure readf(var ar:intar);
{-------------------Чтение значений из текстового файла в массив.---------------
}
var i:integer;
begin reset(far); for i:=1 to n do read(far,ar[i]);
end;

procedure WrOn;
{------------------------Вычисление времени счета.-------------------------------}
  var s2 :string; h2,m2,c2,ce2 : word;
  function InsertZero(w : word) : string;
    var  s : string;
  begin   str(w:0,s);
    if Length(s) = 1 then s := '0' + s;
    InsertZero := s;
  end;
begin h2 := h1 - h;
  if m1 < m then
    begin h2 := h2 - 1; m2 := 60 + m1 - m end
  else m2 := m1 - m;
  if c1 < c then
    begin m2 := m2 - 1; c2 := c1 + 60 - c end
  else c2 := c1 - c;
  if ce1<ce then
    begin
      c2 := c2 - 1;
      e2 := ce1 + 100 - ce
    end
  else ce2:=ce1-ce;
  s2 := InsertZero(m2) + ':' + InsertZero(c2) + ':'
       + InsertZero(ce2);
  writeln(s2:20);
end;
```

```
{----------------------Сортировка методом прямого включения----------------------
}

procedure StraightInsertion(var ar:intar);
```

{Элементы делятся на уже готовую последовательность a1, ..., ai и исходную последовательность. На каждом шаге i из исходной последовательности извлекается i-й элемент и перекладывается в готовую последовательность на нужное место.}

```
var i,j,x : integer;
begin
  for i := 2 to n do
    begin x := ar[i]; ar[0] := x; j := i;
{--------------------------Используется массив с ar[0].-----------------------------}
      while x < ar[j-1] do
        begin
           ar[j] := ar[j-1];
           j := j-1;
        end;
      ar[j] := x;
    end
end;

{----------------------Сортировка методом двоичного включения-------------------
}

procedure BinaryInsertion(var ar:intar);
```

{Это улучшение сортировки методом прямого включения, основанное на том, что готовая последовательность, в которую надо вставить новый элемент, уже упорядочена. При этом используется двоичный поиск, заключающийся в следующем:

- сравнение с серединой готовой последовательности;
- процесс деления пополам, который идет до тех пор, пока не будет найдена точка включения.}

```
var i,j,x,m,l,r : integer;
begin
 for i := 2 to n do
```

```
  begin x := ar[i]; l := 1; r := i;
    while l < r do
      begin m := (l + r) div 2;
        if ar[m] <= x then l := m + 1 else r := m
      end;
    for j := i downto r + 1 do ar[j] := ar[j-1];
    ar[r] := x;
  end
end;
```

{----------Сортировка методом прямого выбора----------}

```
procedure StraightSelection(var ar:intar);
```

{Выполняется следующим образом:

- выбирается элемент с наименьшим значением;
- он меняется местами с первым элементом;
- процесс повторяется с оставшимися n – 1 элементами, n – 2 элементами и т.д., до тех пор, пока не останется один, самый большой элемент.}

```
var i,j,k,x : integer;
begin
   for i := 1 to n-1 do
     begin
       k := i; x := ar[i];
       for  j := i+1 to n do
        if ar[j] < x then
           begin k := j; x := ar[k] end;
       ar[k] := ar[i]; ar[i] := x;
     end
end;
```

{-----Сортировка методом прямого обмена (пузырьковая сортировка)-------
}

```
procedure BubbleSort(var ar:intar);
```

{Данный метод основывается на сравнении и смене мест для пары соседних элементов и продолжении этого процесса до тех пор, пока не будут упорядочены все элементы.}

```
var i,j,x : integer;
begin
  for i :=2 to n do
    for j := n downto i do
      if ar[j-1]>ar[j] then
        begin x := ar[j-1];
          ar[j-1] := ar[j];
          ar[j] := x
        end;
end;
```

{----------Метод шейкерной сортировки----------}

```
procedure ShakerSort(var ar:intar);
```

{Это улучшение пузырьковой сортировки путем запоминания, были или не были перестановки в процессе некоторого прохода. Если в последнем проходе перестановок не было, то алгоритм можно заканчивать. Также происходит чередование направлений последовательных просмотров.}

