Ответы на задачи С4:

1) В этой задаче не нужно хранить в памяти все отсчеты, нас интересуют только средние значения температуры по каждому месяцу и по году, поэтому алгоритм на псевдокоде выглядит так:

```
{ ввод данных, накопление сумм по месяцам и за год }
{ вычисление средних по месяцам и по году }
{ поиск месяца с минимальным отклонением t
  от средней по году }
{ вывод всех месяцев с таким же отклонением }
```

В начале программы не забываем обнулить ячейки, где будут накапливаться суммарные величины:

```
for i:=1 to 12 do tMonth[i]:=0;
tYear:=0;
```

При вводе данных в каждой строке сначала пропускаем все символы до точки (посимвольное чтение), затем читаем номер месяца (целое число) и температуру (вещественное число); температуру добавляем к сумме нужного месяца и к годовой сумме:

```
for i:=1 to DAYS do begin
  repeat read(c); until c='.';
  read(month);
  readln(t);
  tMonth[month] := tMonth[month] + t;
  tYear := tYear + t;
end;
```

Далее находим средние по каждому месяцу и по году (важно! месяцы имеют разное число дней, 2008-ой год – високосный, поэтому в феврале 29 дней)

```
for i:=1 to 12 do
  case i of
          2: tMonth[i] := tMonth[i] / 29;
   4,6,9,11: tMonth[i] := tMonth[i] / 30;
       else  tMonth[i] := tMonth[i] / 31;
  end;
tYear := tYear / DAYS;
```

Определить среднюю температуру по месяцам можно более красиво, если ввести массив констант – дней в каждом месяце:

```
const days: array[1..12] of integer =
              (31,29,31,30,31,30,31,31,30,31,30,31);
```

а потом сделать так:

```
for i:=1 to 12 do
  tMonth[i] := tMonth[i] / days[i];
```

но PascalABC, например, не поддерживает константные массивы.

Теперь можно искать минимальное отклонение среднемесячной температуры от среднегодовой (важно! не забываем ставить модуль):

```
min := abs(tMonth[1] - tYear);
for i:=2 to 12 do
  if abs(tMonth[i] - tYear) < min then
    min := abs(tMonth[i] - tYear);
```

Вывод результата очевиден, приведем сразу полную программу:

```
const DAYS = 366;
var tMonth: array[1..12] of real;
    i, month: integer;
```

```
    t, tYear, min: real;
    c: char;
begin
  for i:=1 to 12 do tMonth[i]:=0;
  tYear:=0;
  for i:=1 to DAYS do begin
    repeat read(c); until c='.';
    read(month);
    readln(t);
    tMonth[month]:=tMonth[month]+t;
    tYear:=tYear+t;
  end;
  for i:=1 to 12 do
    case i of
            2: tMonth[i]:=tMonth[i]/29;
     4,6,9,11: tMonth[i]:=tMonth[i]/30;
         else  tMonth[i]:=tMonth[i]/31;
    end;
  tYear:=tYear/DAYS;
  min:=abs(tMonth[1]-tYear);
  for i:=2 to 12 do
    if abs(tMonth[i]-tYear)<min then
      min:=abs(tMonth[i]-tYear);
  writeln(tYear:0:2);
  for i:=1 to 12 do
    if abs(tMonth[i]-tYear)=min then
      writeln(i,' ',tMonth[i]:0:2,' ',
              tMonth[i]-tYear:0:2);
end.
```

2) Здесь нужно считать одинаковые буквы, которых всего может быть 26 (от **A** до **Z**), причем строчные и заглавные буквы считаются вместе. Поэтому создаем массив счетчиков из 26 элементов:

```
var count: array[1..26] of integer;
```

Для удобства можно сразу коды букв **A** и **a** и записать в целые переменные

```
cA  := Ord('A'); { заглавные }
cAm := Ord('a'); { строчные }
```

В цикле, прочитав очередной символ, находим его код с помощью функции **Ord**,

```
k := Ord(c);
```

Если это заглавная буква, то номер символа в алфавите вычисляется как **k-cA+1**, а для строчных **k-cAm+1**, соответствующий счетчик (элемент массива) нужно увеличить на 1:

```
if ('A' <= c) and (c <= 'Z') then
  count[k-cA+1] := count[k-cA+1] + 1;
if ('a' <= c) and (c <= 'z') then
  count[k-cAm+1] := count[k-cAm+1] + 1;
```

Когда все данные (до первой точки) введены, остается найти номер максимального элемента (переменная **iMax**), а затем вывести на экран соответствующий символ и количество повторений. Вот полная программа:

```
var count:array[1..26] of integer;
    i, k, cA, cAm, iMax:integer;
    c: char;
begin
```

```
  cA  := Ord('A');
  cAm := Ord('a');
  for i:=1 to 26 do count[i] := 0;
  repeat
    read(c);
    k := Ord(c);
    if ('A' <= c) and (c <= 'Z') then
      count[k-cA+1] := count[k-cA+1] + 1;
    if ('a' <= c) and (c <= 'z') then
      count[k-cAm+1] := count[k-cAm+1] + 1;
  until c = '.';
  iMax := 1;
  for i:=2 to 26 do
    if count[i] > count[iMax] then iMax := i;
  writeln(char(cA+iMax-1), ' ', count[iMax]);
 end.
```

Возможно, несколько лучшее решение получится, если использовать массив счетчиков с символьными индексами (это можно сделать в Паскале, но не во всех языках программирования):

```
var count:array['A'..'Z']of integer;
```

После чтения символа удобно сразу привести его к верхнему регистру с помощью функции **UpCase** (преобразовать строчные буквы в заглавные):

```
c := UpCase(c);
```

или (если в вашей версии Паскаля ее нет) вручную

```
if c in ['a'..'z'] then
   c := Char(Ord(c) - Ord('a') + Ord('A'));
```

Если символ – латинская буква, то увеличиваем соответствующий счётчик:

```
if c in ['A'..'Z'] then Inc(count[c]);
```

Поиск максимума и вывод результата тоже упрощаются:

```
iMax:='A';
for c:='B' to 'Z' do
  if count[c] > count[iMax] then iMax:=c;
writeln(iMax,' ',count[iMax]);
```

Отметим, что такое красивое решение возможно только в тех языках программирования, где есть массивы с симврольными индексами. Вот полная программа:

```
var c, iMax:char;
    count: array['A'..'Z'] of integer;
begin
  for c:='A' to 'Z' do count[c]:=0;
  repeat
    read(c);
    if c in ['a'..'z'] then
       c := Char(Ord(c) - Ord('a') + Ord('A'));
    if c in ['A'..'Z'] then Inc(count[c]);
  until c = '.';
  iMax:='A';
  for c:='B' to 'Z' do
    if count[c] > count[iMax] then iMax := c;
  writeln(iMax,' ',count[iMax]);
end.
```

3) Все аналогично предыдущей задаче с двумя изменениями: заглавных букв нет и нужно вывести количество для всех букв. Код программы:

```
var count:array[1..26] of integer;
    i, k, cA:integer;
    c: char;
begin
  cA := Ord('a');
  for i:=1 to 26 do count[i] := 0;
  repeat
    read(c);
    k := Ord(c);
    if ('a' <= c) and (c <= 'z') then
      count[k-cA+1] := count[k-cA+1] + 1;
  until c = '.';
  for i:=1 to 26 do
    if count[i] > 0 then
      writeln(char(cA+i-1), count[i]);
 end.
```

Возможен и другой вариант (идею предложил Р. Басангов, МОУ «СОШ 3» г. Элиста), в котором используется массив с символьными индексами:

```
count: array ['a'..'z'] of integer;
```

Вот полное решение:

```
var count: array ['a'..'z'] of integer;
    c: char;
begin
 for c:='a' to 'z' do count[c]:=0;
 repeat
   read (c);
   if ('a' <= c) and (c <= 'z') then
     count[c] := count[c] + 1;
 until c = '.';
 for c:='a' to 'z' do
   if count[c]>0 then
     writeln(c, count[c]);
end.
```

4) Заметим, что в этой задаче мы должны хранить в памяти все фамилии и считать, сколько раз они встретились. При этом имена нас не интересуют, поэтому можно выделить такой массив записей

```
var Info: array[1..100] of record
            name: string;    { фамилия }
            count: integer;  { счетчик }
          end;
```

Второе поле (счётчик **count**) показывает, какая это запись по счёту с той же самой фамилией. Например, если счётчик равен 5, раньше эта фамилия встречалась уже 4 раза. В этой задаче легко читать информацию целыми строками, а затем «вырезать» фамилию с помощью стандартных функций (фамилия окажется в строке **s**):

```
      readln(s);
      p := Pos(' ', s);
      s := Copy(s,1,p-1);
```

Теперь проверяем, сколько таких фамилий уже есть в списке. Нужно в цикле просмотреть **i-1** первых элементов массива **Info** (где **i** – номер обрабатываемой строки), если фамилия в очередной записи совпала с только что введенной, счетчик (переменная **c**) увеличивается на 1:

```
c := 1;
for k:=1 to i-1 do
  if s = Info[k].name then
    c := c + 1;
```

Затем записываем фамилию ученика и значение счётчика в очередную запись:

```
Info[i].name := s;
Info[i].count := c;
```

После обработки всех строк остается вывести на экран результат (список логинов). Если счётчик равен 1, фамилия встратилась в первый раз, и логин совпадает с фамилией. Если счётчик больше 1, его значение дописывается в конец фамилии (получаются логины вида «Иванов2», «Иванов3» и т.п.):

```
for i:=1 to N do begin
  write(Info[i].name);
  if Info[i].count > 1 then
    write(Info[i].count);
  writeln;
end;
```

Вот полный код программы:

```
var Info: array[1..100] of record
            name: string;
            count: integer;
          end;
    i, k, p, N, c: integer;
    s: string;
begin
  readln(N);
  for i:=1 to N do begin
    readln(s);
    p := Pos(' ', s);
    s := Copy(s,1,p-1);
    c := 1;
    for k:=1 to i-1 do
      if s = Info[k].name then
        c := c + 1;
    Info[i].name := s;
    Info[i].count := c;
  end;
  for i:=1 to N do begin
    write(Info[i].name);
    if Info[i].count > 1 then write(Info[i].count);
    writeln;
  end;
end.
```

Д.Ф. Муфаззалов (УГАТУ, Уфа) предложил решение без структур (записей), в котором поиск полученной фамилии среди существующих выполняется не с начала массива, а с его конца – до первого совпадения:

```
  j := i - 1;
  while  (j > 0) and  (fam[i] <> fam[j]) do
    dec(j);
```

Если такая фамилия не найдена и в результате мы получили **j=0**, принимаем значение счётчика **count**, равное 1:

```
var
  count: array[0..100] of byte;
  fam: array[1..100] of string;
  i, j, n: byte;
begin
  readln(n);
  count[0] := 0;
  for i := 1 to n do
  begin
    readln(fam[i]);
    fam[i] := copy(fam[i], 1, pos(' ', fam[i]) - 1);
    j := i - 1;
    while  (j > 0) and  (fam[i] <> fam[j]) do dec(j);
    count[i] := count[j] + 1;
  end;
  for i := 1 to n do
  begin
    write(fam[i]);
    if count[i] > 1 then write(count[i]);
    writeln;
  end;
end.
```

Фомин Артем (шк. 1523, Москва) предложил следующее решение. Все фамилии учеников сохраняются, а затем для каждого необработанного ученика находятся все однофамильцы, и из них присваивается его порядковый номер (среди однофамильцев). Таким образом, проход по массиву осуществляется лишь столько раз, сколько различных фамилий было у учеников.

Решение на GCC 4.6.3:

```
#include <iostream>
using namespace std;
int main()
{
  int n;
  cin >> n;
  string s[n], name;
  int kol, k[100] = {0}, j, i;
  for (i = 0; i < n; i++) cin >> s[i] >> name;
  for (i = 0; i < n; i++)
   if (!k[i])// если ученик не обработан
    for (kol=j=0; j < n; j++)
     if (s[i]==s[j]) k[j] = ++kol;//присваиваем номер
  for (i = 0; i < n; i++)
  {
    cout << s[i];
    if (k[i]>1) cout << k[i];
    cout << endl;
```

```
    }
      return 0;
}
```

Д.Ф. Муфаззалов (УГАТУ, Уфа) предложил еще одно решение. В нем сохраняются не все фамилии, а только уникальные, и записывается количество раз, которое встретилась каждая из них.

Решение на GCC 4.6.3:

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    cin >> n;
    string s[n], name,f;
    int kol, k[100] = {1}, j, m = 1;
    // m – количество уникальных фамилий
    cin >> s[0] >> name; // первую фамилию вводим вне цикла
    cout << s[0] << endl;
    for (n--; n--;)
    {
        cin >> f >> name;
        // проверяем, встречалась ли уже такая фамилия
        for (j = 0; j<m-1 && f!=s[j]; j++);
        cout << f; // выводим фамилию
        // если встречалась – выводим номер
        if (f==s[j]) cout << ++k[j];
        else
        {   s[m]=f; // если нет – сохраняем фамилию
            k[m++]++;
        }
        cout <<endl;
    }
    return 0;
}
```

Решение на Паскале ABC.NET:

```
var  n, j, m, i, kol: integer;
  s: array[1..100] of string;
  name, f: string;
  k: array[1..100] of integer;
begin
  readln(n);
  m := 1; // m – количество уникальных фамилий
  readln(s[1]); // первую фамилию вводим вне цикла
  s[1] := copy(s[1], 1, pos(' ', s[1]) - 1);
  writeln(s[1]);
  k[1] := 1;
  for i := 2 to n do
  begin
    readln(f);
    f := copy(f, 1, pos(' ', f) - 1);
```

```
      // проверяем, встречалась ли уже такая фамилия
      j := 1;
      while (j < m) and (f <> s[j]) do inc(j);
      write(f); // выводим фамилию
          // если встречалась – выводим номер
      if (f = s[j]) then
      begin
        inc(k[j]);
        write(k[j])
      end
          else // если нет – сохраням фамилию
      begin
        inc(m);
        s[m] := f;
        inc(k[m]);
      end;
      writeln
    end
  end.
```

5) Это упрощенный вариант второй задачи, подробно разобранной в основной части. Отличия: нужно найти максимум вместо минимума, и только один, а не три.

```
  const LIM = 250;
  var Info: array[1..LIM] of record
              name: string;
              sum: integer;
            end;
      i, k, N, mark, max: integer;
      c: char;
  begin
  readln(N);
      { ввод исходных данных }
  for i:=1 to N do begin
    Info[i].name := '';
    for k:=1 to 2 do
      repeat
        read(c);
        Info[i].name := Info[i].name + c;
      until c = ' ';
    Info[i].sum := 0;
    for k:=1 to 3 do begin
      read(mark);
      Info[i].sum := Info[i].sum + mark;
    end;
    readln;
  end;
      { поиск максимума}
  max := Info[1].sum;
  for i:=2 to N do
    if Info[i].sum > max then
      max := Info[i].sum;
      { вывод результата }
```

```
  for i:=1 to N do
    if Info[i].sum = max then
       writeln(Info[i].name);
  end.
```

Возможен другой вариант решения (**А.С. Абрамов, лицей при РГСУ, г. Воронеж**), основанный на следующей идее: в массив записываются фамилии только тех участников, которые имеют суммарный балл, равный максимальному на данный момент; если максимум меняется, возвращаемся к 1-му элементу массива и следующую «цепочку» максимумов записываем поверх предыдущей. Обработка данных выполняется сразу при вводе, отдельный поиск максимума не требуется.

Целая переменная **count** будет обозначать количество найденных участников с максимальным баллом. В переменной **max** будем хранить максимальный (на данный момент) результат, а в переменной **ball** накапливать сумму баллов очередного участника. Тогда алгоритм обработки выглядит так (переменная **s** содержит фамилию и имя):

```
  if ball > max then begin { новый максимум }
    count := 1;
    max := ball;
    names[1] := s;
  end
  else
    if ball = max then begin { еще один участник в списке }
      count := count + 1;
      names[count] := s;
    end;
```

Вот полная программа:

```
const LIM = 250;
var names: array[1..LIM] of string;
    i, k, N, ball, mark, max, count: integer;
    s: string;
    c: char;
begin
  readln(N);          { ввод количества участников }
  max := 0; count:=0;
    { ввод данных в цикле }
  for i:=1 to N do begin
     s := '';
     for k:=1 to 2 do { читаем фамилию и имя }
       repeat
         read(c);
         s := s + c;
       until c = ' ';
       { считываем и суммируем баллы }
     ball := 0;
     for k:=1 to 3 do begin
       read(mark);
       ball := ball + mark;
     end;
     readln;
       { ищем участников с максимальным баллом }
```

```
      if ball > max then begin
        count := 1;
        max := ball;
        names[1] := s;
      end
      else
      if ball = max then begin
        count := count + 1;
        names[count] := s;
      end;
    end;
     { вывод результата }
  for i:=1 to count do
    writeln(names[i]);
end.
```

6) Это вариант второй задачи, подробно разобранной в основной части. Отличия: нужно найти максимум вместо минимума, сумма складывается из четырех оценок.

```
const LIM = 100;
var Info: array[1..LIM] of record
            name: string;
            sum: integer;
          end;
    i, k, N, mark, max1, max2, max3: integer;
    c: char;
begin
readln(N);
    { ввод исходных данных }
for i:=1 to N do begin
  Info[i].name := '';
  for k:=1 to 2 do
    repeat
      read(c);
      Info[i].name := Info[i].name + c;
    until c = ' ';
  Info[i].sum := 0;
  for k:=1 to 4 do begin
    read(mark);
    Info[i].sum := Info[i].sum + mark;
  end;
  readln;
end;
    { поиск трех максимальных }
max1 := 0; max2 := 0; max3 := 0;
for i:=1 to N do begin
  if Info[i].sum > max1 then begin
    max3 := max2; max2 := max1;
    max1 := Info[i].sum;
  end
  else if Info[i].sum > max2 then begin
      max3 := max2;
      max2 := Info[i].sum;
```

```
      end
      else if Info[i].sum > max3 then
           max3 := Info[i].sum;
end;
    { вывод результата }
for i:=1 to N do
  if Info[i].sum >= max3 then
     writeln(Info[i].name);
end.
```

Еще один вариант решения представил **Д.Ф. Муфаззалов** (г. Уфа). Он основан на методе поиска трех лучших (максимальных) элементов в массиве, предложенном Е. В. Андреевой (лекция «Разбор задач группы С» от 21.02.2013). Она предложила использовать сортировку, при которой на каждой итерации максимальный элемент из неотсортированной части массива помещается в начало этой неотсортированной части. Если вы не знакомы с сортировкой выбором, но освоили сортировку пузырьком, можно использовать тот факт, что «обратная» сортировка пузырьком (такую сортировку можно назвать «методом камня» – тяжелый камень проваливается на «дно массива») на каждой итерации помещает больший элемент из неотсортированной части массива в конец этой неотсортированной части. Выполнив 3 итерации такой сортировки, мы получим в конце массива 3 максимальных элемента; на третьем с конца месте будет находиться минимальный из них. Сложность алгоритма составит $O(3N)$, по словам Е.В. Андреевой, признать его неэффективным эксперты не могут.