```
var j,k,l,r,x : integer;
begin
  l := 2; r := n; k := n;
  repeat
    for j := r downto l do
      if ar[j-1] > ar[j] then
        begin
          x := ar[j-1];
          ar[j-1] := ar[j];
          ar[j] := x; k := j;
        end;
    l := k + 1;
    for j := l to r do
      if ar[j-1] > ar[j] then
        begin
          x := ar[j-1];
          ar[j-1] := ar[j];
          ar[j] := x; k := j;
        end;
    r := k-1;
```

```
  until l > r;
end;

{----------------------------Сортировка методом Шелла-------------------------------
-}
{---------------- (включений с уменьшающимися расстояниями) ----------------
}

procedure ShellSort(var a:intar);
```

{Сначала отдельно группируются и сортируются элементы, отстоящие друг от друга на расстоянии 4. Далее элементы перегруппировываются — теперь каждый элемент группы отстоит от другого на две позиции — снова сортируются. На третьем проходе все элементы сортируются методом простых включений. К последнему шагу элементы довольно хорошо упорядочены, и поэтому требуют мало перестановок.}

```
const t = 4;
var i,j,k,x,s : integer;
    m : 1..t;
    h : array [1..t] of integer;
begin
  h[1] := 9; h[2] := 5; h[3] := 3; h[4] := 1;
  for m := 1 to t do
    begin k := h[m]; s := -k;
      for i := k+1 to n do
        begin x := a[i]; j := i – k;
          if s = 0 then s := -k;
          s := s + 1; a[s] := x;
          while x < a[j] do
            begin
              a[j+k] := a[j];
              j := j-k; end;
              a[j+k] := x;
            end
    end;
end;

{-------------------------------Сортировка методом кучи-----------------------------
-}
```

```
{---------------------------------------- (HeapSort) ----------------------------------------
-}

Procedure HeapSort(var ar:intar);

{Является одним из лучших методов сортировки с помощью деревьев.}

var l,r,x : integer;
  procedure Sift (l,r:integer);
  var i,j,x : integer;
  begin
    i:=l;j:=2*i;x:=ar[i];
    while j<=r do
      begin
        if (j<r) then if ar[j]>ar[j+1] then j:=j+1;
        if x<=ar[j] then begin ar[i]:=x;exit end;
        ar[i]:=ar[j]; i:=j; j:=2*i;
      end;
    ar[i]:=x;
  end;
begin
  l := (n div 2) + 1; r := n;
  while l > 1 do begin l := l-1; Sift(l,r); end;
  while r > 1 do
    begin x := ar[1]; ar[1] := ar[r]; ar[r] := x;
          r := r-1; Sift(l,r);
    end
end;

{-----------------------------Метод быстрой сортировки-----------------------------
}

procedure QuickSort(var ar:intar);

{Метод является самым лучшим из известных на данный момент методов
сортировки. Процедура QuikSort представляет собой только "интерфейс"
программы. Собственно сортировка выполняется рекурсивной процедурой
Sort.}

  procedure Sort(l, r: Integer);
  var  i, j, x, y: integer;
```

```
  begin i := l; j := r; x := ar[(l+r) DIV 2];
    repeat
      while ar[i] < x do i := i + 1;
      while x < ar[j] do j := j - 1;
      if i <= j then
        begin y := ar[i]; ar[i] := ar[j];
          ar[j] := y; i := i + 1; j := j - 1;
        end;
    until i > j;
    if l < j then Sort(l, j);
    if i < r then Sort(i, r);
  end;

begin Sort(1,n)end;


begin
  randomize;
{------Создание массива случайных чисел и запись его в текстовой файл-----
}
  assign(far,fname);
  rewrite(far);
  for i:=1 to n do
    begin ar1[i]:=random(n); writeln(far,ar1[i]) end;
  close(far);
  Str(n,s);
  writeln('Сравнение скорости методов сортировки.');
  writeln('На массиве случайных чисел, размера',s:5,);
  write('Исходный массив');
  for i:=1 to n do write(ar1[i]);
  writeln;
  writeln('Название сортировки':25,
         ' Время выполнения':20);

  write('Метод прямого включения':25);
  readf(ar1);
  gettime(h,m,c,ce);
  StraightInsertion(ar1);
  gettime(h1,m1,c1,ce1);
  WrOn;
  write('Отсортированный массив: '); for i:=1 to n do
  write(ar1[i]); writeln;
```

```
write('Метод прямого включения':25);
readf(ar1);
gettime(h,m,c,ce);
StraightInsertion(ar1);
gettime(h1,m1,c1,ce1);
WrOn;
write('Отсортированный массив: ');
for i:=1 to n do write(ar1[i]);
writeln;

write('Метод двоичного включения':25);
readf(ar1);
gettime(h,m,c,ce);
BinaryInsertion(ar1);
gettime(h1,m1,c1,ce1);
WrOn;
write('Отсортированный массив: ');
for i:=1 to n do write(ar1[i]);
writeln;

write('Метод прямого выбора':25);
readf(ar1);
gettime(h,m,c,ce);
StraightSelection(ar1);
gettime(h1,m1,c1,ce1);
WrOn;
write('Отсортированный массив: ');
for i:=1 to n do write(ar1[i]);
writeln;

write('Пузырьковая сортировка':25);
readf(ar1);
gettime(h,m,c,ce);
BubbleSort(ar1);
gettime(h1,m1,c1,ce1);
WrOn;
write('Отсортированный массив: ');
for i:=1 to n do write(ar1[i]);
writeln;

write('Шейкерный метод':25);
readf(ar1);
```

```
  gettime(h,m,c,ce);
  ShakerSort(ar1);
  gettime(h1,m1,c1,ce1);
  WrOn;
  write('Отсортированный массив: ');
  for i:=1 to n do write(ar1[i]);
  writeln;

  write('Метод Шелла':25);
  readf(ar1);
  gettime(h,m,c,ce);
  ShellSort(ar1);
  gettime(h1,m1,c1,ce1);
  WrOn;
  write('Отсортированный массив: ');
  for i:=1 to n do write(ar1[i]);
  writeln;

  write('Метод HeapSort':25);
  readf(ar1);
  gettime(h,m,c,ce);
  HeapSort(ar1);
  gettime(h1,m1,c1,ce1);
  WrOn;
  write('Отсортированный массив: ');
  for i:=1 to n do write(ar1[i]);
  writeln;

  write('Метод быстрой сортировки':25);
  readf(ar1);
  gettime(h,m,c,ce);
  QuickSort(ar1);
  gettime(h1,m1,c1,ce1);
  WrOn;
  write('Отсортированный массив: ');
  for i:=1 to n do write(ar1[i]);
  writeln;

  writeln('Конец работы программы.');
  readln;
end.
```

16. ХАНОЙСКИЕ БАШНИ

Задача. Даны 3 стержня, на одном из них (d1) находятся м дисков. Диски лежат в порядке убывания размера (меньший диск на большем). Перенести диски с d1 на d2, но класть можно только меньший на больший

```
program hanoj;

{-----------Решается с помощью рекурсии--------------}

var m,s1,s2:integer;
procedure disk(l:integer;d1,d2:integer);
begin
 writeln('диск ',l,'со стержня ',d1,'на стержнь ',d2);
end;

procedure bash(h,d1,d2:integer);
  var d3:integer;
begin
  if h=1 then
    disk(h,d1,d2)
  else
    begin
      d3:=6-d1-d2;
      bash(h-1,d1,d3);
      disk(h,d1,d2);
      bash(h-1,d3,d2);
    end;
end;

begin
  writeln('input m,s1,s2');
  read(m,s1,s2);
  writeln('для переноса ',m,' дисков со стержня ',
          s1,' на стержень ',s2,требуется:');
  bash(m,s1,s2);
end.
```

17. ГЕНЕРАЦИЯ ПЕРЕСТАНОВОК

Задача. Получить все перестановки из N элементов в лексикографическом порядке, начиная с перестановки (1, 2, 3, ..., n)

```
program transp;
```

{Начинаем с перестановки (1, 2, 3, ..., n). Переход от текущей перестановки A = (a1, a2, ... an) к перестановке, следующей за текущей в смысле лексикографического порядка, осуществляется последовательным выполнением следующих действий:

- просмотр A справа налево в поисках самой правой позиции i, в которой a[i] < a[i + 1] (если такой позиции i нет, то текущая перестановка является последней и следует закончить порождение);
- просмотр A от a[i] слева направо в поисках наименьшего из таких элементов a[j], что i < j и a[i] < a[j]; поменять местами a[i] с a[j];
- обращение отрезка a[i + 1], ..., a[n] путем транспозиции симметрично расположенных элементов.}