```
const MAX = 100;
var name: array[1..MAX] of string;
    ball:array[1..MAX] of byte;
    s: string;
    mark, k, j, i, N: integer;
    c: char;
begin
  readln(n);
  for i:=1 to n do begin
    name[i]:=''; ball[i]:=0;
    for k:=1 to 2 do
      repeat
        read(c);
        name[i]:= name[i] + c;
      until c = ' ';
    for k:=1 to 4 do begin
      read(mark);
      ball[i]:= ball[i] + mark;
    end;
    readln;
  end;
  { "метод камня" – за один проход самый тяжелый
    элемент упадет "на дно" массива }
  for i:=1 to 3 do { делаем только три прохода по массиву }
    for j:=1 to n-i do
      if ball[j] > ball[j+1] then begin
        mark:=ball[j]; ball[j]:=ball[j+1]; ball[j+1]:=mark;
        s:=name[j]; name[j]:=name[j+1]; name[j+1]:=s;
```

```
      end;
    { выводим всех, у которых баллы не ниже третьего }
    for i:=1 to n do
      if ball[i] >= ball[n-2] then
        writeln(name[i]);
end.
```

7) Особенность этой задачи в том, что фамилии на выходе нужно отсортировать. «Школьные» сортировки имеют сложность $O(N^2)$; это вообще говоря, не лучший вариант, но без сортировки здесь не обойтись. Применять «быстрые» сортировки (например, *QuickSort*) не следует, даже если вы их знаете – эксперты могут не понять. Читаем очередную строку посимвольно до второго пробела, накапливаем строку в переменной **s** – там окажется фамилия вместе с именем:

```
s := '';
for k:=1 to 2 do
  repeat
    read(c);
    s := s + c;
  until c = ' ';
```

Теперь читаем два числа,

```
readln(mark1, mark2);
```

Учитывая, что до конца строки больше нет данных, используем оператор **readln**, а не **read**. Если хотя бы одна из оценок меньше 30, увеличиваем счетчик «неудачников» (переменная **count**) и записываем фамилию и имя (из переменной **s**) в элемент массива **name** с номером **count**:

```
if (mark1 < 30) or (mark2 < 30) then begin
  count := count + 1;
  name[count] := s;
end;
```

После чтения всех данных массив фамилий «неудачников» нужно отсортировать, здесь мы используем простейший метод – классический «пузырек». Не забываем, что нужно сортировать не все **N** строк в массиве **name**, а только **count** (столько, сколько нашли «неудачников»):

```
for i:=1 to count-1 do
  for k:=count-1 downto i do
    if name[k] > name[k+1] then begin
      s := name[k]; name[k] := name[k+1];
      name[k+1] := s;
    end;
```

Вот полная программа:

```
const LIM = 500;
var name: array[1..LIM] of string;
    i, k, count, mark1, mark2, N: integer;
    c: char;
    s: string;
begin
readln(N);
    { ввод исходных данных }
count := 0;
for i:=1 to N do begin
  s := '';
```

```
    for k:=1 to 2 do
      repeat
        read(c);
        s := s + c;
      until c = ' ';
    readln(mark1, mark2);
    if (mark1 < 30) or (mark2 < 30) then begin
      count := count + 1;
      name[count] := s;
    end;
  end;
      { сортировка }
  for i:=1 to count-1 do
    for k:=count-1 downto i do
      if name[k] > name[k+1] then begin
        s := name[k]; name[k] := name[k+1];
        name[k+1] := s;
      end;
      { вывод результата }
  for i:=1 to count do
    writeln(name[i]);
  end.
```

Возможен ещё один вариант, который позволяет избежать сортировки в конце (**Е.В. Хламов, Иркутск**). Можно сортировать данные сразу же при вставке в массив:

```
  const LIM = 500;
  var name: array[1..LIM] of string;
      i, k, count, mark1, mark2, N: integer;
      c: char;
      s: string;
  begin
  readln(N);
      { ввод исходных данных }
  count := 0;
  for i:=1 to N do begin
    s := '';
    for k:=1 to 2 do
      repeat
        read(c);
        s := s + c;
      until c = ' ';
    readln(mark1, mark2);
    if (mark1 < 30) or (mark2 < 30) then begin
       { ищем место для вставки, начиная с конца массива }
      k:= count;




      while (k > 0) and (s < name[k])  do begin
        name[k+1]:= name[k];
        k:=k-1;
      end;
          { ставим новый элемент }
```

```
      count := count + 1;
      name[k+1] := s;
    end;
  end;
      { вывод результата }
  for i:=1 to count do
    writeln(name[i]);
  end.
```

8) Так как номера телефонов подразделений отличаются только двумя последними цифрами, задача сводится к тому, чтобы подсчитать, сколько различных чисел (номеров подразделений) встречается в этой последней части. Их может быть не более 100 (от 0 до 99), поэтому вводим массив из 100 элементов:

```
  var podr: array[1..100] of integer;
```

Количество найденных разных подразделений будем хранить в целой переменной **count** (это счетчик, в начале в него нужно записать 0).

Нас не интересуют фамилии и имена сотрудников, а также их полные телефоны. Поэтому при чтении строки пропускаем все символы до второго знака «–» включительно:

```
  for k:=1 to 2 do
    repeat
      read(c);
    until c = '-';
```

затем читаем номер подразделения в целую переменную **p** и проверяем, нет ли его в массиве **podr** (если есть – логическая переменная **exist** устанавливается в **True**):

```
  for k:= 1 to count do
    if podr[k] = p then begin
       exist := True;
       break;
    end;
```

С помощью оператора **break** досрочно выходим из цикла, если прочитанный номер уже есть в массиве. Если номер не нашли, увеличиваем счетчик и сохраняем этот номер в очередном элементе массива:

```
    if not exist then begin
      count := count + 1;
      podr[count] := p;
    end;
```

После этого остается разделить общее число сотрудников **N** на количество подразделений. Вот полная программа:

```
  var podr: array[1..100] of integer;
      i, k, p, count, N: integer;
      c: char;
      exist: boolean;
      av: real;
  begin
  readln(N);
      { ввод исходных данных }
  count := 0;
  for i:=1 to N do begin
    for k:=1 to 2 do
      repeat read(c); until c = '-';
    readln(p);
```

```
    exist := False;
    for k:= 1 to count do
      if podr[k] = p then begin
         exist := True;
         break;
      end;
    if not exist then begin
      count := count + 1;
      podr[count] := p;
    end;
  end;
      { вывод результата }
  av := N / count;
  writeln(av:0:2);
  end.
```

Еще одно, более красивое решение этой задачи, предложила *Л.Б. Кулагина* (ФМЛ № 239, г. Санкт-Петербург). Идея заключается в том, чтобы создать массив логических значений (по количеству возможных подразделений), сначала в каждый его элемент записать **false** и при чтении номера подразделения в соответствующий элемент записывать **true** (нашли этот номер). В конце программы для определения количества подразделений останется подсчитать, сколько элементов массива имеют значение **true**.

```
  var podr: array[0..99] of boolean;
      i, k, p, count, N: integer;
      c: char;
      av: real;
  begin
  readln(N);
      { ввод исходных данных }
  for i:=0 to 99 do
    podr[i]:=False; { еще ничего не нашли }
  for i:=1 to N do begin
    for k:=1 to 2 do
      repeat read(c); until c = '-';
    readln(p);
    podr[p]:=True;
  end;
  count := 0;
      { считаем найденные подразделения }
  for i:=0 to 99 do
    if podr[i] then count:=count+1;
      { вывод результата }
  av := N / count;
  writeln(av:0:2);
  end.
```

Если нет желания работать с логическим массивом, можно вполне обойтись целочисленным. В этом случае в самом начале в его элементы нужно записать нули (вместо **False**). Целочисленный массив позволит решить подобную задачу в том случае, если нам нужно будет знать количество сотрудников в каждом подразделении отдельно, тогда после чтения номера подразделения нужно увеличить соответствующий элемент массива, который является счетчиком:

```
    podr[p] := podr[p] + 1;
```

Немного изменится и подсчет количества подразделений:

```
for i:=0 to 99 do
  if podr[i] > 0 then  count:=count + 1;
```

Существует еще один способ решения, который в данном случае, по-видимому, и является оптимальным. Однако в нем используются множества, которые в основном школьном курсе чаще всего не изучаются. Множество (англ. *set*) может включать некоторое (заранее неизвестное, а отличие от массива) количество элементов. В Паскале элементами множества могут быть целые числа от 0 до 255 или символы (точнее, коды символов). В данном случае код подразделения – целое число от 0 до 99, поэтому множество можно объявить так:

```
var podr: set of 0..99;
```

или так:

```
var podr: set of byte;
```

Во втором случае в множество могу входить любые числа от 0 до 255.

Когда мы прочитали номер подразделения в переменную **p**, нужно проверить, входит ли это число во множество. Если входит, то ничего делать не требуется, а если не входит, нужно добавить его к множеству:

```
if not (p in podr) then begin
   podr := podr + [p]; { добавить к множеству }
   count := count + 1; { увеличить счетчик подразделений }
end;
```

Запись **[p]** обозначает множество из одного элемента, а знак «плюс» – объединение множеств. Кроме того, нужно увеличить счетчик подразделений **count** (поскольку нет простого способа сразу определить количество элементов множества).

```
var podr: set of 0..99;
    p: byte;
    i, k, N, count: integer;
    c: char;
    av: real;
begin
podr := [];
count := 0;
    { ввод исходных данных }
readln(N);
for i:=1 to N do begin
  for k:=1 to 2 do
    repeat read(c); until c = '-';
  readln(p);
  if not (p in podr) then begin
     podr := podr + [p]; { добавить к множеству }
     count := count + 1; { увеличить счетчик подразделений }
  end;
end;
    { вывод результата }
av := N / count;
writeln(av:0:2);
end.
```

По-видимому, это решение действительно наиболее эффективно в данной конкретной задаче. Однако, нужно помнить, что в других аналогичных задачах такой подход часто не работает из-за существенных ограничений множеств:

- число элементов множества не может быть больше 256;
- элементами множества могут быть только числа от 0 до 255;
- элементами множества не могут быть символьные коды, например, AB34a.

С учетом этого первое из рассмотренных решений является наиболее универсальным.

9) Эта задача имеет очень длинное условие, но решается довольно просто. Сначала нужно «вычленить» из условия и осознать существенные моменты:

- нужные нам участники получили наибольший балл (если нет победителей) или второй по величине (если победители есть); участники с более низкими баллами нас не интересуют;
- нам нужно хранить имя только одного из искомых участников, а не всех;
- класс нас не интересует.

Таким образом, для решения задачи при вводе исходных данных достаточно определить:

- количество участников, получивших высший балл, и имя одного из них;
- количество участников, получивших второй по величине балл, и имя одного из них;

Важно понять, что здесь не нужно заводить массивы для хранения всех имен и результатов в памяти; строго говоря, сделать это невозможно, потому что количество участников по условию не ограничено. Также не нужна никакая сортировка.

Для хранения данных заведем три целочисленных массива, каждый из которых состоит из двух элементов:

```
var ballBest: array[1..2] of integer; { результат (баллы) }
    numBest: array[1..2] of integer;  { количество }
    nameBest: array[1..2] of string;  { имена }
```

Первые элементы этих массивов относятся к тем, кто набрал наивысший балл, а вторые – к тем, кто набрал второй по величине балл.

Программа в целом выглядит так:

```
var ballBest: array[1..2] of integer;
    numBest: array[1..2] of integer;
    nameBest: array[1..2] of string;
    N: integer;           { число участников }
    c: char;              { символ для ввода }
    i, j, k, ball: integer; { вспомогательные переменные }
    name: string;
begin
  Readln(N);              { ввод числа участников }
  ballBest[1] := -1;       { начальное значение, < 0}
  for i:=1 to N do begin
    { читаем фамилию и имя }
    { пропускаем класс }
    { читаем баллы участника }
    { обрабатываем баллы }
  end;
  { определяем, есть ли победители }
  { выводим результат }
end.
```

Теперь последовательно рассмотрим все блоки, обозначенные комментариями.

Начальное значение **ballBest[1]** должно быть меньше, чем самый низкий возможный результат, поэтому можно записать туда любое отрицательное число (так, чтобы у

первого же участника был результат больше). Для остальных элементов массивов начальные значения не нужны.

Чтение фамилии и имени в символьную строку **name** мы уже рассматривали ранее:

```
name := '';
for j:=1 to 2 do
  repeat
    read(c);
    name := name + c;
  until c = ' ';
```

Пропуск класса также выполняется стандартно:

```
read(k);
```

Дальше читаем результат участника (баллы) – чтение до конца строки:

```
readln(ball);
```

Что делать с этими баллами? Нас интересуют 4 варианта, при которых изменяются массивы **ballBest**, **numBest** и **nameBest**, определяющие результат:

- **ball > ballBest[1]**, новый участник набрал больше баллов, чем все предыдущие; в этом случае надо скопировать все 1-ые элементы массивов во 2-ые, а затем в 1-ые записать данные нового участника (имя, баллы, количество = 1);
- **ball = ballBest[1]**, новый участник набрал столько же баллов, сколько лучшие из предыдущих; нужно увеличить их количество на 1;
- **ballBest[2] < ball < ballBest[1]**, новый участник набрал «второе» количество баллов; нужно во 2-ые элементы массивов записать данные нового участника (имя, баллы, количество = 1);
- **ball = ballBest[2]**, новый участник набрал столько же баллов, сколько участники с вторым известным ранее результатом; нужно увеличить их количество на 1.

Остальные варианты (когда **ball < ballBest[2]**) нас не волнуют, потому что они не влияют на результат. Ниже приведен блок обработки прочитанного количества баллов нового участника. Обратите внимание, что каждый новый **if** вложен в блок **else** предыдущего условного оператора. Подумайте, почему это необходимо.

```
if ball > ballBest[1] then begin
  ballBest[2] := ballBest[1];
  numBest[2] := numBest[1];
  nameBest[2] := nameBest[1];
  ballBest[1] := ball;
  numBest[1] := 1;
  nameBest[1] := name;
end
else
  if ball = ballBest[1] then
    numBest[1] := numBest[1] + 1
  else
    if ball > ballBest[2] then begin
      ballBest[2] := ball;
      numBest[2] := 1;
      nameBest[2] := name;
    end
    else
      if ball = ballBest[2] then
        numBest[2] := numBest[2] + 1;
```

Теперь определим есть ли победители, то есть, верно ли, что **ballBest[1] > 200** и **numBest[1]** не превышает 20% от **N**. Если эти два условия верны одновременно, победители есть, и для ответа нужно использовать вторые элементы массивов (запишем в переменную **i** значение 2), иначе – первые.

```
if (ballBest[1] > 200) and (numBest[1]*100 <= N*20) then
     i := 2
else i := 1;
```

Обратите внимание, что во втором условии используется отношение «меньше или равно» (нестрогое равенство). Кроме того, определение доли 20% сведено к операциям только с целыми числами! Вариант **numBest[1]<=N*0.2** хуже, потому что выражение в правой части неравенства – вещественное, а большинство вещественных чисел (в том числе, 0,2) невозможно точно представить в памяти (они представляют собой бесконечную дробь).

Остается вывести результат на экран. Если искомых участников больше 1, выводим их количество **numBest[i]**, иначе – имя единственного участника **nameBest[i]**.

```
if numBest[i] > 1 then
     writeln(numBest[i])
else writeln(nameBest[i]);
```

Вместо трех массивов можно использовать массив структур, состоящих из трех полей. Приведем сразу полную программу со структурами:

```
var Best: array[1..2] of record
      ball: integer;
      num: integer;
      name: string;
    end;
    N: integer;
    c: char;
    i, j, k, ball: integer;
    name: string;
begin
  Readln( N);
  Best[1].ball := -1;
  for i:=1 to N do begin
    name := '';
    for j:=1 to 2 do
      repeat
        Read( c);
        name := name + c;
      until c = ' ';
    Readln(k, ball);
    if ball > Best[1].ball then begin
      Best[2] := Best[1];
      Best[1].ball := ball;
      Best[1].num := 1;
      Best[1].name := name;
    end
    else
      if ball = Best[1].ball then
        Best[1].num := Best[1].num + 1
      else
        if ball > Best[2].ball then begin
          Best[2].ball := ball;
```

```
            Best[2].num := 1;
            Best[2].name := name;
          end
          else
            if ball = Best[2].ball then
              Best[2].num := Best[2].num + 1
    end;
    if (Best[1].ball > 200) and
       (Best[1].num*100 <= N*20) then
         i := 2
    else i := 1;
    if Best[i].num > 1 then
        writeln(Best[i].num)
    else writeln(Best[i].name);
  end.
```

10) Прежде всего, нужно понять, что «магазин» определяется сочетанием «Фирма + Улица». Каждый магазин может продавать сметану разных сортов, каждому сорту соответствует своя строчка в исходных данных. Важно, что нам **НЕ** нужно запоминать ни фирму, ни улицу, поэтому при чтении их можно вообще пропускать.

Фактически это задача на поиск количества минимальных элементов в потоке данных, причем нужно отдельно работать с тремя наборами данных (молоко разной жирности). Введем массивы из трех элементов для хранения минимальной цены и количества магазинов, продающих по этой цене:

```
var MinPrice, Count: array[1..3] of integer;
```

Для поиска минимальных элементов нужно записать начальные значения: в каждый элемент массива **MinPrice** – любую цену, больше максимально возможной, а все счетчики обнулить.

```
    for i:=1 to 3 do begin
      MinPrice[i] := 5001;   { любое число > 5000 }
      Count[i] := 0;         { обнулить счетчики }
    end;
```

Дальше возникает следующий вопрос: как, прочитав из файла жирность в процентах **k**, рассчитать номер соответствующего элемента массива (код):

15 →1, 20 →2, 25 →3

Оказывается, это сделать достаточно просто, код рассчитывается по формуле

k div 5 - 2

Как эту формулу получить? Мы видим, что при увеличении **k** *на 5 код увеличивается на 1, поэтому мы имеем дело с линейной зависимостью с коэффициентом 1/5:*

код = k div 5 + b

Свободный член b подбирается, например, из условия **15 div 5 + b = 1** *(при* **k = 15** *мы должны получить код 1).* Тогда **3 + b = 1** и **b = -2**.

Если в какой-то задаче числа совсем «нескладные» и не удается вывести формулу, можно использовать оператор выбора (**case**) или серию условных операторов. Никаких других хитростей в программе нет:

```
program milk;
var MinPrice, Count: array[1..3] of integer;
    N: integer;
    c: char;
    i, j, k, price: integer;
begin
```

```
    Readln(N);
    for i:=1 to 3 do begin
      MinPrice[i] := 5001;
      Count[i] := 0;
    end;
    for i:=1 to N do begin
      for j:=1 to 2 do    { пропускаем фирму и улицу }
        repeat read(c); until c = ' ';
      readln(k, price);   { читаем жирность и цену }
      k := k div 5 - 2;   { получаем код – номер в массивах }
      if price < MinPrice[k] then begin
        MinPrice[k] := price;
        Count[k] := 1;
      end
      else
      if price = MinPrice[k] then
        Count[k] := Count[k] + 1;
    end;
    for k:=1 to 3 do
      write(Count[k], ' ');
  end.
```

11) Определимся с данными, которые фактически влияют на результат:

- нас интересует только количество участников, их нужно разделить по классам и по баллам
- нас не интересуют имена и фамилии, поэтому при чтении их можно пропускать

Мы будем использовать два массива: в массиве **Total** будем хранить общее количество участников с разбивкой по баллам (в элементе **Total[i]** хранится количество участников, получивших ровно **i** баллов), а в двухмерном массиве **Count** – количество участников с разбивкой по баллам и классам, то есть, **Count[i,j]** хранит количество участников из класса **j**, которые получили ровно **i** баллов[1]:

```
  const MAX = 70;
  var Total: array[0..MAX] of integer;
      Count: array[0..MAX,7..11] of integer;
```

В начале программы оба массива нужно обнулить[2].

```
    for ball:=0 to MAX do begin
      Total[ball] := 0;
      for class:=7 to 11 do Count[ball,class] := 0;
    end;
```

Таким образом, «скелет» программы можно записать так:

```
  const MAX = 70;
  var Count: array[0..MAX,7..11] of integer;
      Total: array[0..MAX] of integer;
      N: integer;
      c: char;
      i, j, class, ball, minBall, Sum: integer;
  begin
```

[1] Вообще говоря, без массива **Total** можно обойтись, потому что **Total[i]** – это сумма **i**-ой строки матрицы **Count**. Но его использование сильно упрощает дело при обработке данных. На досуге вы можете написать программу без него.

[2] На практике это не обязательно, потому что глобальные переменные и массивы обнуляются автоматически во всех известных автору версиях Паскаля. Тем не менее, на экзамене вы должны показать эксперту, что вы понимаете суть дела.