```
uses crt,dos;

var a: array[1..300] of integer;
    i,n,t,r,min,mix: integer;
begin
  clrscr;
  writeln('Введите n'); readln(n);
  if (n > 300) or (n < 1) then
    begin
      writeln('n > 300 or n < 1');
      exit;
    end;
  for i:=1 to n do a[i] := i;
  for mix := 1 to n do write(a[mix],' ');
  writeln('');
  for i:=n-1 downto 1 do
    begin
      if a[i] < a[i+1] then
        begin
          mix := i+1; min := a[i+1];
          for t:=i+1 to n do
            begin
```

```
            if (a[t] < min) and (a[i] < a[t]) then
              begin
                min := a[t];
                mix := t;
              end;
          end;
        t:= a[mix];
        a[mix]:= a[i];
        a[i]:= t;
        for t:=1 to ((n-i) div 2) do
          begin
            r:= a[i+t];
            a[i+t]:= a[n+1-t];
            a[n+1-t]:= r;
          end;
        for mix := 1 to n do write(a[mix],' ');
        writeln('');
        i:=n;
      end;
  end;
end.
```

18. ВЫКЛАДЫВАНИЕ ЧИСЕЛ СПИЧКАМИ

Задача. Сколько можно выложить различных чисел, состоящих из цифр 1, 5, 8 из n спичек (1 <= n <= 75), все n спичек должны быть использованы. При этом цифра 1 выкладывается из двух спичек, 5 из пяти спичек и 8 из семи спичек.

```
program matchs;
```

{A — массив. A[i] — количество различных чисел, которые можно выложить из i спичек. A[1], A[2], ..., A[7] легко находятся. А для n > 7 справедлива формула: A[n] = A[n – 2] + A[n – 5] + A[n – 7], так как из n спичек мы можем получить числа, или начинающиеся на 1 (остается n – 2 спички), или на 5 (остается n – 5 спичек), или на 8 (остается n – 7 спичек).}

```
var a: array [1..75] of longint;
    i, n: integer;
begin
  Write('Enter number of matches: ');
```

```
  ReadLn(n);
  a[1]:=0;
  a[2]:=1;
  a[3]:=0;
  a[4]:=1;
  a[5]:=1;
  a[6]:=1;
  a[7]:=3;
  for i:=8 to n do
  a[i]:=a[i-2]+a[i-5]+a[i-7];
  WriteLn('Number of combinations: ', a[n]);
end.
```

19. ЧАСЫ ДРЕВНИХ МАРСИАН

Задача. 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 возвращается во вторую корзину, а шестой — в третью корзину и так далее (в третьей корзине не может быть больше 7 шариков, в четвёртой — восьми, ...). Часы проработали T секунд. Требуется определить минимальное количество шариков в большой корзине, необходимое для того, чтобы часы успешно проработали ещё P секунд (T + P(1000000000)). Входными данными служат числа T и P.

```
Program clock;
```

{При решении воспользуйтесь тем, что состояние часов в любой момент времени легко описывается с помощью факториальной позиционной системы счисления с основанием, равным 5. 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 приведённых 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.}

```
uses crt;
Type mas=array[1..10] of integer;
Var Q1,Q2:mas;
  T,P:longint;
  i,Sum:integer;

Procedure Trans(num:longint;Var Q:mas);
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);       {Вычисляем очередную цифру.}
     num:=num div (i+4);
     i:=i+1;
    end;
end;

Begin
  writeln('input T P');
  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 + 1 — max цифра в i-ом разряде.}
      i:=i-1
    end;
  writeln(Sum);
  delay(5000);
end.
```

20. ПРЕОБРАЗОВАНИЕ СТРОК

Задача. За минимальное число преобразований (удалений и вставок символов) из строки S1 получить строку S2. Распечатать шаги преобразований.

```
program transf;
```

{Используем метод Ахо. По строкам S1 и S2 заполняем матрицу M. Нулевая строка в M — это номера символов строки S1, нулевой столбец — это номера символов строки S2. Элементы остальных строк матрицы заполняем по правилу: элемент M[i, j] равен min(M[i, j – 1], M[I – 1, j]) + 1, если S1[i] <> S2[j], и равен min(min(M[i, j – 1],M[i – 1, j]) + 1, M[i – 1, j – 1]) иначе. Значение M[length(S2), length(S1)] — это количество шагов преобразований. Далее движемся от M[length(S2), length(S1)] к M[0, 0]. Если переходим от M[i, j] к M[i – 1, j], то после S1[j] вставляем S2[i]. Если переходим от M[i, j] к M[i, j – 1], то удаляем S1[j].}

```
var
  s1,s2:string;
  m:array[0..10,0..10] of integer;
  f:text;
  c,a,i,j:integer;

function min(a,b:integer):integer;
begin
  if a>b then min:=b
  else min:=a;
end;

begin
  assign(f, 'aho.in');
  reset(f);
  readln(f,s1);
  readln(f,s2);
  close(f);
  for i:=0 to length(s1) do
  begin
    m[i,0] := i;
  end;
  for j:=0 to length(s2) do
  begin
```

```
    m[0,j] := j;
  end;
  for i:=1 to length(s1) do
  begin
    for j:=1 to length(s2) do
    begin
    a:=min(m[i-1,j],m[i,j-1])+1;
    if (s1[i]=s2[j]) then a:=min(a, m[i-1,j-1]);
    m[i,j]:=a;
    end;
  end;
  for i:=0 to length(s1) do
    begin
      for j:=0 to length(s2) do write(m[i,j],' ');
      writeln('');
    end;
  writeln(m[length(s1),length(s2)]);
  i:=length(s1);
  j:=length(s2);
  repeat
    if (i>0) then a:=m[i-1,j];
    if (j>0) then a:=m[i,j-1];
    if (i>0) and (j>0) then
      a:=min(min(m[i-1,j],m[i,j-1]),m[i-1,j-1]);
    c:=m[i,j];
    if (a+1<c) then a := c-1;
    if (i>0) and (j>0) and (a=m[i-1,j-1]) then
      begin i:=i-1; j:=j-1; end
    else
      if (i>0) and (a=m[i-1,j]) then
        begin writeln('del(',i,')'); i:=i-1; end
      else
        if (j>0) and (a=m[i,j-1]) then
          begin
            writeln('ins(',i,',''',s2[j],''')');
            j:=j-1;
          end;
  until m[i,j]=0;
end.
```

21. ПЛОЩАДЬ МНОГОУГОЛЬНИКА

Задача. Многоугольник (не обязательно выпуклый) задан на плоскости перечислением координат вершин в порядке обхода его границы. Определить его площадь.

```
Program polygon_area;

{Площадь считается по формуле: S = 0.5 * abs(сумма(r(i)x di)) — векторное произведение, где r(i) = (xi, yi), di = r(I + 1) – r(i). Эта формула не зависит от порядка задания вершин и выбора начала координат. Затем выводится результат.}

uses Crt;

const max=200;                          {Максимальное число точек.}

type point=record x,y:real;
           end;
     mnoz=array[1..max]of point;

var m,i:integer;
    ss,k,s:real;
    razn,mass:mnoz;

begin
  clrscr;
  write('Введите количество точек:');
  readln(m);
  for i:=1 to m do
    begin                                        {Ввод координат.}
      write('X',i,',Y',i,':');
      read(mass[i].x,mass[i].y);
    end;
  mass[m+1].x:=mass[1].x;
  mass[m+1].y:=mass[1].y;
  for i:=1 to m do
    begin                              {Вычисление di, где I = 1..m.}
      razn[i].x:=mass[i+1].x-mass[i].x;
      razn[i].y:=mass[i+1].y-mass[i].y;
    end;
```

```
  ss:=0;
  for i:=1 to m do
    begin                {Суммирование векторных произведений.}
      k:=mass[i].x*razn[i].y-mass[i].y*razn[i].x;
      ss:=ss+k;
    end;
  s:=0.5*abs(ss);                    {Окончательная площадь.}
  writeln('Площадь многоугольника есть число:',s:9:3);
  readkey;
 end.
```

22. ЗАДАЧА О ФЕРЗЯХ

Задача. Существуют способы расстановки 8 ферзей на шахматной доске размером 8*8 так, чтобы они не били друг друга. Составить программу нахождения всех возможных способов.