```
    Readln(N);
    for ball:=0 to MAX do begin
      Total[ball] := 0;
      for class:=7 to 11 do Count[ball,class] := 0;
    end;
    for i:=1 to N do begin
      { пропустить фамилию и имя }
      { прочитать класс и баллы }
      { увеличить счетчики }
    end;
    { определить <=25% призеров и их минимальный балл }
    { если получилось <25%, проверить следующих }
    { вывести минимальный балл }
    { количество призеров по классам }
  end.
```

Теперь расшифруем все блоки, обозначенные комментариями. При чтении пропускаем фамилию и имя:

```
      for j:=1 to 2 do
        repeat read(c); until c = ' ';
```

затем считываем класс и баллы (**readln**, до конца строки)

```
      readln(class, ball);
```

и увеличиваем общий счетчик и счетчик для данного класса:

```
      Total[ball] := Total[ball] + 1;
      Count[ball,class] := Count[ball,class] + 1;
```

Теперь определяем всех, кто гарантированно попадает в призеры. Накапливаем количество призеров, начиная с максимально возможного количества баллов, пока сумма укладывается в 25%:

```
    Sum := 0;
    ball := MAX;
    while (Sum+Total[ball])*100 <= 25*N do begin
      Sum := Sum + Total[ball];
      if Total[ball] > 0 then minBall := ball;
      ball := ball - 1;
    end;
```

Здесь нужно обратить внимание на два момента. Во-первых, для проверки на 25% используется нестрогое неравенство, и все операции выполняются с целым числами. Во-вторых, новое значение записывается в переменную **minBall** только тогда, когда количество участников, набравших этот балл, не ноль (по условию нужно вывести минимальный балл, который был **фактически** набран).

На следующем шаге проверяем участников «на границе».

```
    if ((Sum+1)*100 <= 25*N) and (ball*2 > MAX) then
      minBall := ball;
```

Условие

(Sum+1)*100 <= 25*N

означает, что по крайней мере еще один участник «вписывается» в 25% лучших, а условие

ball*2 > MAX

говорит о том, что он набрал больше половины от максимального количества баллов.

Теперь можно вывести минимальный балл призеров:

```
    writeln(minBall);
```

Чтобы вывести количество призеров по параллелям, мы сначала для каждого суммируем количество участников, набравших от **minBall** до **MAX** баллов:

```
    for class:=7 to 11 do begin
      Sum := 0;
      for ball:=minBall to MAX do
        Sum := Sum + Count[ball,class];
      write(Sum, ' ');
    end;
```

12) Эта задача – небольшая модификация задачи 7, отличающаяся только условием отбора нужных данных. Решение можно разбить на два этапа:
- прочитать данные и запомнить имена и фамилии тех, кто прошел тестирование;
- отсортировать список по алфавиту и вывести на экран

Количество участников ограничено (не более 500), это косвенно говорит о том, что нужно использовать массив для хранения результатов. Для сортировки надо одновременно удерживать в памяти все данные, поэтому без массива символьных строк здесь не обойтись:

```
var List: array[1..500] of string;
```

Структура программы:

```
var List: array[1..500] of string;
    name, temp: string;
    c: char;
    i, j, N, ball1, ball2, ball3, count: integer;
begin
  count := 0;  { счетчик несдавших }
  readln(N);
  for i:=1 to N do begin
    { прочитать фамилию и имя }
    { прочитать баллы }
    { если не сдал, запомнить }
  end;
  { сортировка по алфавиту }
  { вывод списка }
end.
```

Расшифруем отдельные блоки, обозначенные комментариями. В цикле сначала читаем фамилию и имя очередного абитуриента и записываем их в переменную **name**:

```
name := '';
for j:=1 to 2 do
  repeat
    read(c);
    name := name + c;
  until c = ' ';
```

Далее читаем оценки в переменные **ball1**, **ball2** и **ball3**, используя оператор **readln** (чтение до конца строки).

```
readln(ball1, ball2, ball3);
```

Если абитуриент прошел тестирование, увеличиваем счетчик **count** и записываем его фамилию и имя в очередной элемент списка:

```
if (ball1 >= 30) and (ball2 >= 30) and (ball3 >= 30)
   (ball1+ball2+ball3 >= 140)then begin
  count := count + 1;
  List[count] := name;
end;
```

Предполагая, что коды русских букв стоят по алфавиту, после ввода данных применим сортировку, например, так:

```
for i:=1 to count-1 do
  for j:=i to count do
    if List[i] > List[j] then begin
      temp := List[i];
      List[i] := List[j];
      List[j] := temp;
    end;
```

Заметьте, что в сортировке участвуют не все 500 элементов массива **List**, а только **count** – столько абитуриентов прошли тестирование. Остается вывести список на экран:

```
  for i:=1 to count do writeln(List[i]);
```

Вот полная программа:

```
var List: array[1..500] of string;
    name, temp: string;
    c: char;
    i, j, N, ball1, ball2, ball3, count: integer;
begin
   count := 0;
   readln(N);
   for i:=1 to N do begin
     name := '';
     for j:=1 to 2 do
       repeat
         read(c);
         name := name + c;
       until c = ' ';
     readln(ball1, ball2, ball3);
     if (ball1 >= 30) and (ball2 >= 30) and
        (ball3 >= 30) and
        (ball1+ball2+ball3 >= 140) then begin
       count := count + 1;
       List[count] := name;
     end;
  end;
  for i:=1 to count-1 do
    for j:=i to count do
      if List[i] > List[j] then begin
         temp := List[i];
         List[i] := List[j];
         List[j] := temp;
       end;
  for i:=1 to count do writeln(List[i]);
end.
```

13) Эта задача – полный аналог задачи 10. Прежде всего, нужно понять, что «АЗС» определяется сочетанием «Фирма + Улица». Каждая АЗС может продавать бензин разных сортов, каждому сорту соответствует своя строчка в исходных данных. Важно, что нам **НЕ** нужно запоминать ни фирму, ни улицу, поэтому при чтении их можно вообще пропускать.

Фактически это задача на поиск количества минимальных элементов в потоке данных, причем нужно отдельно работать с тремя наборами данных (бензин разных марок).

Введем массивы из трех элементов для хранения минимальной цены и количества магазинов, продающих по этой цене:

```
var MinPrice, Count: array[1..3] of integer;
```

Для поиска минимальных элементов нужно записать начальные значения: в каждый элемент массива **MinPrice** – любую цену, больше максимально возможной, а все счетчики обнулить.

```
for i:=1 to 3 do begin
  MinPrice[i] := 3001;   { любое число > 3000 }
  Count[i] := 0;         { обнулить счетчики }
end;
```

Дальше возникает следующий вопрос: как, прочитав из файла марку бензина **k**, рассчитать номер соответствующего элемента массива (код):

92 →1, 95 →2, 98 →3

Так же, как и в задаче 10, замечаем, что при увеличении **k** на 3 код увеличивается на 1, то есть, мы получаем линейную зависимость с коэффициентом 1/3. Свободный член находим из условия **92 div 3 + b = 1**, что дает **b = –29**, так что

код = k div 3 - 29

Если в какой-то задаче числа совсем «нескладные» и не удается вывести формулу, можно использовать оператор выбора (**case**) или серию условных операторов. Никаких других хитростей в программе нет[3]:

```
program gasoline;
var MinPrice, Count: array[1..3] of integer;
    N: integer;
    c: char;
    i, j, k, price: integer;
begin
  Readln(N);
  for i:=1 to 3 do begin
    MinPrice[i] := 3001;
    Count[i] := 0;
  end;
  for i:=1 to N do begin
    for j:=1 to 2 do     { пропускаем фирму и улицу }
      repeat read(c); until c = ' ';
    readln(k, price);    { читаем марку бензина и цену }
    k := k div 3 - 29;   { получаем код – номер в массивах }
    if price < MinPrice[k] then begin
      MinPrice[k] := price;
      Count[k] := 1;
    end
    else
    if price = MinPrice[k] then
      Count[k] := Count[k] + 1;
  end;
  for k:=1 to 3 do
    write(Count[k], ' ');
end.
```

[3] Решение, предложенное в проекте демо-варианта ФИПИ 2010 года, содержит массивы, описанные как **array[92..98]**, что само по себе очень неграмотно.

14) В этой задаче нужно подсчитать, сколько раз встречается каждая буква. Если из букв можно составить палиндром, то одна буква (центральная) может встречаться нечетное число раз, а остальные – обязательно четное.

Для подсчета количества букв (в английском языке всего 26 букв) можно использовать массив

```
var count: array[1..26] of integer;
```

Но более интересно использовать красивую возможность, когда в качестве индексов используются сами символы:

```
var count: array['A'..'Z'] of integer;
```

Перед началом работы нужно заполнить его нулями (ни одного символа еще не получено):

```
for c:='A' to 'Z' do count[c] := 0;
```

Ввод символов (до точки) естественно делать в цикле **while**:

```
read(c);
while c <> '.' do begin
  count[c] := count[c] + 1;
  read(c);
end;
```

Обратите внимание, что в такая конструкция 1) правильно обрабатывает ситуацию, когда первый символ – это точка; 2) не теряет символ, стоящий перед точкой. А вот эти два варианта – неправильные (разберитесь, почему?):

```
repeat
  read(c);
  count[c]:=count[c]+1;
until c = '.';
```

```
while c <> '.' do begin
  count[c]:=count[c]+1;
  read(c);
end;
```

Теперь считаем, сколько символов встречаются нечетное число раз. Здесь **nOdd** – целая переменная, а **cOdd** – символьная переменная, куда мы записываем центральный символ.

```
nOdd := 0;
for c:='A' to 'Z' do
  if count[c] mod 2 = 1 then begin
    cOdd := c;
    Inc(nOdd);
  end;
```

Если нашли нечетное количество таких символов, то палиндром составить нельзя:

```
if nOdd > 1 then
   writeln('Нет')
else begin
   writeln('Да');
  { можно составить! }
end;
```

Остается разобраться, как вывести палиндром в алфавитном порядке. Сначала проходим весь массив **count** и выводим каждую букву в «половинном» количестве (вторая половина будем справа от центра!):

```
for c:='A' to 'Z' do
  for i:=1 to count[c] div 2 do
    write(c);
```

Обратите внимание, что буква, стоящая по центру, тут тоже может появиться, если она встречается более одного раза.

Затем выводим центральный символ, если он есть:

```
if nOdd = 1 then write(cOdd);
```

и оставшийся «хвост», уже в обратном порядке, от **'Z'** до **'A'**:

```
for c:='Z' downto 'A' do
  for i:=1 to count[c] div 2 do
    write(c);
```

Вот полная программа:

```
var count: array['A'..'Z'] of integer;
    i, nOdd: integer;
    c, cOdd: char;
begin
  for c:='A' to 'Z' do count[c] := 0;
  read(c);
  while c <> '.' do begin
    count[c]:= count[c] + 1;
    read(c);
  end;
  nOdd := 0;
  for c:='A' to 'Z' do
    if count[c] mod 2 = 1 then begin
      cOdd := c;
      Inc(nOdd);
    end;
  if nOdd > 1 then
     writeln('Нет')
  else begin
     writeln('Да');
     for c:='A' to 'Z' do
       for i:=1 to count[c] div 2 do
         write(c);
     if nOdd = 1 then write(cOdd);
     for c:='Z' downto 'A' do
       for i:=1 to count[c] div 2 do
         write(c);
  end;
end.
```

Альтернативное решение предложила **Кочешкова А.С.** Отличие от приведённого выше варианта состоит в том, что строка-результат строится в памяти и в конце программы выводится на экран. При проходе по массиву счётчиков от 'A' до 'Z' новые буквы вставляются в середину строки **res** следующим образом:

```
Insert(c, res, ((length(res) div 2)+1));
```

Здесь **c** – вставляемый символ, а выражение **(length(res) div 2)+1)** вычисляет место первого символа после середины строки.

Вот полная программа:

```
var i, oddCount:integer;
    res:string;
    count:array ['A'..'Z'] of integer;
    c, cOdd: char;
begin
  oddCount:= 0;  { количество непарных букв }
  res:= ''; { строка-результат }
  for c:='A' to 'Z' do count[c]:=0; { обнуление массива }
```

```
    { считываем символы до точки }
    read(c);
    while c <> '.' do begin
      Inc(count[c]); { увеличиваем счётчик }
      read(c);
    end;
    { строим результат в строке res }
    for c:='A' to 'Z' do
      if count[c] mod 2 = 0 then begin
        for i:=1 to count[c] do
          Insert( c,res,((length(res) div 2)+1));
      end
      else begin { нашли непарную букву }
        Inc(oddCount);
        cOdd:= c { запомнили непарную букву }
      end;
    { вывод результата }
    if oddCount > 1 then writeln('Нет')
    else begin
      writeln ('Да');
      if oddCount = 1 then { непарную букву ставим в середину }
        for i:=1 to count[cOdd] do
          Insert(cOdd,res,(length(res) div 2)+1);
      writeln(res);
    end;
  end.
```

15) Для решения задачи нужно ответить на ряд вопросов:

Какие данные нужно хранить?
Какие структуры данных применить (простые переменные, массив, запись и т.п.)?
Как читать данные?
Какую обработку можно выполнить прямо при чтении?
Какую обработку нужно выполнить после чтения всех данных?
Как выводить результаты?

По условию нас интересует только фамилия, имя и сумма баллов, поэтому отдельные баллы, полученные по каждому из видов многоборья, мы хранить не будем.

В условии сказано, что количество спортсменов не более 1000. Фактически, это явное указание на то, что нужно сначала прочитать данные всех спортсменов в массив, а потом делать окончательную обработку. Удобно использовать массив записей такого типа:

```
type TInfo = record
       name: string[33];
       sum: integer;
     end;
```

Поле **name** хранит имя и фамилию как одну символьную строку, ее длина равна сумме максимальных длин имени и фамилии (12 + 20) плюс 1 символ на пробел между ними. Второе поле – сумма баллов, ее мы будем считать прямо во время чтения данных. Уже можно написать начало программы:

```
var Info: array[1..1000] of TInfo;
    M, N, i, j, ball: integer;
    c: char;
begin
```

```
    readln(N); { число спортсменов }
    readln(M); { число видов многоборья }
    for i:=1 to N do begin
      Info[i].name := '';
      for j:=1 to 2 do  { читаем два блока: фамилию и имя }
        repeat
          read(c);
          Info[i].name := Info[i].name + c;
        until c = ' ';
      { здесь нужно читать баллы и суммировать их }
    end;
    { сортировка массива }
    { вывод таблицы результатов }
  end.
```

Чтение и суммирование баллов по отдельным видам спорта (их всего **M**) выполняем в цикле:

```
  Info[i].sum := 0;
  for j:=1 to M do begin
    read(ball);
    Info[i].sum := Info[i].sum + ball;
  end;
```

При сортировке массива нам потребуется переставлять структуры типа **TInfo**, поэтому нужно объявить вспомогательную структуру:

```
  var temp: TInfo;
```

Для сортировки можно использовать любой метод, например, классический «метод пузырька»:

```
  for i:=1 to N-1 do
    for j:=N-1 downto i do
      if Info[j].sum < Info[j+1].sum then begin
         temp := Info[j];
         Info[j] := Info[j+1];
         Info[j+1] := temp;
      end;
```

Осталось решить вопрос о выводе данных. Итак, список спортсменов отсортирован по убыванию суммы баллов, но места не расставлены. Сложность в том, что несколько спортсменов могут набрать одинаковую сумму, при этом они должны получить одно и то же место.

Сделаем вывод места следующим образом. Введем целую переменную **mesto**. Очевидно, что тот, кто стоит первым в списке, занял первое место (запишем в переменную **mesto** значение 1). Теперь в цикле рассмотрим всех спортсменов, стоящих в списке под номерами от 1 до **N**. Если номер очередного спортсмена больше 1 и его сумма баллов меньше сумме балов предыдущего, то увеличиваем переменную **mesto** на 1. Затем выводим фамилию и имя, сумму баллов и **mesto**.

```
  mesto := 1;
  for i:=1 to N do begin
    if (i > 1) and (Info[i].sum < Info[i-1].sum) then
      mesto := mesto + 1;
    writeln(Info[i].name, ' ', Info[i].sum, ' ', mesto);
  end;
```

Вот вся программа целиком:

```
type TInfo = record
       name: string[33];
       sum: integer;
     end;
var Info: array[1..1000] of TInfo;
    M, N, i, j, ball, mesto: integer;
    c: char;
    temp: TInfo;
begin
  readln(N); { число спортсменов }
  readln(M); { число видов многоборья }
  for i:=1 to N do begin
    Info[i].name := '';
    for j:=1 to 2 do  { читаем два блока: фамилию и имя }
      repeat
        read(c);
        Info[i].name := Info[i].name + c;
      until c = ' ';
    { читаем баллы и суммируем их }
    Info[i].sum := 0;
    for j:=1 to M do begin
      read(ball);
      Info[i].sum := Info[i].sum + ball;
    end;
  end;
  { сортировка массива }
  for i:=1 to N-1 do
    for j:=N-1 downto i do
      if Info[j].sum < Info[j+1].sum then begin
        temp := Info[j];
        Info[j] := Info[j+1];
        Info[j+1] := temp;
      end;
  { вывод таблицы результатов }
  mesto := 1;
  for i:=1 to N do begin
    if (i > 1) and (Info[i].sum < Info[i-1].sum) then
      mesto := mesto + 1;
    writeln(Info[i].name, ' ', Info[i].sum, ' ', mesto);
  end;
end.
```

16) В этой задаче используются данные типа «время», которые вводятся в символьном виде. Работать с ними в таком формате (например, сравнивать) неудобно, потому нужно переводить время в числовую форму, например, в число минут от 00:00. Так время 09:45 преобразуется в число 60*9+45=585.

Поскольку эта операция выполняется неоднократно в разных местах программы (сначала ввод текущего времени в первой строке, а потом – ввод времени освобождения ячейки для каждого пассажира), удобно написать функцию, которая преобразует символьную строку в формате hh:mm (hh обозначает часы, а mm – минуты) в целое число так, как рассказано выше. Вот пример такой функции:

```
function Time2Int(sTime: string): integer;
var h, m, code0: integer;
begin
  code0 := Ord('0');
  h := 10*(Ord(sTime[1])-code0) + (Ord(sTime[2])-code0);
  m := 10*(Ord(sTime[4])-code0) + (Ord(sTime[5])-code0);
  Time2Int := 60*h + m;
end;
```

Здесь в переменную **code0** мы записываем код символа '0', чтобы не вычислять его повторно.

В условии сказано, что число пассажиров в списке не превышает 1000, это явное указание на то, что нужно прочитать данные в массив записей примерно такой структуры:

```
type TInfo = record
       name: string[20]; { фамилия }
       time: integer;    { время освобождения ячейки }
     end;
```

Сам массив мы объявим так:

```
var Info: array[1..1000] of TInfo;
```

Сложность заключается в том, что нам нужно записывать в массив информацию только о тех пассажирах, для которых время освобождения ячейки не больше, чем **curTime+120**, где **curTime** – текущее время. Все остальные строки нужно игнорировать. Это значит, что требуется ввести счетчик **count** (целую переменную) , в которой мы будем хранить количество «хороших» пассажиров, которые освободят свои ячейки не более, чем через 2 часа (120 минут). Получается такой цикл ввода:

```
count := 0;
for i:=1 to N do begin
  ... { здесь ввести данные в Info[count+1] }
  if Info[count+1].time <= curTime+120 then
     count := count + 1;
end;
```

Иначе говоря, мы вводим данные в первый неиспользованный элемент массива **Info**, а к следующему переходим только тогда, когда очередной пассажир «хороший» и его данные нужно сохранить.

Как вводить данные? Хотя все официальные рекомендации по решению задачи С4 основаны на посимвольном вводе данных, многие профессионалы предпочитают сначала прочитать всю очередную строку в символьную переменную **s**, а потом «разбирать» ее в памяти. В данном случае такой подход позволяет значительно упростить программу, и мы его применим (для разнообразия).

Будем вводить строку s целиком, искать пробел и делить ее на две части (слева от пробела – фамилия, справа – время). Затем время преобразуем в целое число с помощью уже написанной функции **Time2Int**:

```
for i:=1 to N do begin
  readln(s);
  p := Pos(' ', s);
  Info[count+1].name := Copy(s,1,p-1);
  Info[count+1].time := Time2Int(Copy(s,p+1,Length(s)-p));
  if Info[count+1].time <= curTime+120 then
     count := count + 1;
end;
```

Теперь остается только отсортировать массив и вывести список фамилий в нужном порядке. Важно не забыть, что нужно сортировать не **N** элементов, а **count** (именно столько мы нашли «хороших» пассажиров):

```
for i:=1 to count do
  for j:=count-1 downto i do
    if Info[j].time > Info[j+1].time then begin
      temp := Info[j];
      Info[j] := Info[j+1];
      Info[j+1] := temp;
    end;
```

Вот полная программа:

```
type TInfo = record
       name: string[20];
       time: integer;
     end;
var Info: array[1..1000] of TInfo;
    s: string;
    N, p, i, j, count, curTime: integer;
    c: char;
    temp: TInfo;
{ функция для преобразования времени в число }
function Time2Int(sTime: string): integer;
var h, m, code0: integer;
begin
  code0 := Ord('0');
  h := 10*(Ord(sTime[1])-code0) + (Ord(sTime[2])-code0);
  m := 10*(Ord(sTime[4])-code0) + (Ord(sTime[5])-code0);
  Time2Int := 60*h + m;
end;
{-----------------------------------------}
begin
  readln(s);
  curTime := Time2Int(s);
  readln(N);
  count := 0;
  { ввод данных о пассажирах }
  for i:=1 to N do begin
    readln(s);
    p := Pos(' ', s);
    Info[count+1].name := Copy(s,1,p-1);
    Info[count+1].time := Time2Int(Copy(s,p+1,Length(s)-p));
    if Info[count+1].time <= curTime+120 then
       count := count + 1;
  end;
  { сортировка массива }
  for i:=1 to count do
    for j:=count-1 downto i do
      if Info[j].time > Info[j+1].time then begin
        temp := Info[j];
        Info[j] := Info[j+1];
        Info[j+1] := temp;
      end;
```

```
  { вывод списка }
  for i:=1 to count do
    writeln(Info[i].name);
end.
```

17) условия становится ясно, что задача решается в два этапа:

I. прочитать символы до точки и определить длину самого короткого слова из латинских букв (обозначим ее **minLen**);

II. сделать «сдвиг» кодов латинских букв на **minLen** влево.

Начнем с первого. Простое посимвольное чтение строки **s** до первой встреченной точки выглядит так (здесь **c** – переменная типа **char**):

```
s := ''; { пустая строка }
repeat
  read(c);     { прочитали символ }
  s := s + c; { добавили в конец строки }
until c = '.';
```

При этом нам нужно еще определить длину самого короткого слова с учетом того, что между словами может быть сколько угодно символов-разделителей (разных!). Введем переменную **len**, которая будет определять длину текущего (очередного, вводимого в данный момент) слова.

Как определить, что прочитанный символ – латинская буква? Конечно, можно использовать условный оператор со сложным условием:

```
if (('a' <= c) and (c <= 'z')) or
   (('A' <= c) and (c <= 'Z')) then ...
```

Более красиво это можно сделать с помощью оператора in, который проверяет, входит ли элемент во множество:

```
if c in ['a'..'z', 'A'..'Z'] then ...
```

Здесь множество в квадратных скобках содержит два интервала: от 'a' до 'z' и от 'A' до 'Z'.

Если очередной прочитанный символ – латинская буква, нужно увеличить **len** на единицу (слово продолжается). Если же это не латинская буква, то слово закончилось, так как встречен символ-разделитель . Если в переменной **len** ненулевое значение, нужно сравнить эту длину с минимальной и, если прочитанное слово короче всех предыдущих, записать его длину в **minLen**. Таким образом, цикл ввода выглядит так:

```
s := '';
minLen := 201; { любое число > 200 }
len := 0;
repeat
  read(c);
  s := s + c;
  if c in['a'..'z','A'..'Z'] then
    len := len + 1
  else begin
    if (len > 0) and (len < minLen) then
       minLen := len;
    len := 0;
  end;
until c = '.';
```

Теперь нужно в цикле пройти всю прочитанную строку и «сдвинуть» каждый символ (точнее, его код) вправо на **minLen**:

```
for i:=1 to Length(s) do
    if s[i] in ['a'..'z','A'..'Z'] then begin
      code := Ord(s[i]);  { старый код }
      k := code - minLen; { новый код }
      s[i] := Chr(k);
    end;
```

Однако такое решение не учитывает цикличность: например, при сдвиге буквы **'A'** на 2 символа влево мы не получим **'Y'**. Поэтому после изменения кода нужно проверить, не вышел ли он за допустимые границы (диапазона латинских букв), а если вышел, то добавить к полученному коду 26 (число латинских букв), что обеспечит циклический сдвиг:

```
k := code - minLen; { новый код }
{ цикличность }
if s[i] in ['a'..'z'] then
  if k < Ord('a') then k := k + 26;
if s[i] in ['A'..'Z'] then
  if k < Ord('A') then k := k + 26;
```

Вот полная программа:

```
var c: char;
    s: string;
    len, minLen, code, i, k: integer;
begin
  s := '';
  minLen := 201; { любое число > 200 }
  len := 0;
  { чтение данных }
  repeat
    read(c);
    s := s + c;
    if c in['a'..'z','A'..'Z'] then
      len := len + 1
    else begin
      if (len > 0) and (len < minLen) then
         minLen := len;
      len := 0;
    end;
  until c = '.';
  { сдвиг кодов на minLen влево }
  for i:=1 to Length(s) do
    if s[i] in ['a'..'z','A'..'Z'] then begin
      code := Ord(s[i]);  { старый код }
      k := code - minLen; { новый код }
      { цикличность }
      if s[i] in ['a'..'z'] then
         if k < Ord('a') then k := k + 26;
      if s[i] in ['A'..'Z'] then
         if k < Ord('A') then k := k + 26;
      { запись нового кода }
      s[i] := Chr(k);
```

```
      end;
    writeln(s);
  end.
```

18) В условии очень важна последняя строчка: «количество голосов избирателей в исходном списке может быть велико (свыше 1000), а количество различных партий в этом списке не превосходит 10». Это значит, что
    - нельзя хранить в массиве все прочитанные записи
    - можно (и нужно) хранить в массиве названия партий, их не больше 10.

    Таким образом, нужно выделить массив строк:

```
const MAX = 10;
var Names: array[1..MAX] of string;
```

    Поскольку нужно считать, сколько голосов получила каждая партия, нужно выделить массив счетчиков такого же размера:

```
var count: array[1..MAX] of integer;
```

    Кроме того, нужна переменная **nParties**, в которой мы будем хранить количество найденных различных партий (в начале программы в нее нужно записать 0).

```
var nParties: integer;
```

    В начале программы во все счетчики обычно записывают 0. Однако можно поступить хитрее. Когда мы нашли запись с новой партией, в счетчик нужно сразу добавить 1. А это можно сделать заранее, записав все начальные значения счетчиков, равные 1:

```
for i:=1 to MAX do count[i]:=1;
```

    Теперь алгоритм можно записать так

    1. прочитать количество голосовавших **N**
    2. в цикле **N** раз
       а) прочитать название партии
       б) искать ее среди уже известных партий – в массиве **Names**
       в) если партия найдена, то увеличить ее счетчик голосов
          иначе увеличить счетчик партий **nParties** и записать новое название в массив, в элемент **Names[nParties]**
    3. отсортировать массив count по **убыванию** (только первые **nParties** элементов, потому что партий может быть меньше 10), одновременно с перестановкой в массиве **count** нужно не забывать переставлять соответствующие элементы массива **Names**, например, так:

```
for i:=1 to nParties-1 do
  for j:=nParties-1 downto i do
    if count[j] < count[j+1] then begin
      k:=count[j]; count[j]:=count[j+1]; count[j+1]:=k;
      s:=Names[j]; Names[j]:=Names[j+1]; Names[j+1]:=s;
    end;
```

       здесь **k** – вспомогательная целая переменная, а **s** – вспомогательная символьная строка.
    4. вывести элементы массива **Names** с 1-ого по **nParties**.

```
for i:=1 to nParties do writeln(Names[i]);
```

Осталось разобраться, как искать прочитанное название партии в массиве **Names**. Предположим, что мы прочитали очередную строку в переменную s:

```
Readln(s);
```

Сначала переменной j присваиваем значение 1 (начать с первого элемента массива **Names**). Затем в цикле **while** увеличиваем **j**, пока не просмотрим все партии (**j** станет больше, чем **nParties**) или не найдем строку **s**:

```
j := 1;
while (j <= nParties) and
      (Names[j] <> s) do j:=j+1;
```

Если после цикла **j<=nParties**, значит, мы нашли название партии в массиве и нужно увеличить ее счетчик. Если нет – увеличиваем **nParties** и сохраняем ее имя в очередном элементе массива **Names**:

```
if j <= nParties then
  count[j]:=count[j]+1
else begin
  nParties:=nParties+1;
  Names[nParties]:=s;
end;
```

Вот программа целиком

```
const MAX=10;
var Names: array[1..MAX] of string;
    count: array[1..MAX] of integer;
    N, i, j, k, nParties: integer;
    s: string;
begin
  nParties := 0; { еще нет партий }
    { начальные значения счетчиков = 1 }
  for i:=1 to MAX do count[i]:=1;
  Readln(N); { количество людей }
  for i:=1 to N do begin
    Readln(s); { читаем название партии }
      { ищем его в списке }
    j := 1;
    while (j <= nParties) and
          (Names[j] <> s) do j:=j+1;
      { если нашли – увеличили счетчик голосов }
    if j <= nParties then
      count[j]:=count[j]+1
      { не нашли – добавили в список }
    else begin
      nParties:=nParties+1;
      Names[j]:=s;
    end;
  end;
    { сортировка массива count по убыванию }
  for i:=1 to nParties-1 do
    for j:=nParties-1 downto i do
      if count[j] < count[j+1] then begin
        k:=count[j]; count[j]:=count[j+1]; count[j+1]:=k;
```

```
          s:=Names[j]; Names[j]:=Names[j+1]; Names[j+1]:=s;
        end;
      { вывод отсортированного списка }
    for i:=1 to nParties do writeln(Names[i]);
  end.
```

Использование записей позволяет сделать программу немного более профессиональной. Действительно, название партии и счетчик ее голосов тесно связаны, поэтому логично объявить новый тип данных

```
  type TParty = record
         Name: string;
         count: integer;
       end;
```

и массив этих записей вместо двух отдельных массивов **Names** и **count**:

```
  var Parties: array[1..MAX] of TParty;
```

Вместо **Names[j]** и **count[j]** теперь нужно обращаться к полям записи **Parties[j].Name** и **Parties[j].count**. Упрощается и сортировка (меняем местами структуры целиком):

```
    for i:=1 to nParties-1 do
      for j:=nParties-1 downto i do
        if Parties[j].count < Parties[j+1].count then begin
          p:=Parties[j];
          Parties[j]:=Parties[j+1];
          Parties[j+1]:=p;
        end;
```

здесь **p** – вспомогательная запись, переменная типа **TParty**. Вот вся программа:

```
  const MAX=10;
  type TParty = record
         Name: string;
         count: integer;
       end;
  var Parties: array[1..MAX] of TParty;
      N, i, j, nParties: integer;
      s: string;
      p: TParty;
  begin
    nParties := 0; { еще нет партий }
      { начальные значения счетчиков = 1 }
    for i:=1 to MAX do Parties[i].count:=1;
    Readln(N); { количество людей }
    for i:=1 to N do begin
      Readln(s); { читаем название партии }
        { ищем его в списке }
      j := 1;
      while (j <= nParties) and
            (Parties[j].Name <> s) do j:=j+1;
        { если нашли - увеличили счетчик голосов }
      if j <= nParties then
        Parties[j].count:=Parties[j].count+1
        { не нашли - добавили в список }
```

```
      else begin
        nParties:=nParties+1;
        Parties[nParties].Name:=s;
      end
    end;
      { сортировка массива count по убыванию }
    for i:=1 to nParties-1 do
      for j:=nParties-1 downto i do
        if Parties[j].count < Parties[j+1].count then begin
          p:=Parties[j];
          Parties[j]:=Parties[j+1];
          Parties[j+1]:=p;
        end;
      { вывод отсортированного списка }
     for i:=1 to nParties do
       writeln(Parties[i].Name);
end.
```

19) Главная проблема в этой задаче – разобраться с датами. Нужно

- хранить дату в таком виде, чтобы ее было удобно сравнивать за один этап
- «разобрать» дату, преобразовать ее из символьного представления в тот формат, который мы выберем

Напрашивается вариант хранения даты в виде трех целых чисел (день, месяц, год). Плюс – естественность для человека. Минус – сложность сравнения двух дат (попробуйте написать условие «дата-1» < «дата-2»).

Поэтому мы пойдем другим путем. Будем хранить дату в виде одного целого числа – числа дней, прошедших (примерно) с Рождества Христова. Примерно – потому, что мы не будем «возиться» с високосными и невисокосными годами, а также не будем учитывать, что разные месяцы имеют разное число дней. Отметим, что в задачах ЕГЭ всегда предполагается, что все данные вводятся без ошибок.

Пусть нам удалось раскодировать день, месяц и год, и записать и соответственно в переменные **d**, **m** и **y**. Тогда будем вычислять дату так:

```
date := (y - 1)*366 + (m-1)*31 + d-1;
```

Понятно, что это не точное число дней, прошедших с даты 00.00.0000, но нам вполне достаточно того, что две любые реальные даты в таком формате представляются разными значениями, и их можно сравнивать просто как целые числа.

Проверим, поместится ли это число в ячейку, отводимую для хранения целого числа. По условию наибольшая дата, 31.12.2100, дает значение даты около $2100 \cdot 366 \approx 800000$. В старых версиях Паскаля целые числа занимали 2 байта и были ограничены диапазоном –32768..32767. В новых версиях, например, в PascalABC, целое число занимает 4 байта и вмещает ±2 млрд., то есть, наше число заведомо помещается. Таким образом, у вас есть два варианта:

1) написать, что вы используете новую версию, например, PascalABC (это разрешено);
2) для старых версия использовать тип **longint** вместо **integer**.

Теперь сообразим, как раскодировать день, месяц и год из символьной строки **s**, которую мы прочитаем из входного потока

```
Readln ( s );
```

Сначала нужно дойти до второго пробела и выделить фамилию с именем в строковую переменную **Name**, а все остальное (дату) оставить в **s**:

```
p := 1;   { начали с первого символа }
while s[p] <> ' ' do p := p+1; { дошли до пробела }
```

```
  p := p+1; { пропустили этот пробел }
  while s[p] <> ' ' do p := p+1; { дошли до второго пробела }
  Name := Copy(s, 1, p-1);      { выделили фамилию с именем }
  s:= Copy(s,p+1,Length(s)-p);  { здесь осталась только дата }
```

Теперь в строке **s** первые два символа – день, символы 4 и 5 – месяц, а символы 7-10 – год. Преобразуем их в целые числа с помощью стандартной процедуры **Val** (здесь **c** – вспомогательная целая переменная, в которой возвращается код ошибки, мы ее далее не используем):

```
  Val(Copy(s,1,2), d, c);
  Val(Copy(s,4,2), m, c);
  Val(Copy(s,7,4), y, c);
```

После этого остается «собрать» дату в переменную **date**:

```
  date := (y - 1)*366 + (m-1)*31 + d-1;
```

Если новая дата равна минимальной на данный момент, хранящейся в переменной **minDate**, нужно увеличить счетчик **count**:

```
      if date = minDate then count:=count+1;
```

Если новая дата меньше минимальной, нужно запомнить ее в переменной **minDate**, запомнить имя человека в переменной **minName** и записать в счетчик **count** единицу:

```
  if date < minDate then begin
    minName := Name;
    minDate := date;
    count := 1;
  end;
```

В самом начале в переменную **minDate** нужно записать очень большое число, например, **MaxInt** (максимальное целое число). Вот полная программа:

```
{ Версия для PascalABC }
var s, Name, minName: string;
    m, d, y, date, minDate: integer;
    N, i, p, c, count: integer;
begin
  minDate:=MaxInt;
    { вводим количество людей }
  Readln(N);
  for i:=1 to N do begin
    Readln(s); { вводим данные одного человека }
    p:=1;   while s[p] <> ' ' do p:=p+1;
    p:=p+1; while s[p] <> ' ' do p:=p+1;
      { выделили фамилию и имя, отделили дату }
    Name := Copy(s, 1, p-1);
    s:=Copy(s,p+1,Length(s)-p);
      { выделили день, месяц, год }
    Val(Copy(s,1,2), d, c);
    Val(Copy(s,4,2), m, c);
    Val(Copy(s,7,4), y, c);
      { построили дату }
    date := (y-1)*366 + (m-1)*31 + (d-1);
      { сравниваем дату с минимальной }
    if date = minDate then count:=count+1;
    if date < minDate then begin
      minName := Name;
      minDate := date;
      count := 1;
```

```
      end;
    end;
      { выводим результат }
    if count>1 then
        writeln(count)
    else writeln(minName);
  end.
```

Нестандартный вариант решения предложил **Е.И. Тищенко** (ОГОУ Полянская школа-интернат Рязанской обл.). Идея состоит в том, чтобы сравнивать символьные даты, не приводя их к числовому виду. Оказывается, для этого достаточно переставить символы в порядке ГГГГММДД, где ГГГГ – четырехзначный номер года, ММ – номер месяца и ДД – номер дня. То есть нужно из строки '12.09.1998' сделать строку '19980912'. Для этого можно использовать функцию **Copy**. Если в строке **s** записана дата в «стандартном виде», для такого преобразования используется оператор:

```
specDate := Copy(s,7,4)+Copy(s,4,2)+Copy(s,1,2);
```

Кроме того, для выделения имени из строки, можно использовать функцию **Pos**:

```
p := Pos(' ', s);          { ищем первый пробел }
name := Copy(s, 1, p);     { выделили фамилию с пробелом }
Delete(s, 1, p);           { удалили фамилию с пробелом из s }
p := Pos(' ', s);          { ищем второй пробел }
name := name + Copy(s, 1, p-1); { добавили имя к фамилии }
Delete(s, 1, p);           { удалили имя, осталась только дата }
```

Вот полная программа:

```
var s, Name, specDate, minName, minDate: string;
    count, i, p, N: integer;
begin
  minDate:='21001231';  { большая дата, 31.12.2100 }
    { вводим количество людей }
  Readln(N);
  for i:=1 to N do begin
    Readln(s); { вводим данные одного человека }
      { выделяем фамилию и имя в переменную Name }
    p := Pos(' ', s);
    Name := Copy(s, 1, p);
    Delete(s, 1, p);
    p := Pos(' ', s);
    Name := Name + Copy(s, 1, p-1);
    Delete(s, 1, p);
      { строим дату в специальном формате }
    specDate := Copy(s,7,4)+Copy(s,4,2)+Copy(s,1,2);
      { сравниваем дату с минимальной }
    if specDate = minDate then count := count + 1;
    if specDate < minDate then begin
      minDate := specDate;
      minName := Name;
      count := 1
    end
  end;
    { выводим результат }
  if count>1 then
      writeln(count)
  else writeln(minName);
```

```
end.
```

20) Чтение данных в этой задаче выполняется стандартно (см. разборы предыдущих задач). Важно, что нас интересуют только ученики школы № 50, всех остальных пропускаем.

Заметим, что в условии задачи содержатся лишние данные (например, указано, что номер школы находится в интервале от 1 до 99, это никак не влияет на результат). Возможно, это сделано специально с целью спровоцировать выделение массива.

Самое сложное – определить, как обрабатывать данные. Желательно обойтись без сортировки, в данном случае она совсем не нужна. Запоминать все данные тоже не нужно (и невозможно!), поскольку

- количество учеников неизвестно
- нас интересуют только фамилии двух лучших, то есть достаточно ввести всего две символьных переменных (**name1** и **name2**)

Две целых переменных **max1** и **max2** будут использоваться для хранения баллов, которые набрали два лучших ученика. Кроме того, нужно считать, сколько учеников набрали высший балл. Для этого введем счетчики **count1** и **count2**.
Предположим, что мы прочитали данные очередного ученика (и он из школы № 50!): его имя находится в переменной **name**, набранные баллы – в переменной **ball**.
Если балл ученика больше, чем **max1**, нужно запомнить нового лидера (записать его имя и балл в переменные **max1** и **name1**, в счетчик **count1** – единицу), но предварительно записать данные по старому лидеру в переменные **max2**, **name2** и **count2**:

```
if ball > max1 then begin
  max2:=max1; name2:=name1; count2:=count1;
  max1:=ball; name1:=name; count1:=1;
end
else ...
```

Если балл равен максимальному, нужно увеличить счетчик **count1** и запомнить данные нового ученика в переменных **max2** и **name2** (теперь он второй!)