```
program queens;

{Перебираются всевозможные способы размещения ферзей на шахматной
доске с помощью метода поиска с возвращением.}

const n=8;

var k:integer;              {Счетчик проверки безопасных полей.}
    x:array[1..n] of integer;                        {Решение.}
    a:array[1..n] of boolean;                        {Вертикаль.}
    b:array[1..2*n] of boolean;     {Восходящая диагональ /.}
    c:array[-n+1..n-1] of boolean; {Нисходящая диагональ \.}

procedure init;
{------------------------------Начальные значения.-----------------------------------}
begin
      fillchar(a,sizeof(a),true);
      fillchar(b,sizeof(b),true);
      fillchar(c,sizeof(c),true);
end;

procedure move(i,j:integer);
{------------------------------Фиксируй ход.-----------------------------------------}
```

```
begin
      x[i]:=j;
      a[j]:=false;
      b[i+j]:=false;
      c[i-j]:=false;
end;

procedure back_move(i,j:integer);
{------------------------------Отмена хода.-----------------------------------------}
begin
      a[j]:=true;
      b[i+j]:=true;
      c[i-j]:=true;
end;

function d_hod(i,j:integer):boolean;
{---------------------------Проверка возможности хода.-------------------------}
begin d_hod:=a[j] and b[i+j] and c[i-j]
end;

procedure print;
{-------------------------------Вывод решений.-----------------------------------}
var i,j:integer;
begin
      for i:=1 to n do
      begin
            for j:=1 to n do
                  if j=x[i] then write('i':2)
                  else write('*':2);
            writeln(x[i]:3);
      end;
      writeln(k);
end;
procedure solve(i:integer);
{-------------------------Нахождение решения.--------------------------------}
var j:integer;
begin
      if i=n+1 then
            begin inc(k);print end
      else
            for j:=1 to n do
                  if d_hod(i,j) then
```

```
                        begin
                             move(i,j);
                             solve(i+1);
                             back_move(i,j);
                        end;
end;

{---------------------------------Программа.---------------------------------------}
begin
      init;
      solve(1);
      readln;
end.
```

23. КРАТНОЕ ВЫРАЖЕНИЕ

Задача. Дана строка чисел. Имеются знаки операций + –. Дано число k. Определить, можно ли так расставить знаки (+ –) между числами, чтобы результат полученного выражения делился на К без остатка.

```
program expression;
```

{Берем первое число по модулю k (x := x mod k) и в (x + 1)-й бит массива a заносим 1. Затем x складываем со вторым числом (y) и находим остаток от деления на k, заносим 1 в ((y + x) mod k)-й бит массива b, аналогично заносим 1 в ((x – y) mod k)-й бит массива b. Затем A := B. Массив b обнуляем, и так до последнего числа. В конце смотрим A[1], если A[1] = 1, то ответ положительный, иначе отрицательный.}

```
type mas=array[1..13] of byte;
var a,b:mas;
    f,f1:text;
    n,k,x,i,j,y:integer;

function get_bit(i:integer;a:mas):boolean;
  var k,l:integer;
begin
  get_bit:=false;
  k:=(i div 8)+1;
  l:=(i mod 8);
  if ((a[k] shr (8-l)) mod 2)=1 then get_bit:=true
```

```
end;

procedure put_bit(i:integer;var a:mas);
  var k,l:integer;
begin
  k:=(i div 8)+1;
  l:=i mod 8;
  if ((a[k] shr (8-l)) mod 2)=0 then  a[k]:=a[k]+(1
shl (8-l));
end;

begin
  assign(f,'input.txt');
  assign(f1,'output.txt');
  reset(f);
  rewrite(f1);
  readln(f,n);
  readln(f,k);
  readln(f,x);
  x:=x mod k;
  if x<0 then x:=x+k;
  put_bit(x+1,a);
  for j:= 2 to n do
    begin
      readln(f,y);
      for i:=0 to 99 do
       if get_bit(i+1,a) then
        begin
         x:=(y+i) mod k;
         if x<0 then x:=x+k;
         put_bit(x+1,b);
         x:=(i-y) mod k;
         if x<0 then x:=x+k;
         put_bit(x+1,b);
        end;
      a:=b;
      fillchar(b,sizeof(b),0);
    end;
  if get_bit(1,a) then
    writeln(f1,'divisibility')
  else  writeln(f1,'no');
  close(f);
  close(f1)
```

```
end.
```

24. ПОИСК ПОДСТРОКИ

Задача. Дана строка s и подстрока x. Найти вхождение подстроки в строку. Если подстрока не содержится в строке, сообщить об этом.

```
program boermur;
```

{Для поиска позиции, с которой подстрока входит в строку используется алгоритм Бойера-Мура.}

```
var s,x:string;
    sd:array [0..255] of integer;

procedure search(s,x:string);
var i,j,n,m:integer;
    f:boolean;
    h:char;
begin
  n:=length(s);
  m:=length(x);
  for i:=0 to 255 do sd[i]:=m;
  for i:=1 to m-1 do
    begin
      h:=x[i];
      sd[ord(h)]:=m-i;
    end;
  i:=1; f:=false;
  while (i<n-m+1) and (not f) do
    begin
      j:=m;
      while (j>0) and(s[i+j]=x[j]) do j:=j-1;
      if j=0 then f:=true
      else i:=i+sd[ord(s[i+j])];
    end;
  if f then
    writeln(x,'javljaetsja podstrokoj ',s,
              's pozicii',i+1)
  else writeln('net vchojdenij');
end;
```

```
begin
  writeln('input stroku'); readln(s);
  writeln('input podstroku'); readln(x);
  search(s,x);
  readln
end.
```

СПИСОК ЛИТЕРАТУРЫ

1. **Алексеев А. В.** Олимпиады для школьников по информатике. Задачи и решения. — Красноярск: Красноярское книжное изд-во, 1995. — 222 с.
2. **Арсак Ж.** Программирование игр и головоломок. — М.: Наука, 1990. — 222 с.
3. **Ахо А., Хопктофт Дж., Ульман Дж.** Построение и анализ вычислительных алгоритмов. — М.: Мир, 1979. — 536 с.
4. **Бежанова М. М., Голубева Л. А. Москвина Л. А.** Практическое программирование. Выпуск 2. — Новосибирск: НГУ, 1998. — 109 с.
5. **Беров В. И., Лапунов А. В., Матюхин В. А., Понаморев А. Е.** Особенности национальных олимпиад. — Киров: "Триада-С", 2000. — 151 с.
6. **Вирт Н.** Систематическое программирование: Введение. — М.: Мир, 1977. — 183 с.
7. **Вирт Н.** Алгоритмы + структуры данных = программы. — М.: Мир, 1985. — 406 с.
8. **Задачи** по программированию / Абрамов С. А., Гнездилова Г. Г. и др. — М.: Наука, 1988. — 224 с.
9. **Избранные** задачи олимпиад по информатике / Овсянников А. П., Овсянников Т. В. и др. — М.: Тровант, 1997. — 95 с.
10. **Касьянов В. Н.** Курс программирования на Паскале в заданиях и упражнениях. — Новосибирск: НГУ, 2001. — 448 с.
11. **Касьянов В. Н., Сабельфельд В. К.** Сборник заданий по практикуму на ЭВМ. — М.: Наука, 1986. — 272 с.
12. **Кирюхин В. М., Лапунов А. В., Окулов С. М.** Задачи по информатике. Международные олимпиады 1989-1996 гг. — М.: ABF, 1996. — 269 с.
13. **Практикум** по ТурбоПаскалю / Бабушкина И. А., Бушмилева Н. А. и др. — М.: АБФ, 1998. — 380 с.
14. **Фаронов В. В.** Turbo Pascal 7.0. Начальный курс. — М.: Нолидж, 1997. — 612 с.
15. **Фаронов В. В.** Турбо Паскаль 7.0. Практика программирования. — М.: Нолидж, 1997. – 429 с.
16. **Шень А.** Программирование: теоремы и задачи. — М.: МЦНМО, 1995. – 262 с.

СОДЕРЖАНИЕ

Касьянова Елена Викторовна,
Касьянова Светлана Николаевна

ПРОГРАММИРОВАНИЕ ДЛЯ ШКОЛЬНИКОВ: СБОРНИК ЗАДАЧ ПОВЫШЕННОЙ СЛОЖНОСТИ С РЕШЕНИЯМИ

Препринт
95

Рукопись поступила в редакцию 05.02.2002
Рецензент В. А. Евстигнеев
Редактор З. В. Скок

Подписано в печать 11.03.2002
Формат бумаги 60 × 84 1/16 Объем 2.9 уч.-изд.л., 3.2 п.л.
Тираж 50 экз.

НФ ООО ИПО “Эмари” РИЦ, 630090, г. Новосибирск, пр. Акад. Лаврентьева, 6