```
...
else
  if ball = max1 then begin
    count1:=count1+1;
    max2:=ball; name2:=name;
  end
  else ...
```

Если балл находится между **max1** и **max2**, нужно сохранить новые данные по второму ученику (пока он один, поэтому в **count2** записываем 1):

```
...
else
  if ball > max2 then begin
    max2:=ball; name2:=name; count2:=1;
  end
  else ...
```

Если **ball** = **max2**, мы нашли еще одного ученика, имеющего второй результат, нужно его посчитать:

```
...
else
  if ball = max2 then count2:=count2+1;
```

Если **ball** < **max2**, ничего делать не нужно. Полный код обработки выглядит так:

```
if ball > max1 then begin
```

```
    max2:=max1; name2:=name1; count2:=count1;
    max1:=ball; name1:=name; count1:=1;
  end
  else
    if ball = max1 then begin
      count1:=count1+1;
      max2:=ball; name2:=name;
    end
    else
      if ball > max2 then begin
        max2:=ball; name2:=name; count2:=1;
      end
      else
        if ball = max2 then count2:=count2+1;
```

В этой задаче самая сложная часть – это приведенная выше логика обработки данных очередного ученика. Обратите внимание, что это серия *вложенных* условных операторов, каждый следующий находится в *else*-ветке предыдущего. Кроме того, условия проверяются начиная с самого «большего» (**ball** > **max1**) в порядке «уменьшения» сравниваемой величины, иначе в некоторых случаях программа будет работать неправильно.

При выводе результата нужно учесть три варианта:

1. Если **count1** = 2 или **count1+count2** = 2, выводим фамилии первых двух учеников.
2. Если **count1** =1 и **count2** > 1, выводим только фамилию и имя лидера.
3. Если **count1** > 2, выводим только **count1**.

Вот полная программа:

```
var name,name1,name2:string;
    c:char;
    i, j, N, school, ball,
    max1, max2, count1, count2: integer;
begin { начальные значения }
  max1:=-1; max2:=-1;
  count1:=0; count2:=0;
  readln(N); { читаем количество учеников }
  for i:=1 to N do begin
    { читаем фамилию имя в name }
    name := ' ' ;
    for j:=1 to 2 do
      repeat
        read(c) ; name:=name+c;
      until c=' ';
    readln(school,ball); { читаем школу и баллы }
    if school = 50 then  { только ученики школы № 50 }
      if ball > max1 then begin
        max2:=max1; name2:=name1; count2:=count1;
        max1:=ball; name1:=name; count1:=1;
      end
      else
        if ball = max1 then begin
          count1:=count1+1;
          max2:=ball; name2:=name;
        end
        else
```

```
            if ball > max2 then begin
              max2:=ball; name2:=name; count2:=1;
            end
            else
              if ball = max2 then count2:=count2+1;
    end;
    { вывод результата }
    if (count1 = 2) or
       (count1+count2 = 2) then begin
       writeln(name1);
       writeln(name2);
    end
    else
      if (count1 = 1) and (count2 > 1) then
        writeln(name1)
      else { здесь count1 > 2 }
        writeln(count1);
  end.
```

21) Нам нужно считать средний балл по каждой школе, его можно вычислить только после того, как будут прочитаны все данные. Во время чтения нужно накапливать сумму баллов по школе и количество учеников. Поэтому заведем два целочисленных массива с индексами от 1 до 99: в одном из них (назовем его **sum**) будем накапливать суммы, второй (**count**) будет массивом счетчиков учеников. Сначала эти массивы нужно обнулить.

```
var sum, count: array[1..99] of integer;
begin
for i:=1 to 99 do begin
  sum[i]:=0; count[i]:=0;
end;
```

Фамилии и имена учеников нас не интересуют, поэтому будем пропускать их при чтении. Фамилия заканчивается на первом пробеле, имя – на следующем.

```
    repeat read(c) until c=' '; { пропуск фамилии }
    repeat read(c) until c=' '; { пропуск имени }
```

Теперь читаем номер школы (в переменную **sch**) и количество баллов ученика (в переменную **ball**):

```
    readln(sch, ball);
```

Увеличиваем сумму и количество учеников по школе с номером **sch**:

```
    sum[sch] := sum[sch] + ball;
    count[sch] := count[sch] + 1;
```

После обработки всех входных строк нужно найти среднее по каждой школе и среднее по району. Найти «среднее с точностью до целых» обычно означает «применить округление до ближайшего целого», в Паскале это делает функция **round**. Проходим в цикле все школы и для тех, в которых число учеников не равно нулю, считаем среднее и записываем его в тот же массив **sum**:

```
for i:=1 to 99 do
  if count[i] > 0 then
    sum[i]:= round(sum[i]/count[i]);
```

Чтобы найти среднее по району, нам нужна общая сумма баллов. Ее можно вычислить в том же цикле (до вычисления среднего) с помощью переменной **total**:

```
total := 0;
for i:=1 to 99 do
  if count[i] > 0 then begin
    total := total + sum[i];
    sum[i]:= round(sum[i]/count[i]);
  end;
```

Количество учеников хранится в переменной **N** (ее значение считывается в первой строке входных данных), поэтому среднее по району вычислим так:

```
total := round(total / N);
```

Дальше остается в цикле пройти по массиву **sum** и вывести номера всех школ, для которых выполняется условие **sum[i]** > **total**.

```
var sum, count: array[1..99] of integer;
    c: char;
    i, N, sch, ball, k, total: integer;
begin
{ обнуляем массивы }
for i:=1 to 99 do begin
  sum[i]:=0; count[i]:=0;
end;
{ читаем количество учеников }
readln(N);
for i:=1 to N do begin
    { пропускаем фамилию и имя }
  repeat read(c) until c=' ';
  repeat read(c) until c=' ';
    { читаем номер школы и балл }
  readln(sch, ball);
  sum[sch] := sum[sch] + ball;
  count[sch] := count[sch] + 1;
end;
  { средний балл по району }
total := 0;
for i:=1 to 99 do
  if count[i] > 0 then begin
    total := total + sum[i];
    sum[i]:= round(sum[i]/count[i]);
  end;
total := round(total / N);
{ находим школы, где средний балл выше районного }
k := 0;
for i:=1 to 99 do
   if sum[i] > total then begin
     k := k + 1;
     ball := sum[i];
     write(i, ' ');
   end;
writeln;
if k = 1 then { если такая школа одна }
  writeln('Средний балл = ', ball);
end.
```

22) Эта задача аналогична задаче 17 в том, что для шифровки используется циклический сдвиг символов алфавита. Чтение строки можно выполнять целиком

```
readln(s);
```

учитывая, что ее длина не превышает ограничение стандартного[4] Паскаля (255 символов). Дальше в цикле рассматриваем все символы строки. Для того, чтобы определить длину слова, используем переменную **len** (счетчик символов слова) и логическую переменную (флаг) **inside**, которая показывает, что мы находимся в середине слова (текущий символ – не первый в слове).

Если очередной символ – буква английского алфавита, проверяется переменная **inside**. Если она равна **True** (это не первая буква слова), длину **len** увеличиваем на единицу. Иначе (если это начало слова) записываем в **len** единицу и устанавливаем переменной **inside** значение **True**:

```
if s[i] in ['a'..'z','A'..'Z'] then
  if inside then len := len+1
  else begin
    len := 1;
    inside := True;
  end
else …
```

В *else*-блоке нужно обработать ситуацию, когда очередная буква не входит в английский алфавит, то есть, слово закончилось. В этом случае нужно зашифровать *предыдущие* **len** символов, используя циклический сдвиг на **len** вправо (то есть, к коду символа добавляется **len**). Тут еще необходимо учесть цикличность при выходе за границы алфавита: если код символа становится больше кода буквы **Z** (или **z**, для строчных букв), нужно вычесть из него 26 (длину латинского алфавита).

```
...
else
  if inside then begin
    inside := False;
    for j:=1 to len do begin
      k := Ord(s[i-j]) + len; { сдвиг кода }
      if s[i-j] in ['a'..'z'] then
         if k > Ord('z') then k := k - 26;
      if s[i-j] in ['A'..'Z'] then
         if k > Ord('Z') then k := k - 26;
      s[i-j] := Chr(k);
    end;
  end;
```

Дадим некоторые пояснения к циклу. Сейчас рассматривается символ **s[i]**, на нем закончилось слово длиной **len**. Переменная **j** меняется от 1 до **len**, в теле цикла обрабатывается символ **s[i-j]**, то есть, меняются все символы от **s[i-1]** до **s[i-len]** включительно. Вот полная программа:

```
var s: string;
    inside: boolean;
    i, j, k, len: integer;
begin
  readln(s); { читаем строку }
```

[4] Это ограничение снято в Delphi.

```
    inside := False;
     { цикл по всем символам строки }
    for i:=1 to length(s) do begin
        { если латинская буква }
      if s[i] in ['a'..'z','A'..'Z'] then
        if inside then
          len := len+1 { не первая буква слова }
        else begin     { первая буква слова }
          len := 1;
          inside := True;
        end
      else { если не латинская буква }
        if inside then begin { закончилось слово }
          inside := False;
            { шифруем len символов «назад» }
          for j:=1 to len do begin
            k := Ord(s[i-j]) + len; { сдвиг кода }
              { цикличность при выходе за границы }
            if s[i-j] in ['a'..'z'] then
               if k > Ord('z') then k := k - 26;
            if s[i-j] in ['A'..'Z'] then
               if k > Ord('Z') then k := k - 26;
            s[i-j] := Chr(k); { новый символ }
          end;
        end;
    end;
      { вывод результата }
    writeln(s);
  end.
```

Интересно рассмотреть обобщение этой задачи на случай, когда входная строка может быть больше 255 символов, то есть, читать ее сразу нельзя. Придется читать по одному символу. Прочитав английскую букву, добавляем ее в конец символьной переменной **word**, в которой хранится текущее слово. Если прочитан символ, не являющийся английской буквой, смотрим на переменную **word**. Если она не пустая (хранит английское слово), перекодируем его с помощью циклического сдвига на **length(word)** и выводим слово на экран, после этого выводим сам прочитанный символ. Цикл останавливается, когда прочитан символ #:

```
  repeat
    read(c);
    ...
  until c = '#';
```

Программа оказывается не сложнее, чем при ограничении на длину строки:

```
  var word: string;
      c: char;
      i, k, len: integer;
  begin
    word := '';
    repeat
      read(c); { чтение символа }
      if c in ['a'..'z','A'..'Z'] then
        word := word + c
      else begin
```

```
        if word <> '' then begin
          len := Length(word);
            { обрабатываем слово }
          for i:=1 to len do begin
            k := Ord(word[i]) + len;
            if word[i] in ['a'..'z'] then
               if k > Ord('z') then k := k - 26;
            if word[i] in ['A'..'Z'] then
               if k > Ord('Z') then k := k - 26;
            word[i] := Chr(k);
          end;
           { вывод слова без перехода на новую строку}
          write(word);
          word := ''; { пустая строка }
        end;
        write(c); { вывод символа после слова }
      end;
    until c = '#';
    writeln;
  end.
```

Другое решение предложила **О.В. Алимова**. Так как длина строки не превышает 255 символов, сразу считываем всю строку:

```
  readln(s);
```

Далее в цикле перебираем все символы с 1 до символа «#» Для каждого символа проверяем, является ли он буквой латинского алфавита, т.е. входит ли в очередное слово. Если текущий символ буква, то увеличиваем **word_len** - счетчик количества букв в слове. Иначе переменная position указывает на первый символ после конца очередного слова, **word_len** равен длине слова, и можно приступать к обработке этого слова. Следует заметить, что **word_len** может равнятся 0, в том случае, когда подряд идут символы, не являющиеся английскими буквами. После перебора всех символов в строке останется напечатать зашифрованную строку.

```
  word_len := 0;
  for position:=1 to pos('#',s) do
     if s[position] in ['a'..'z','A'..'Z'] then
          word_len := word_len + 1
     else begin...
          { очередное слово закончилось }
          { шифруем буквы этого слова   }
          end;
  writeln(s);
```

Для шифрования букв текущего слова необходимо от текущей позиции вернуться по строке на **word_len** символов назад. В этот момент точно известно, что все символы в строке в интервале от **s[position-word_len]** до **s[position-1]**, являются буквами латинского алфавита. Остается определить является буква строчной или прописной, и заменить этот сивол на зашифрованный. После этого обнулить **word_len** для обработки следующего слова в строке.

```
  for dist:= 1 to word_len do
    if s[position - dist] in ['A'..'Z'] then
      {Шифруем прописные буквы}
    else
      {Шифруем строчные буквы}
  word_len := 0;
```

Для шифрования буквы, надо помнить, что все буквы (отдельно строчные и отдельно прописные) находятся в таблице ASCII друг за другом, и в латинском алфавите 26 букв.

1. Находим на сколько код текущей буквы отличается от кода буквы «A».

```
Ord(s[position - dist]) - Ord('A')
```

2. Увеличиваем это отличие на длину текущего слова

```
Ord(s[position - dist]) - Ord('A') + word_len
```

3. Если в результате получилось значение больше 26 (т.е. вылезли из интервала «A..Z»), то необходимо учесть цикличность.

```
(Ord(s[position - dist]) - Ord('A') + word_len) mod 26
```

4. Осталось к коду бувы «A» добавить полученное значение и взять символ с этим кодом.

```
s[position - dist] :=
  chr(Ord('A') +
  (Ord(s[position - dist]) - Ord('A') + word_len)mod 26)
```

Для шифрования строчных символов все то-же самое, только используем символ «a».

```
var s: string;
    word_len, dist, position: integer;
begin
  readln(s);
  word_len := 0;
  for position := 1 to pos('#',s) do
    if s[position] in ['a'..'z', 'A'..'Z'] then
     word_len := word_len + 1
    else
    begin
     for dist := word_len downto 1 do
       if s[position - dist] in ['A'..'Z'] then
         s[position - dist] := chr(Ord('A') +
          (Ord(s[position - dist]) - Ord('A') +
           word_len)mod 26)
       else
         s[position - dist] := chr(Ord('a') +
         (Ord(s[position - dist]) - Ord('a')
             + word_len)mod 26);
      word_len := 0;
   end;
  writeln(s);
end.
```

Это решение удалось еще сократить (**Д.Ф. Муфаззалов**, Уфа, УГАТУ) с помощью использования функции **upcase**, которая преобразует буквенные символы к верхнему регистру (строчную букву делает прописной):

```
var
  s: string;
  word_len, i, position: integer;
begin
  readln(s);
  word_len := 0;
  for position := 1 to pos('#', s) do
    if upcase(s[position]) in ['A'..'Z'] then
        inc(word_len)
    else begin
```

```
        for i:=position-word_len to position-1 do begin
          s[i] := chr(Ord(s[i]) + word_len);
          if upcase(s[i]) > 'Z' then
              s[i] := chr(ord(s[i]) - 26);
        end;
        word_len := 0;
      end;
    writeln(s);
  end.
```

23) Для решения задачи нужно сначала прочитать все данные. Нас не интересуют фамилии, имена и номера школ, нужно только знать, сколько участников набрали определенное число баллов. Для этого заведем массив счетчиков **count** с индексами в интервале от 0 до 100, так что **count[i]** будет обозначать число участников экзамена, набравших **i** баллов. В начале массив нужно обнулить, а затем прочитать все данные:

```
{ обнуляем массив }
for i:=0 to 100 do count[i]:=0;
{ читаем количество строк}
readln(N);
for i:=1 to N do begin
  { пропускаем фамилию и имя}
  repeat read(c) until c=' ';
  repeat read(c) until c=' ';
  readln(sch, ball); { читаем номер школы и балл ученика}
  count[ball]:=count[ball]+1;
end;
```

Определяем 20% от всех участников:

```
M := N div 5;
```

и пытаемся найти такой балл **i**, что **M** участников получили балл не меньше **i**:

```
s:=0; i:=100;
while s < M do begin
  s:=s+count[i];
  i:=i-1;
end;
i := i + 1;
```

или так

```
s:=0; i:=101;
while s < M do begin
  i:=i-1;
  s:=s+count[i];
end;
```

Если после этого **s** = **M**, то ровно **M** участников получили балл **i** и выше, им ставится отличная отметка. Кроме того, «отлично» ставится в том случае, когда более 20% учеников набрали высший балл. Это значит, что **s** = **count[i]**, то есть, сумма s в самом деле состоит из одного слагаемого. Поэтому

```
if (s = M) or (s = count[i]) then writeln(i)
else ...
```

Если ни один из этих вариантов не подошел, то участникам, набравшим **i** баллов, не будет поставлена отметка «отлично». Но вывести на экран **i-1** в виде результата нельзя, потому что может быть так, что **i-1** балл никто не набрал. Поэтому ищем первый балл, больший **i**, который набрал хотя бы один человек:

```
else begin
  repeat
    i:=i+1;
  until count[i]<>0;
  writeln(i);
```

```
end;
```

Вот полная программа:

```
var count:array[0..100] of integer;
    c: char;
    i, N, sch, ball, M, s: integer;
begin
{ обнуляем массив }
for i:=0 to 100 do count[i]:=0;
{ читаем количество строк}
readln(N);
for i:=1 to N do begin
  { пропускаем фамилию и имя}
  repeat read(c) until c=' ';
  repeat read(c) until c=' ';
  readln(sch,ball); { читаем номер школы и балл ученика}
  count[ball]:=count[ball]+1;
end;
M := N div 5; {вычисляем 20% от количества учеников}
  { ищем минимальный балл этих 20% }
s := 0;
i:=101;
while s < M do begin
  i:=i-1;
  s:=s+count[i];
end;
  { вывод результата }
if (s = M) or (s = count[i]) then
   { i баллов – «отлично» }
  writeln (i)
else begin
   { i баллов – не «отлично» }
  repeat
    i:=i+1;
  until count[i]<>0;
  writeln(i);
end;
end.
```

24) Во-первых, нас интересуют только цифры, остальные символы можно просто пропускать. Во-вторых, предполагается, что текст может быть достаточно длинный, так что читать его в массив (или строку) не нужно – в этом случае получится неоптимальная программа. Фактически для ответа на вопрос задачи нам нужно знать только *количество* цифр, поэтому достаточно выделить в памяти массив счетчиков с индексами от 0 до 9 и считать цифры:

```
num: array[0..9] of integer;
```

Кроме того, нам нужно узнать, есть ли в строке хотя бы одна цифра (чтобы вывести ответ «Да» или «Нет»). Для этого во время чтения будем считать найденные цифры в переменной **count**. В начале программы счетчики нужно обнулить:

```
for i:=0 to 9 do num[i] := 0;
count := 0;
```

При чтении проверяем очередной символ так: если его код находится между кодом цифры ‘0’ (код 48) и цифры ‘9’ (код 57), то это цифра. Так можно делать, потому что цифры во всех кодовых страницах стоят последовательно, одна за другой. Вот полный цикл чтения до точки:

```
repeat
  read(c);
  if ('0' <= c) and (c <= '9') then begin
    k := Ord(c) - Ord('0');
    num[k] := num[k] + 1;
    count := count + 1;
  end;
until c = '.';
```

Здесь используется цикл **repeat**, потому что хотя бы один символ необходимо прочитать. Чтобы найти номер нужного счетчика **k**, вычисляем разность между кодами прочитанного символа и цифры '0'.

Чтобы при выводе получилось максимальное число, нужно сначала выводить все девятки (их количество записано в **num[9]**), затем все восьмерки и т.д. до нуля:

```
for i:=9 downto 0 do
  for k:=1 to num[i] do write(i);
```

Здесь используется оператор **write**, который не переходит на новую строку и выводит все цифры в одной строке.

Осталась одна проблема: если в строке есть только нули, нужно вывести только один ноль. Состояние «цифры есть, но все они – нули» записывается в виде условия

```
if (count = num[0]) and (num[0] > 0) then
   num[0] := 1;
```

в этом случае в счетчик нулей записываем единицу. После этого будет выведен только один нуль.

Вот полная программа:

```
program qq;
var c: char;
    num: array[0..9] of integer;
    i, k, count: integer;
begin
    { обнулить счетчики }
  for i:=0 to 9 do num[i] := 0;
  count := 0;
    { читаем последовательность }
  repeat
    read(c);
      { если цифра, увеличили счетчик }
    if ('0' <= c) and (c <= '9') then begin
      k := Ord(c) - Ord('0');
      num[k] := num[k] + 1;
      count := count + 1;
    end;
  until c = '.';
     { если только нули }
  if (count = num[0]) and (num[0] > 0) then
```

```
      num[0] := 1;
    { вывод результата }
  if count = 0 then
     writeln('Нет')
  else writeln('Да');
  for i:=9 downto 0 do
    for k:=1 to num[i] do write(i);
end.
```

Возможно еще одно аналогичное решение, где индексы у массива – символьные:

```
var c, i: char;
    num: array['0'..'9'] of integer;
    k, count: integer;
begin
    { обнулить счетчики }
  for i:='0' to '9' do num[i] := 0;
  count := 0;
  repeat
    read(c);
      { если цифра, увеличили счетчик }
    if ('0' <= c) and (c <= '9') then begin
      num[c] := num[c] + 1;
      count := count + 1;
    end;
  until c = '.';
     { если только нули }
  if (count = num['0']) and (num['0'] > 0) then
     num['0'] := 1;
    { вывод результата }
  if count = 0 then
     writeln('Нет')
  else writeln('Да');
  for i:='9' downto '0' do
    for k:=1 to num[i] do write(i);
end.
```

Ещё одно интересное решение (автор идеи – ***Д. Тоджибаев***) позволяет обойтись вообще без массивов счётчиков. Идея состоит в том, чтобы собрать все цифры в новую символьную строку, затем отсортировать их по убыванию внутри этой строки и вывести полученную строку на экран. Особым образом нужно обрабатывать случай, когда в исходной строке из цифр встречаются только нули: признаком этого будет первый символ получившейся цифровой строки, содержащй 0. В этом случае программа выводит один ноль.

Нужно только учитывать два момента: 1) в классических версиях Паскаля строка не превышает 255 символов (выход – писать на *FreePascal* или *Delphi*, где есть «длинные» строки); 2) при большой длине строки сортировка символов в строке может считаться неэффективным решением с точки зрения количества операций.

Вот полное решение:

```
var digits, s: string;
    i, j, L: integer;
    temp: char;
```

```
begin
 read(s);
   { собираем цифры в строку digits }
 digits:= '';
 for i:= 1 to length(s) do
   if s[i] in ['0'..'9'] then begin
     if s[i] <> '0' then nonZero:= nonZero+1;
     digits:= digits + s[i]
   end;
 L:= length(digits);
   { выводим результат }
 if L = 0 then
   writeln('Нет')
 else begin
   writeln('Да');
       { сортируем символы строки }
   for i:= 1 to L-1 do
    for j:= L-1 downto i do
      if digits [j]< digits[j+1] then begin
        temp:= digits[j];
        digits[j]:= digits[j+1];
        digits[j+1]:= temp;
      end;
   if digits[1] = '0' then  { только нули }
        writeln(0)
   else writeln(digits);
 end
end.
```

25) В этой задаче, как и в предыдущей, нас тоже интересует только количество разных цифр, поэтому при вводе будем считать их с помощью массива.

Затем нужно отсортировать массив по возрастанию. Но при этом возникает проблема – мы потеряем информацию о том, какой цифре соответствует, например, счетчик **num[0]**. Поэтому заведем другой массив **ind**, в котором будем хранить цифры, соответствующие счетчикам. Например, если в **ind[2]** находится число 5, то **num[2]** – это количество цифр '5'. В начале программы в **i**-й элемент массива **ind** записываем число **i** (цифры стоят по порядку):

```
for i:=0 to 9 do ind[i] := i;
```

После чтения данных нужно отсортировать цифры по возрастанию частоты встречаемости, причем при одинаковой частоте встречаемости порядок должен сохраниться. Этим свойством обладает, например, метод сортировки «пузырьком»:

```
for i:=0 to 8 do
  for j:=8 downto i do
    if num[j] > num[j+1] then begin
      k:=num[j]; num[j]:=num[j+1]; num[j+1]:=k;
      k:=ind[j]; ind[j]:=ind[j+1]; ind[j+1]:=k;
    end;
```

Обратите внимание, что при перестановке элементов массива **num** также переставляются и соответствующие элементы массива **ind**, чтобы сохранить информацию о том, каким цифрам соответствуют счетчики массива **num**.

Выводить нужно только те цифры, для которых счетчик больше нуля, причем саму цифру берем из массива **ind**:

```
for i:=0 to 9 do
  if num[i] > 0 then write(ind[i]);
```

Вот полная программа:

```
var c: char;
    num, ind: array[0..9] of integer;
    i, j, k: integer;
begin
    { начальные значения }
  for i:=0 to 9 do begin
    num[i] := 0;
    ind[i] := i;
  end;
    { ввод данных, подсчет цифр в массиве num }
  repeat
    read(c);
    if ('0' <= c) and (c <= '9') then begin
      k := Ord(c) - Ord('0');
      num[k] := num[k] + 1;
    end;
  until c = '.';
    { сортировка «пузырьком» }
  for i:=0 to 8 do
    for j:=8 downto i do
      if num[j] > num[j+1] then begin
        k:=num[j]; num[j]:=num[j+1]; num[j+1]:=k;
        k:=ind[j]; ind[j]:=ind[j+1]; ind[j+1]:=k;
      end;
    { вывод результата }
  for i:=0 to 9 do
    if num[i] > 0 then write(ind[i]);
end.
```

Еще одно решение этой задачи предложил **А. Тарасов** (с. Красноусольский, Республика Башкортостан):

```
var c:char;
    num, ind:array[0..9] of integer;
    i, j, k:integer;
begin
    { начальные значения }
  for i:=0 to 9 do begin
    num[i] := 0;
    ind[i] := i;
  end;
    { ввод данных, подсчет цифр в массиве num }
  read(c);
  while c <> '.' do begin
    k := Ord(c) - Ord('0');
    num[k] := num[k] + 1;
    read(c);
  end;
```

```
    { сортировка «пузырьком» }
  for i:=0 to 8 do
   for j:=8 downto i do
    if num[ind[j]] > num[ind[j+1]] then begin
       k:=ind[j]; ind[j]:=ind[j+1]; ind[j+1]:=k
    end;
    { вывод результата }
  for i:=0 to 9 do
   if num[ind[i]]<>0 then write(ind[i])
end.
```

В чем отличие? Во-первых, цикл ввода изменен так, чтобы избавиться от условного оператора. Мы знаем, что по условию кроме цифр и точки никаких других символов быть не может, поэтому отдельно нужно обработать только точку. Для этого цикл с постусловием **repeat** заменен на цикл с предусловием **while**, оператор **read** перенесен в конец цикла, но перед циклом пришлось поставить еще один оператор **read** (иначе на первом проходе в переменной **c** будет неопределенное значение).

Во-вторых, здесь применяется сортировка по индексам. Элементы массива **ind** представляют собой уже не цифры, а индексы элементов массива **num** в отсортированном порядке. Поэтому при сортировке можно переставлять только индексы, а элементы массива **num** не трогать. За такую оптимизацию приходится расплачиваться менее понятной записью: к элементу отсортированного массива num приходится обращаться с помощью двойной адресации типа **num[ind[i]]**, что означает «значение элемента массива **num**, номер которого находится в элементе массива **ind** с номером **i**». Это стандартный профессиональный прием, но для его использования нужно хорошо понимать, что вы делаете.

Ещё одно интересное решение задачи предложил **Е.И. Тищенко** (ОГОУ Полянская школа-интернат Рязанской обл.):

```
var num: array['0'..'9'] of integer;
    i, min: integer;
    c, iMin: char;
begin
  for c:='0' to '9' do num[c] := 0;
    { ввод данных, подсчет цифр в массиве num }
  repeat
    read(c);
    if ('0' <= c) and (c <= '9') then
      num[c] := num[c] + 1;
  until c = '.';
    { вывод результата совместно с сортировкой }
  for i:=1 to 10 do begin
    min := maxInt;
    for c:='0' to '9' do
      if (num[c] > 0) and (num[c] < min) then begin
        min:=num[c];
        iMin:=c
      end;
    if min = maxInt then break;
    write(iMin);
    num[iMin]:=0;
  end;
```

```
end.
```

Во-первых, здесь использован массив с символьными индексами (это можно делать в Паскале), что позволило упростить цикл ввода данных: получив очередную цифру, не требуется специально вычислять индекс нужного элемента массива.

Во-вторых, фактически сортировка совмещена с выводом результата, при этом второй массив не нужен вообще. Всего может быть не более 10 цифр, поэтому организуем цикл из 10 шагов:

```
for i:=1 to 10 do begin
  { ищем индекс iMin минимального ненулевого элемента в num }
  { если не нашли, то выйти из цикла }
  write(iMin);
  num[iMin]:=0;
end;
```

На каждом шаге цикла ищем минимальный ненулевой элемент в массиве **num**. Если такой элемент не найден (все оставшиеся элементы нулевые), выходим из цикла с помощью оператора **break**. Если такой элемент найден, выводим на экран его символьный индекс **iMin** и обнуляем соответствующий элемент массива **num[iMin]**, чтобы не «захватить» его при следующем проходе.

Для поиска минимального элемента используем стандартный алгоритм с запоминанием значения минимального элемента и его номера, причем в начале в переменную **min** записывается наибольшее целое число **maxInt**:

```
min := maxInt;
for c:='0' to '9' do
  if (num[c] > 0) and (num[c] < min) then begin
    min:=num[c];
    iMin:=c
  end;
```

Если в переменной **min** осталось это значение **maxInt**, все элементы массива **num** нулевые и нужно выйти из цикла:

```
if min = maxInt then break;
```

26) Как и в двух предыдущих задачах, нас интересует только количество разных цифр. Поэтому ввод данных и подсчет цифр выполняется так же, как и в задаче 24 (с помощью массива **num**). Разница только в том, что здесь индексы массива начинаются не с нуля, а с 1 (ноль нам не нужен, это признак окончания ввода):

```
for i:=1 to 9 do num[i] := 0;
repeat
  read(c);
  if ('1' <= c) and (c <= '9') then begin
    k := Ord(c) - Ord('0');
    num[k] := num[k] + 1;
  end;
until c = '0';
```

Главный вопрос – как найти число-палиндром, и притом минимальное из всех возможных? Палиндром можно разбить на три части – два «крыла», симметричных относительно вертикальной оси, и центральная часть – она может состоять из одного символа или вообще отсутствовать:

Палиндром	Левое крыло	Центр	Правое крыло
1234554321	12345	–	54321
12345654321	12345	6	54321

Если в левую часть входит некоторая цифра, то она входит и в правую часть, поэтому «крылья» образованы **парами** цифр. Причем для того, чтобы число было минимальным, числа в левом крыле должны увеличиваться, а в правом – уменьшаться.

Удобнее всего ввести две символьных переменных **sL** и **sR**, которые будут обозначать соответственно левое и правое крылья. Сначала запишем в них пустые строки:

```
sL := '';
sR := '';
```

Затем просматриваем все цифры с младшей до старшей, чтобы числа в левом крыле увеличивались. Пока для очередного числа **i** есть пара (счетчик **num[i]>1**), приписываем эту цифру в конец левого крыла и в начало правого, после этого уменьшаем счетчик **num[i]** на 2 (пара цифр израсходована):

```
for i:=1 to 9 do
  while num[i] > 1 do begin
    c := Chr(i+Ord('0'));
    sL := sL + c;
    sR := c + sR;
    num[i] := num[i] - 2;
  end;
```

После этого парных цифр уже нет. Какой выбрать центральную цифру? Чтобы число было минимальным, она должна быть минимальной. Поэтому ищем с помощью цикла **while** первую цифру, счетчик которой больше нуля.

```
i := 1;
while (i < 10) and (num[i] = 0) do
  i := i + 1;
```

Если такая обнаружена, ставим ее в центр, то есть приписываем в конец левого крыла (вариант – в начало правого):

```
if i < 10 then sL := sL + Chr(i+Ord('0'));
```

Остается вывести результат – сцепить строки **sL** и **sR**:

```
  writeln(sL + sR);
```

Вот полная программа:

```
var c: char;
    num: array[1..9] of integer;
    i, k, count: integer;
    sL, sR: string;
begin
    { ввод данных, подсчет количества цифр }
  for i:=1 to 9 do num[i] := 0;
  repeat
    read(c);
    if ('1' <= c) and (c <= '9') then begin
      k := Ord(c) - Ord('0');
      num[k] := num[k] + 1;
    end;
```

```
    until c = '0';
      { строим левую sL и правую sR части }
    sL := '';
    sR := '';
    for i:=1 to 9 do
      while num[i] > 1 do begin
        c := Chr(i+Ord('0'));
        sL := sL + c;
        sR := c + sR;
        num[i] := num[i] - 2;
      end;
      { центральная цифра }
    i := 1;
    while (i < 10) and (num[i] = 0) do
      i := i + 1;
    if i < 10 then sL := sL + Chr(i+Ord('0'));
      { вывод результата }
    writeln(sL + sR);
  end.
```

27) Задача очень похода на предыдущую, но вместо цифр используются символы английского алфавита. Значит, заводим массив для всех символов, удобнее всего – с символьным индексом:

```
num: array['A'..'Z'] of integer;
```

При чтении нужно все строчные буквы преобразовать в соответствующие прописные. Для этого из кода символа нужно вычесть разницу кодов букв «a» и «A»:

```
if ('a' <= c) and (c <= 'z') then
  c := Chr(Ord(c) - Ord('a') + Ord('A'));
```

Для того, чтобы в конце программы определить, все ли символы использованы, мы будем считать все английские буквы с помощью счетчика **count**. Полный цикл ввода получается такой:

```
count := 0;
repeat
  read(c);
  if ('a' <= c) and (c <= 'z') then
    c := Chr(Ord(c) - Ord('a') + Ord('A'));
  if ('A' <= c) and (c <= 'Z') then begin
    num[c] := num[c] + 1;
    count := count + 1;
  end;
until c = '.';
```

Поскольку нужно *последнее* по алфавиту слово, поиск пар нужно выполнять от 'Z' до 'A':

```
for i:='Z' downto 'A' do
  while num[i] > 1 do begin
    sL := sL + i;
    sR := i + sR;
    num[i] := num[i] - 2;
  end;
```

Центральный символ должен быть максимальным, поэтому ищем его, начиная с 'Z':

```
i := 'Z';
while (i >= 'A') and (num[i] = 0) do
  i := Pred(i);
if i >= 'A' then sL := sL + i;
```

Здесь функция **Pred** возвращает предыдущее значение для порядкового типа, в данном случае – предыдущий символ в кодовой таблице, имеющий код на единицу меньше. Есть также и функция **Succ**, возвращающая следующее порядковое значение.

Для того, чтобы определить, все ли символы задействованы, мы «сложим» левую и правую части результата и сравним длину получившейся строки со счетчиком **count**. Если они отличаются или ни одного нужного символа не найдено, строку-палиндром составить нельзя.

```
s := sL + sR;
if (Length(s) = count) and (count > 0) then begin
  writeln('Да');
  writeln(s);
end
else
  writeln('Нет');
```

Вот полная программа:

```
var c, i: char;
    num: array['A'..'Z'] of integer;
    k, count: integer;
    s, sL, sR: string;
begin
    { обнуление счетчиков }
  for i:='A' to 'Z' do num[i] := 0;
    { ввод данных, подсчет количества символов }
  count := 0;
  repeat
    read(c);
      { строчные -> прописные }
    if ('a' <= c) and (c <= 'z') then
      c := Chr(Ord(c) - Ord('a') + Ord('A'));
    if ('A' <= c) and (c <= 'Z') then begin
      num[c] := num[c] + 1;
      count := count + 1;
    end;
  until c = '.';
    { строим левую sL и правую sR части }
  sL := '';
  sR := '';
  for i:='Z' downto 'A' do
    while num[i] > 1 do begin
      sL := sL + i;
      sR := i + sR;
      num[i] := num[i] - 2;
    end;
    { центральный символ }
  i := 'Z';
  while (i >= 'A') and (num[i] = 0) do
    i := Pred(i);
```

```
    if i >= 'A' then sL := sL + i;
      { вывод результата }
    s := sL + sR;
    if (Length(s) = count) and (count > 0) then begin
      writeln('Да');
      writeln(s);
    end
    else
      writeln('Нет');
end.
```

28) Эта задача фактически представляет собой вариант задач 26-27. В отличие от задачи 26,

- цепочка заканчивается не точкой, а нулем
- нужно составить палиндром из ВСЕХ найденных цифр, или сказать, что это невозможно (аналогично задаче 27)
- нужно вывести максимальный, а не минимальный палиндром

В отличие от задачи 27, используются не буквы, а цифры. Но есть ещё одна особенность, на которую обратил внимание автора И. Титков: поскольку по условию ведущих нулей в числе быть не должно, то в ответ на входные строки

`0000000.` или `1000000.`

программа должна выдать ответ '**NO**' (палиндром без ведущих нулей составить нельзя). Как «отловить» эти варианты? В первом случае все цифры – нули, поэтому справедливо условие

num['0'] = count

Во втором случае есть одна ненулевая цифра, а остальные – нули, поэтому выполняется условие

num['0'] = count-1

Чтобы обеспечить вывод ответа '**NO**', достаточно в этих случаях записать в **num['0']** единицу, тогда условие **Length(s) = count** в блоке вывода окажется ложно. Таким образом, перед построением левой и правой частей палиндрома нужно добавить оператор (поправку предложил И. Титков):

```
if (num['0']=count) or (num['0']=count-1) then
    num['0']:=1;
```

Приведем полное решение:

```
var c: char;
    num: array['0'..'9'] of integer;
    k, count: integer;
    s, sL, sR: string;
begin
    { ввод данных, подсчет количества цифр }
  for c:='0' to '9' do num[c] := 0;
  count := 0;
  repeat
    read(c);
    if ('0' <= c) and (c <= '9') then begin
      num[c] := num[c] + 1;
```

```
        count := count + 1;
      end;
    until c = '.';
      { строим левую sL и правую sR части }
    sL := '';
    sR := '';
    if (num['0']=count) or (num['0']=count-1) then
      num['0']:=1;
    for c:='9' downto '0' do
      while num[c] > 1 do begin
        sL := sL + c;
        sR := c + sR;
        num[c] := num[c] - 2;
      end;
      { центральная цифра }
    c := '9';
    while (c >= '0') and (num[c] = 0) do
      c := Pred(c);
    if c >= '0' then sL := sL + c;
      { вывод результата }
    s := sL + sR;
    if (Length(s) = count) and (count > 0) then begin
      writeln('YES');
      writeln(s);
    end
    else
      writeln('NO');
  end.
```

29) В этой задаче нужно считать, сколько раз встречается во входной последовательности каждая цифра. Для этого используется массив счетчиков

```
var num: array['1'..'9'] of integer;
```

Этот массив сначала заполняем нулями:

```
for c:='1' to '9' do num[c] := 0;
```

Тогда цикл ввода данных получается почти такой же, как в предыдущей задаче:

```
repeat
  read(c);
  if ('1' <= c) and (c <= '9') then
    num[c] := num[c] + 1;
until c = '.';
```

В принципе, нас не интересует число цифр, поэтому вместо увеличения **num[c]** на единицу можно просто записывать туда 1

```
num[c]:= 1;
```

или даже сделать логический массив вместо целочисленного (оставим это в качестве упражнения).

Теперь остается вывести все цифры (начиная с младшей), для которых элемент массива **num** равен нулю. Для того, чтобы обработать случай, когда все цифры 1..9 есть во входных данных нет, введем счетчик цифр **count**, который будем увеличивать на 1, получив ненулевую цифру.

```
count:=0;
for c:='1' to '9' do begin
  if num[c] = 0 then write(c)
```

```
    else count := count + 1;
  end;
```

После этого цикла нужно проверить счетчик **count**: если он равен 9, выводим «0» (все цифры встретились во входных данных). Приведем полное решение (И. Титков):

```
var num: array['1'..'9'] of integer;
 c: char;
 count: integer;
begin
  for c:='1' to '9' do num[c] := 0;
  repeat
    read(c);
    if ('1' <= c) and (c <= '9') then
      num[c]:= 1;
  until c = '.';
  count := 0;
  for c:='1' to '9' do begin
    if num[c] = 0 then write(c)
    else count := count + 1;
  end;
  if count = 9 then writeln('0');
end.
```

30) В решении этой задачи можно выделить несколько ключевых моментов:

1. Выбор удобного способа хранения данных.
2. Ввод данных и предварительная обработка.
3. Сортировка в нужном порядке.
4. Вывод результата.

Информации о каждой ячейке включает ее номер и дату сдачи багажа. Для облегчения расчетов мы сразу представим дату как день от начала года (как это сделать – обсудим после). Эти данные можно объединить в структуру (запись) следующего вида:

```
type TCell = record no, day: integer; end;
```

Поле **no** – это номер ячейки, а поле **day** – день от начала года, когда в ячейку положен багаж.

Теперь подумаем, как из символьной строки вида `'01.06'` получить день от начала года. Для этого построим функцию **dayFromStart**, которая принимает символьную строку такого формата и возвращает целое число – день от начала года:

```
function dayFromStart(s: sring):integer;
begin
  ...
end;
```

В этой функции сначала выделим день и месяц в отдельные переменные **d** и **m**, используя стандартную процедуру **Val**. Номер дня записан в первых двух символах (их можно выделить как **Copy(s,1,2)**), а номер месяца – в 4 и 5-м символах (**Copy(s,4,2)**):

```
  Val(Copy(s,1,2), d, i);
  Val(Copy(s,4,2), m, i);
```

Вспомогательная переменная **i** может быть использована для обнаружения ошибок преобразования, но мы будем считать, что все данные корректны.

Сейчас в переменной d записан номер дня от начала месяца. Чтобы получить номер дня от начала года, нужно добавить к этому значению количество дней во всех предыдущих месяцах, от **1** до **m-1**:

```
  for i:=1 to m-1 do d:=d+dm[i];
```

Здесь массив dm содержит количество дней в каждом месяце:

```
var dm: array[1..12] of integer;
...
dm[1]:=31; dm[2]:=28; dm[3]:=31; dm[4]:=30;
dm[5]:=31; dm[6]:=30; dm[7]:=31; dm[8]:=31;
dm[9]:=30; dm[10]:=31; dm[11]:=30; dm[12]:=31;
```

Таким образом, функция **dayFromStart** принимает следующий вид:

```
function dayFromStart(s: string): integer;
var d, m, i: integer;
begin
  Val(Copy(s,1,2), d, i);
  Val(Copy(s,4,2), m, i);
  for i:=1 to m-1 do d:=d+dm[i];
  dayFromStart := d;
end;
```

Ввод данных. Сначала вводим текущую дату и сразу преобразуем ее в день от начала года (целая переменная **curDay**):

```
var curDay: integer;
...
readln(s);
curDay := dayFromStart(s);
```

Затем вводим количество занятых ячеек

```
  readln(N);
```

Далее в цикле от 1 до **N** вводим данные по каждой ячейке, записывая их во вспомогательную переменную **c** типа **TCell**:

```
var c: TCell;
...
readln(s);                      { читаем очередную строку }
p := Pos(' ', s);               { ищем пробел }
Val(Copy(s,1,p-1), c.no, k);    { номер слева от пробела }
Delete(s, 1, p);                { в "s" остается дата }
c.day := dayFromStart(s);
```

Все данные о ячейках, в которых багаж хранится более 3 дней (то есть **curDay-c.day>3**), нужно записать в отдельный массив (поскольку потом нужно будет их отсортировать!):

```
var cells: array[1..1000] of TCell;
```

Чтобы считать количество «отобранных» ячеек, введем счетчик **count**. Тогда цикл ввода данных будет выглядеть так:

```
count := 0;
for i:=1 to N do begin
  readln(s);                      { читаем очередную строку }
  p := Pos(' ', s);               { ищем пробел }
  Val(Copy(s,1,p-1), c.no, k);    { номер слева от пробела }
  Delete(s, 1, p);                { в "s" остается дата }
  c.day := dayFromStart(s);
  if curDay - c.day > 3 then begin
    count := count + 1;
    cells[count] := c;
  end;
end;
```

После этого сортируем структуры по возрастанию поля **day** любым методом, например, методом пузырька:

```
for i:=1 to count-1 do
  for j:=count-1 downto 1 do
    if cells[j].day > cells[j+1].day then begin
      c:=cells[j]; cells[j]:=cells[j+1];
      cells[j+1]:=c;
    end;
```

Обратите внимание, что сортировать нужно только первые **count** структур, а не **N**!

Остается только вывести номера ячеек (поле **no** каждой структуры):

```
for i:=1 to count do
    writeln(cells[i].no);
```

Вот полная программа:

```
type TCell = record
       no, day: integer;
     end;
var dm: array[1..12] of integer;
    s: string;
    i, j, N, count, curDay, p, k: integer;
    c: TCell;
    cells: array[1..1000] of TCell;
{ функция для вычисления дня от начала года }
function dayFromStart(s: string): integer;
var d, m, i: integer;
begin
  Val(Copy(s,1,2), d, i);
  Val(Copy(s,4,2), m, i);
  for i:=1 to m-1 do d:=d+dm[i];
  dayFromStart := d;
end;
{ основная программа }
begin
  dm[1]:=31; dm[2]:=28; dm[3]:=31; dm[4]:=30;
  dm[5]:=31; dm[6]:=30; dm[7]:=31; dm[8]:=31;
  dm[9]:=30; dm[10]:=31; dm[11]:=30; dm[12]:=31;
  readln(s); curDay := dayFromStart(s); { ввод даты }
  readln(N); { ввод количества ячеек }
  { ввод данных по всем ячейкам }
  count:= 0;
  for i:=1 to N do begin
    readln(s);                    { читаем очередную строку }
    p := Pos(' ', s);             { ищем пробел }
    Val(Copy(s,1,p-1), c.no, k);  { номер слева от пробела }
    Delete(s, 1, p);              { в "s" остается дата }
    c.day := dayFromStart(s);
    if curDay - c.day > 3 then begin
      count := count + 1;
      cells[count] := c;
    end;
  end;
  { сортировка по возрастанию поля day }
  for i:=1 to count-1 do
    for j:=count-1 downto 1 do
      if cells[j].day > cells[j+1].day then begin
        c:=cells[j]; cells[j]:=cells[j+1];
```

```
          cells[j+1]:=c;
        end;
    { вывод результата }
    for i:=1 to count do
      writeln(cells[i].no);
  end.
```

Отметим, что большинство сред для программирования на языке Паскаль допускает более красивое объявление массива **dm** как массива целых констант:

```
const dm: array[1..12] of integer =
          (31,28,31,30,31,30,31,31,30,31,30,31);
```

31) Сложность этой задачи состоит в том, что студентов с максимальной стипендией на каждом курсе может быть несколько (хоть все!). Поэтому придется хранить все входные данные в массиве. Сведения о студенте будем записывать в структуру (запись)

```
type TStud = record
       name: string;
       kurs, stip: integer;
     end;
```

В поле **name** хранятся фамилия и имя, в поле **kurs** – номер курса, в поле **stip** – размер стипендии.

Для поиска максимальной стипендии по каждому курсу введем массив счетчиков и обнулим его:

```
var maxStip: array[1..5] of integer;
...
for i:=1 to 5 do maxStip[i] := 0;
```

Из первой строки исходных данных читаем число студентов в переменную N, затем в цикле от 1 до N читаем строки с данными о студентах. Информацию записываем прямо в поля структуры. В поле name записываем все символы до второго пробела:

```
stud[i].name := '';
repeat
  read(c);
  stud[i].name := stud[i].name + c;
until c = ' ';
repeat
  read(c);
  stud[i].name := stud[i].name + c;
until c = ' ';
```

Затем читаем номер курса и размер стипендии (до конца строки, поэтому **readln**):

```
readln(stud[i].kurs, stud[i].stip);
```

После этого проверяем, не превышает ли эта стипендия максимальную на этом курсе; если превышает, обновляем максимальное значение:

```
if stud[i].stip > maxStip[stud[i].kurs] then
  maxStip[stud[i].kurs] := stud[i].stip;
```

Когда все данные прочтены, нужно вывести фамилии и имена всех студентов, стипендия которых равна максимальной на курсе. Для этого требуется вложенный цикл: во внешнем цикле меняется номер курса **k** (от 1 до 5), а во внутреннем просматривается весь массив сведений о студентах:

```
for k:=1 to 5 do begin
  writeln('Курс ', k);
  for i:=1 to N do
    if (stud[i].kurs = k) and
```

```
          (stud[i].stip = maxStip[k]) then
        writeln(stud[i].name);
  end;
```

Вот полная программа:

```
  const MAX = 100;
  type TStud = record
         name: string;
         kurs, stip: integer;
       end;
  var i, k, N: integer;
      c: char;
      maxStip: array[1..5] of integer;
      stud: array[1..MAX] of TStud;
  begin
    for i:=1 to 5 do maxStip[i] := 0;
    readln(N);
    for i:=1 to N do begin
      stud[i].name := '';
      repeat read(c); stud[i].name := stud[i].name + c;
      until c = ' ';
      repeat read(c); stud[i].name := stud[i].name + c;
      until c = ' ';
      readln(stud[i].kurs, stud[i].stip);
      if stud[i].stip > maxStip[stud[i].kurs] then
         maxStip[stud[i].kurs] := stud[i].stip;
    end;
    for k:=1 to 5 do begin
      writeln('Курс ', k);
      for i:=1 to N do
        if (stud[i].kurs = k) and
           (stud[i].stip = maxStip[k]) then
          writeln(stud[i].name);
    end;
  end.
```

Альтернативное решение задачи предложил А. Тарасов (с. Красноусольский, Республика Башкортостан). Идея состоит в том, чтобы фактически составить из студентов, имеющих максимальную стипендию на курсе, линейный список. В структуру включаем фамилию, имя и поле **next**, в котором будем хранить номер следующего в списке студента:

```
  type TStud = record
         name: string;
         next: integer;
       end;
```

Номер первого студента в списке (для каждого курса) будем хранить в массиве

```
  var first: array[1..5] of integer;
```

В самом начале в поле **next** каждой записи вносим 0 (на всякий случай, хотя это будет сделано автоматически во всех известных автору трансляторах Паскаля):

```
  for i:=1 to max do begin
    stud[i].name:='';
    stud[i].next:=0;
  end;
```

Если при чтении данных очередного студента его стипендия равна найденной ранее максимальной, записываем в его поле next номер первого студента в списке, а в массив **first** заносим номер текущего студента (он становится головой списка):

```
if stip = maxStip[kurs] then begin
  stud[i].next:= first[kurs];
  first[kurs]:= i;
end;
```

Если стипендия больше найденной ранее максимальной, запоминаем новый максимум и записываем в массив **first** номер этого студента:

```
if stip > maxStip[kurs] then begin
  maxStip[kurs]:= stip;
  first[kurs]:= i;
end;
```

Заметим, что этот студент будет последним в списке, и его поле **next** будет равно нулю. Тогда при выводе результата достаточно пройти по списку, начав с того студента, на который указывает элемент массива **first[kurs]** и перескакивая к следующему по полю **next** соответствующей записи. Цикл заканчивается, когда поле **next** очередной записи будет равно нулю:

```
for kurs:=1 to 5 do
  if maxStip[kurs] > 0 then begin
    writeln('Курс ', kurs);
    i := first[kurs];
    repeat
      writeln(stud[i].name);
      i := stud[i].next;
    until i = 0;
  end;
```

Вот полная программа:

```
type TStud = record
       name: string;
       next: integer;
     end;
var i, N, kurs, stip: integer;
    c: char;
    maxStip, first: array[1..5] of integer;
    stud: array[1..MAX] of TStud;
begin
  for i:=1 to 5 do begin
    maxStip[i] := 0; first[i] := 0;
  end;
  readln(N);
  for i:=1 to N do begin
    stud[i].name := '';
    repeat read(c); stud[i].name := stud[i].name + c;
    until c = ' ';
    repeat read(c); stud[i].name := stud[i].name + c;
    until c = ' ';
    readln(kurs, stip);
    if stip = maxStip[kurs] then begin
      stud[i].next:= first[kurs];
      first[kurs]:= i;
    end;
```

```
      if stip > maxStip[kurs] then begin
        maxStip[kurs]:= stip;
        first[kurs]:= i;
      end;
    end;
    for kurs:=1 to 5 do
      if maxStip[kurs] > 0 then begin
        writeln('Курс ', kurs);
        i := first[kurs];
        repeat
          writeln(stud[i].name);
          i := stud[i].next;
        until i = 0;
      end;
  end.
```

32) Понятно, что в этой задаче нужно ввести массив, в котором будем хранить число пассажиров на каждом перегоне (перегонов всегда на 1 меньше, чем станций):

```
const MAX = 10;
var people: array[1..MAX-1] of integer;
```

В самом начале все элементы массива должны быть равны нулю.

Не очень ясно, как проще считать пассажиров на каждом перегоне. Пусть очередной пассажир вошел на станции с номером **s1** и вышел на станции **s2**. Тогда на всех перегонах от **s1** до **s2-1** нужно увеличить соответствующие значения массива **people**, что не очень эффективно – появляется дополнительный внутренний цикл.

Значительно лучше при вводе данных считать, на сколько изменилось количество пассажиров на каждой станции. Введем соответствующий массив **Delta**:

```
var Delta: array[1..MAX] of integer;
```

Тогда, если очередной пассажир вошел на станции с номером **s1** и вышел на станции **s2**, нужно увеличить на 1 значение **Delta[s1]** и уменьшить на 1 значение **Delta[s2]**:

```
  Delta[s1]:= Delta[s1] + 1;
  Delta[s2]:= Delta[s2] - 1
```

При чтении фамилию и имя пассажира мы просто пропускаем – они нас не интересуют. Полный цикл чтения данных выглядит так:

```
for i:= 1 to P do begin
  repeat read(c) until c = ' ';
  repeat read(c) until c = ' ';
  readln(s1, s2);
  Delta[s1] := Delta[s1] + 1;
  Delta[s2] := Delta[s2] - 1
end;
```

Теперь можно заполнить массив **people**. На первом перегоне в поезде было **Delta[1]** человек, а на каждой следующей станции (с номером **i**) количество увеличивалось в сравнении с предыдущим на **Delta[i]**:

```
people[1] := Delta[1];
for i:=2 to N-1 do
  people[i]:= people[i-1] + Delta[i];
```

Теперь остается найти минимум:

```
min := P;
for i:=1 to N-1 do
  if people[i] < min then
    min := people[i];
```

и вывести на экран перегоны, где количество пассажиров равно минимальному:

```
for i:=1 to N-1 do
  if people[i] = min then
    writeln(i, '-', i+1)
```

Вот полная программа:

```
const MAX = 10;
var Delta: array[1..MAX] of integer;
    people: array[1..MAX-1] of integer;
    i, s1, s2, N, P, min: integer;
    c: char;
begin
readln(N, P);
for i:=1 to N do Delta[i] := 0;
for i:= 1 to P do begin
  repeat read(c) until c = ' ';
  repeat read(c) until c = ' ';
  readln(s1, s2);
  Delta[s1] := Delta[s1] + 1;
  Delta[s2] := Delta[s2] - 1
end;
people[1] := Delta[1];
for i:=2 to N-1 do
  people[i]:= people[i-1] + Delta[i];
min := P;
for i:=1 to N-1 do
  if people[i] < min then
    min := people[i];
for i:=1 to N-1 do
  if people[i] = min then
    writeln(i, '-', i+1)
end.
```

Возможен и другой вариант, когда массив Delta не нужен: прочитав **s1** и **s2**, мы сразу добавляем по 1 человеку на все перегоны между указанными станциями. Цикл ввода приобретает такой вид:

```
for i:= 1 to P do begin
  repeat read(c) until c = ' ';
  repeat read(c) until c = ' ';
  readln(s1, s2);
  for j:=s1 to s2-1 do
    people[j] := people[j] + 1;
end;
```

И вот полная программа:

```
const MAX = 10;
var people: array[1..MAX-1] of integer;
```

```
    i, j, s1, s2, N, P, min: integer;
    c: char;
begin
readln(N, P);
for i:=1 to N do Delta[i] := 0;
for i:= 1 to P do begin
  repeat read(c) until c = ' ';
  repeat read(c) until c = ' ';
  readln(s1, s2);
  for j:=s1 to s2-1 do
    people[j] := people[j] + 1;
end;
min := P;
for i:=1 to N-1 do
  if people[i] < min then
    min := people[i];
for i:=1 to N-1 do
  if people[i] = min then
    writeln(i, '-', i+1)
end.
```

Оба решения имеют небольшие недостатки. В первом используется дополнительный массив **Delta** (расход памяти), но все циклы простые, не вложенные. Поэтому алгоритм имеет линейную сложность – количество операций при больших N и P возрастает почти по линейной зависимости относительно обеих величин.

Во втором решении мы сэкономили память (нет массива **Delta**), однако при вводе данных получили вложенный цикл, что можно считать несколько неэффективным по скорости выполнения.

Какой вариант лучше? Как всегда, решение – это компромисс между быстродействием и расходуемой памятью. Есть надежда, что и в том, и в другом случае эксперт не будет снижать балл за неэффективность.

33) Нужно завести массивы, в одном из которых будем хранить максимальные баллы по каждой школе, а в другом – число учеников, получивших этот максимальный балл:

```
const MAX = 99;
var schMax, schCount: array[1..MAX] of integer;
```

В начале оба массива обнуляются:

```
for i:=1 to MAX do begin
  schMax[i] := 0;
  schCount[i] := 0;
end;
```

В этой задаче фамилии и имена нам не нужны, при чтении их можно пропускать. Дойдя до второго пробела, читаем номер школы и балл в целые переменные **sch** и **ball**:

```
repeat read(c) until c = ' ';
repeat read(c) until c = ' ';
readln(sch, ball);
```

Если балл текущего ученика равен максимальному, увеличиваем счетчик максимальных баллов для этой школы:

```
if ball = schMax[sch] then
```

```
    schCount[sch] := schCount[sch] + 1;
```

Если балл текущего ученика больше предыдущего максимального, записываем новый максимум и сбрасываем счетчик максимальных баллов для этой школы в единицу:

```
if ball > schMax[sch] then begin
   schMax[sch] := ball;
   schCount[sch] := 1;
end;
```

После чтения всех N строк (в цикле) нужно искать максимум в массиве **schMax**. Попутно (чтобы не делать второй цикл) ищем наибольшее значение счетчика среди всех школ, у которых наибольший балл. Сначала считаем, что лучшая школа – первая:

```
ball := schMax[1];
count := schCount[1];
```

Затем в цикле просматриваем все остальные школы. Если балл равен максимальному, ищем наибольшее значение счетчика:

```
if (schMax[i] = ball) and (schCount[i] > count) then
  count := schCount[i];
```

Если балл больше предыдущего максимального, запоминаем новое максимальное значение и новое значение счетчика:

```
if schMax[i] > ball then begin
  ball := schMax[i];
  count := schCount[i];
end;
```

В конце программы выводим номера школ, в которых и балл, и счетчик равны найденным максимальным значениям:

```
for i:=1 to MAX do
  if (schMax[i] = ball) and (schCount[i] = count) then
    writeln(i)
```

Вот полная программа:

```
const MAX = 99;
var schMax, schCount: array[1..MAX] of integer;
    i, N, sch, ball, count: integer;
    c: char;
begin
for i:=1 to MAX do begin
  schMax[i] := 0;
  schCount[i] := 0;
end;
readln(N);
for i:= 1 to N do begin
  repeat read(c) until c = ' ';
  repeat read(c) until c = ' ';
  readln(sch, ball);
  if ball = schMax[sch] then
     schCount[sch] := schCount[sch] + 1;
  if ball > schMax[sch] then begin
     schMax[sch] := ball;
     schCount[sch] := 1;
```

```
    end;
  end;
  ball := schMax[1];
  count := schCount[1];
  for i:=2 to MAX do begin
    if (schMax[i] = ball) and (schCount[i] > count) then
      count := schCount[i];
    if schMax[i] > ball then begin
      ball := schMax[i];
      count := schCount[i];
    end;
  end;
  for i:=1 to MAX do
    if (schMax[i] = ball) and (schCount[i] = count) then
      writeln(i)
  end.
```

Более простое решение предложил (**Д.Ф. Муфаззалов**, г. Уфа). Можно запоминать только максимальный балл среди всех учащихся района в отдельной переменной, а в массиве – количество учащихся, набравших такой балл в каждой школе:

```
  const MAX = 99;
  var schCount:array[1..MAX] of integer;
      maxBall, maxCount, i, N: integer;
      sch, ball, j:integer;
      c: char;
  begin
    maxBall:=0;
    readln(N);
    for i:=1 to n do begin
      for j:=1 to 2 do
        repeat read(c); until c=' ';
      readln(sch, ball);
      if ball > maxBall then begin
        maxBall:= ball;
       for j:=1 to MAX do schCount[j]:=0;
  {если найден новый максимум, то количество учеников с таким
  баллом во всех школах обнуляется}
      end;
      if ball = maxBall then
        Inc(schCount[sch]);
    end;
    maxCount:= schCount[1];
    for i:=2 to MAX do
      if schCount[i] > maxCount then
        maxCount:= schCount[i];
    for i:=1 to MAX do
      if schCount[i] = maxCount then writeln(i);
  end.
```

34) Особенность этой задачи состоит в том, что здесь жестко прописано, что вводить данные нужно из файла, а выводить результаты – в тот же файл, дописывая их в конец файла. Тут помогут стандартные функции для работы с файлами: **Assign** (связать указатель на файл

с файлом на диске), **Reset** (открыть файл на чтение), **Rewrite** (открыть файл на запись), **Append** (открыть файл на добавление в конец) и **Close** (закрыть файл). Схема работы с файлами в данной программе выглядит так (считаем, что файл называется **eq.txt**):

```
var F: Text; { указатель на текстовый файл }
    s: string;
    a, b, c: real; { коэффициенты уравнения }
    D: real;       { дискриминант }
    x1, x2: real;  { корни уравнения }
    i, r: integer; { вспомогательные переменные }
begin
  Assign(F, 'eq.txt');
  Reset(F);
  readln(F, s); { читаем первую строку файла }
  Close(F);
    { обработка данных, запись корней уравнения в x1 и x2 }
  Append(F);
  writeln(F, x1);  { вывод результатов в конец файла }
  writeln(F, x2);
  Close(F);
end.
```

Обратите внимание на три момента:

- после завершения файловой операции (чтения, записи или добавления) файл нужно закрывать вызовом **Close**;
- после вызова **Close** мы можем снова использовать тот же указатель **F**;
- вызывать второй раз **Assign** не нужно, поскольку мы записываем в тот же файл.

Вторая проблема, которую часто не замечают, считая эту задачу слишком легкой. В условии не сказано, что коэффициенты уравнения перечисляются от старшего к младшему. То есть, строго говоря, уравнение $2x^2-4x-6=0$ может быть задано 6 способами:

2a-4b-6	-4b+2a-6	-6+2a-4b
2a-6-4b	-4b-6+2a	-6-4b+2a

Поэтому примитивный алгоритм (найти символ «a», слева от него – старший коэффициент и т.д.) не срабатывает. Тогда как искать коэффициенты?

Понятно, что нужно проходить вдоль строки, начиная с первого символа, и выделять очередной коэффициент. Рассмотрим алгоритм выделения первого (по порядку) коэффициента из строки **s**. Число заканчивается там, где встречается один из стоп-символов: '**a**', '**b**', '**+**' или '**-**'. Для удобства добавим в конец строки символ '**!**' – он нужен для того, чтобы остановиться в том случае, когда последний коэффициент – это свободный член уравнения, и за ним строка кончается.

```
s:=s+'!';
i:=1;
while i <= Length(s) do begin
    { обработка символа s[i] }
  i:=i+1;
end;
```

Предположим, что мы нашли символ 'a', слева от него стоит коэффициент при x^2. Его можно раскодировать в вещественную переменную **a**:

```
if s[i] = 'a' then begin
  if i = 1 then a := 1          { нет коэффициента => a=1}
  else
  if (i = 2) and (s[1] in ['+','-'] then
    if s[1] = '+' then a := 1   { знак => a=1 или a=-1}
    else a := -1
  else
  Val(Copy(s,1,i-1), a, r); { есть коэффициент }
  s:=Copy(s, i+1, Length(s)-i);  { удалить из строки }
  i := 0;                        { начать с начала }
end;
```

Поясним этот фрагмент. Если символ 'a' – первый в строке, значит коэффициент равен 1, этот вариант мы учитываем с помощью условного оператора. Если это второй символ, то перед ним может стоять знак «**+**» или «**–**», в этом случае сработает второй условный оператор и в переменную **a** запишем соответственно 1 или -1 (**s[1] in ['+','-']** означает, что **s[1]** принадлежит множеству, состоящему из знаков «**+**» и «**–**»). В остальных случаях раскодируем значение переменной **a** (с помощью Val) из той части строки, которая находится слева от буквы 'a'. Затем удаляем из строки всю обработанную часть (вместе с буквой 'a') и устанавливаем номер символа в 0 (чтобы начать с начала измененной строки на следующем шаге цикла). Аналогично раскодируется значение **b**.

Когда мы определяем свободный член уравнения, мы найдем символы ‘+’, ‘-‘ или ‘!’ (если этот коэффициент стоит в конце строки). Обратим внимание, что если символ ‘+’ и ‘-‘ – первый в строке, то останавливаться не нужно, число еще не закончилось, а только началось.

```
if (s[i] = '+') or (s[i] = '-') then
  if i > 1 then begin
    Val(Copy(s,1,i-1), c, r);
    s:=Copy(s, i, Length(s)-i);
    i := 0;
  end;
```

Если коэффициент последний, то встретим '!':

```
if (s[i]] = '!') and (i > 1) then
  Val(Copy(s,1,i-1), c, r);
```

Условие **i>1** отсекает случай, когда свободный член – не последний коэффициентв в строке, и перед символом '!' стоит ‘a’ или ‘b’.

Квадратное уравнение решается классическим способом. Предполагаем (по условию), что дискриминант не меньше нуля и уравнение имеет вещественные корни. Вот полная программа:

```
var F: Text;
    s: string;
    i, r: integer;
    a, b, c, D, x1, x2: real;
begin
  Assign(F, 'eq.txt');
  Reset(F);
  Readln(F, s);
  Close(F);
  s:=s+'!';
```

```
    i:=1;
    while i <= Length(s) do begin
      case s[i] of
        'a': begin
               if i = 1 then a := 1 { нет коэффициента => a=1}
               else
               if (i = 2) and (s[1] in ['+','-'] then
                 if s[1] = '+' then a := 1   { a=1 или a=-1}
                 else a := -1
               else
                 Val(Copy(s,1,i-1), a, r); { есть коэффициент }
               s:=Copy(s, i+1, Length(s)-i);
               i := 0;
             end;
        'b': begin
               if i = 1 then b := 1 { нет коэффициента => b=1}
               else
               if (i = 2) and (s[1] in ['+','-'] then
                 if s[1] = '+' then b := 1   { b=1 или b=-1}
                 else b := -1
               else
                 Val(Copy(s,1,i-1), b, r); { есть коэффициент }
               s:=Copy(s, i+1, Length(s)-i);
               i := 0;
             end;
        '+','-':
             if i > 1 then begin
               Val(Copy(s,1,i-1), c, r);
               s:=Copy(s, i, Length(s)-i);
               i := 0;
             end;
        '!':
             if i > 1 then
                Val(Copy(s,1,i-1), c, r);
      end;
      i:=i+1;
    end;
    D:= b*b-4*a*c;
    x1 := (-b - sqrt(D)) /(2*a);
    x2 := (-b + sqrt(D)) /(2*a);
    Append(F);
    writeln(F, x1);
    writeln(F, x2);
    Close(F);
  end.
```

Альтернативное решение задачи предложил **А.С. Абрамов** (лицей при РГСУ, г. Воронеж):

```
  var F: Text;
      s, s0: string;
      i, k, r: integer;
      a, b, c, D, x1, x2: real;
      mas : array[1..3] of string;
  begin
```

```
  Assign(F, 'eq.txt');
  Reset(F);
  Readln(F, s);
  Close(F);
{ разбиваем уравнение на 3 части, каждая часть
  будет содержать коэффициент уравнения и его знак,
  а также букву 'a' или 'b', если это не свободный член }
  i:= 0;
  for k:=1 to 3 do
    repeat
      i:=i+1;
      mas[k]:= mas[k] + s[i]
    until (s[i+1]='+') or (s[i+1]='-') or (i=length(s));
{ если 1-й член положительный, добавляем ему его знак '+'}
  if s[1] <> '-' then
    mas[1]:= '+' + mas[1];
{ если коэффициент при a и b указан неявно, т.е. равен 1,
 добавим эту 1 между знаком и буквой }
  for k:=1 to 3 do
    if (length(mas[k]) = 2)  and
       ((mas[k][2] = 'a') or (mas[k][2] = 'b')) then
      mas[k]:= mas[k][1] + '1' + mas[k][2];
{ в случае a или b на последнем месте, читаем до
  предпоследнего символа, иначе - до последнего,
  и переводим строку в число }
  for k:=1 to 3 do begin
     s:= mas[k];
     i:= length(s);
     if mas[k][i] = 'a' then begin
       s0:= Copy(s,1,i-1);
       Val(s0, a, r);
     end
     else
     if mas[k][i] = 'b' then begin
       s0:= Copy(s,1,i-1);
       Val(s0, b, r);
     end
     else begin
       s0:= Copy(s,1,i);
       Val(s0, c, r);
     end;
  end;
  { вычисление корней уравнения }
  D:= b*b - 4*a*c;
  x1 := (-b - sqrt(D)) /(2*a);
  x2 := (-b + sqrt(D)) /(2*a);
  Append(F);
  writeln(F, x1);
  writeln(F, x2);
  Close(F);
end.
```

Еще одно решение (**Е.В. Хламов, г. Иркутск**) основано на использовании конечного автомата:

```
const sost:array[1..2, 1..6] of
       integer = ((2,2,2,1,1,7),(2,2,2,1,1,7));
var k, D,i,n,z: integer;
    s:string;
    a,b,c,Y:integer;
begin
  a:=0; b:=0; c:=0;
  i:=1;       n:=1;
  read(s);
  s:= s + '!';
  while n<>7 do begin
    case s[i] of
      '-': k:=1;
      '+': k:=2;
      '0'..'9': k:=3;
      'A','a': k:=4;
      'B','b': k:=5;
      '!': k:=6;
     end;
     case n of
       1: case k of
            1: begin z:=-1;Y:=0; end;
            2: begin z:=1; Y:=0; end;
            3: begin z:=1; Y:=ord(s[i])-ord('0');end;
            4: a:=1;
            5: b:=1;
            6: if c=0 then c:=y*z;
          end;
       2: case k of
            1:  begin c:=Y*z; z:=-1;Y:=0;  end;
            2:  begin c:=Y*z; z:=1;Y:=0;  end;
            3:  Y:=Y*10+ord(s[i])-ord('0') ;
            4:  a:=Y*z;
            5:  b:=Y*z;
            6:  c:=Y*z;
          end;
    end;
    n:=sost[n,k]; i:=i+1;
  end;
  D:=b*b-4*a*c;
  if D<0 then
    Writeln('Нет корней')
  else begin
    writeln((-b-sqrt(D))/(2*a));
    writeln((-b+sqrt(D))/(2*a));
  end;
end.
```

В решении, которое предложила **О.Д. Аралова** (г. Воронеж), строка обрабатывается с конца. Главная проблема состоит в том, чтобы правильно разбить исходную строку символов на три части:

- слагаемое с коэффициентом уравнения *a* и символом 'a',
- слагаемое с коэффициентом уравнения *b* и символом 'b'

- свободный член уравнения *c*.

Если для такого разделения идти по строке слева направо, то возникает необходимость добавления какого-либо символа в конец строки для корректного завершения проверки. Удобнее организовать цикл справа налево.

Поскольку надо проверять исходную строку на наличие знака "+" или "-" перед первым слагаемым, то с этого и начинается алгоритм. При отсутствии знака в начало строки приписывается "+". Получается сумма трёх слагаемых со знаком "+" или "-" перед каждым:

```
if (s[1]<>'+') and (s[1]<>'-') then
  s:='+'+s;
```

Далее в цикле проверяется вся строка S от последнего символа до первого. Каждый из символов копируется в новую строку S1 (S1:=S[k]+S1) до тех пор, пока не встретится знак "+" или "-". В этом случае в S1 уже собрано всё слагаемое. Данное слагаемое-строка превращается в числовой коэффициент квадратного уравнения *a*, *b* или *c* при помощи функции *Coeff* с формальными параметрами *St* (строка, подлежащая переводу в число) и *len* (её длина):

```
For k:= length(s) downto 1 do begin
  s1:=s[k]+s1; {В s1 копируем все символы из s, начиная с
                последнего}
  if (s[k]='+') or (s[k]='-') then begin
       {если встретился знак, то в S1 - весь коэффициент}
    L:=length(s1);
    case s1[L] of {проверка S1: это a? b? c?}
      'a': a:=Coeff(s1,L);
      'b': b:=Coeff(s1,L)
      else c:=Coeff(s1,L)
    end;
    s1:=''    {S1 готово для работы с новым коэффициентом}
  end
end;
```

После вызова функции Coeff строка *S1* очищается для работы со следующей частью исходной строки *S*.

Выход из цикла *For-downto* происходит, когда *в S* не остаётся ни одного непроверенного символа. Это значит, что в переменных *a*, *b* и *c* находятся числовые значения соответствующих коэффициентов квадратного уравнения.

В функции удаляется буква из строк, соответствующих коэффициентам *a* или *b*. Если в строке остаётся только знак "+" или "-", то после него вставляется "1". Изменив таким образом переданную строку, функция переводит её в целое число и возвращает в основную программу результат - значение коэффициента уравнения:

```
function Coeff(St: String; len:integer): integer;
begin
  if (St[len]='a') or (St[len]='b') then
    delete (St,len,1);  {удалили букву}
  if length(St)=1 then St:=St+'1';
    {если в строке остался только знак, приписали "1"}
  Coeff:=StrToInt(St);  {перевели строку в число}
end;
```

Корни уравнения определяются стандартным способом. Так как по условию задачи проверка дискриминанта на отрицательность не нужна, в *d* можно сразу вычислять корень из дискриминанта. Вот полная программа:

```
Var s, s1: string;
    a, b, c, k, L: integer;
    d, x: real;
function Coeff(St: String; len:integer): integer;
   {функция переводит строку-коэффициент в число}
begin
  if (St[len]='a') or (St[len]='b') then
    delete (St,len,1);  {удалили букву}
  if length(St)=1 then St:=St+'1';
    {если в строке остался только знак, приписали "1"}
  Coeff:=StrToInt(St);  {перевели строку в число}
end;
begin
  Readln(s);
  if (s[1]<>'+') and (s[1]<>'-') then s:='+'+s;
  {добавили в начало "+", если 1-ый символ строки - не знак}
  a:=0; b:=0; c:=0;
  s1:='';
  For k:= length(s) downto 1 do begin
    s1:=s[k]+s1;   {В s1 копируем все символы из s,
                    начиная с последнего}
    if (s[k]='+') or (s[k]='-') then begin
     {если встретился знак, то в S1 - весь коэффициент}
    L:=length(s1);
    case s1[L] of  {проверка S1: это a? b? c?}
      'a': a:=Coeff(s1,L);
      'b': b:=Coeff(s1,L)
     else c:=Coeff(s1,L)
    end;
    s1:='' {S1 готово для работы с новым коэффициентом}
  end
end;
{Нахождение корней уравнения, в d вычисляется сразу
 корень из дискриминанта}
d:=sqrt(b*b-4*a*c);
x:=(-b-d)/(2*a);
if d=0 then writeln ('Один корень x=', x)
else writeln ('x1=', x,'   x2=',  (-b+d)/(2*a));
end.
```

35) Простая задача, в которой нужно подсчитать, сколько раз встречается каждое из чисел из диапазона 1..12 во входных данных, а затем отсортировать список задач по возрастанию числа запросов. Для хранения данных удобнее использовать запись (структуру) с двумя целыми полями: в одном хранится номер задачи, во втором – количество запросов по ней:

```
type TInfo = record
       zadacha: integer;
       count: integer;
     end;
```

Всего может быть 12 задач, поэтому нужно объявить массив из 12 таких записей:

```
const MAX = 12;
var Info: array[1..MAX] of TInfo;
```

Перед началом работы в поле **zadacha** каждой структуры нужно записать ее порядковый номер, а в поле **count** – нуль (обнулить счетчик):

```
for i:=1 to MAX do begin
   Info[i].zadacha := i;
   Info[i].count := 0;
end;
```

Теперь читаем исходные данные: сначала общее количество запросов N, а потом N чисел (номеров задач), после чтения очередного номера задачи увеличиваем соответствующй счетчик:

```
readln(N);
for i:=1 to N do begin
  readln(zNo);
  Info[zNo].count := Info[zNo].count + 1;
end;
```

После этого массив записей сортируется по возрастанию поля **count** (здесь применен метод пузырька «наоборот» - за каждый проход самый тяжелый элемент едет вниз):

```
for i:=1 to MAX-1 do
  for j:=1 to MAX-i do
    if Info[j].count > Info[j+1].count then begin
      temp := Info[j];
      Info[j] := Info[j+1];
      Info[j+1] := temp;
    end;
```

Здесь чувствуется выгода от использования структур: иначе пришлось бы вводить два массива (номера задач и счетчики запросов) и в цикле переставлять одновременно два массива (см. решение задачи 25). Обратите внимание, что в циклах нужно использовать размер массива **MAX**, а не **N**!

Осталось вывести результат, не забывая проверить на равенство счетчика нулю (задачи, которых нет в запросах, выводить не нужно):

```
for i:=1 to MAX do
  if Info[i].count > 0 then
    writeln(Info[i].zadacha, ' ', Info[i].count);
```

Полная программа для решения этой задачи:

```
program qq;
const MAX = 12;
type TInfo = record
       zadacha: integer;
       count: integer;
     end;
var Info: array[1..MAX] of TInfo;
    i, j, N, zNo: integer;
    temp: TInfo;
begin
    { начальные установки }
```

```
    for i:=1 to MAX do begin
       Info[i].zadacha := i;
       Info[i].count := 0;
    end;
      { ввод данных }
    readln(N);
    for i:=1 to N do begin
      readln(zNo);
      Info[zNo].count := Info[zNo].count + 1;
    end;
      { сортировка }
    for i:=1 to MAX-1 do
      for j:=1 to MAX-i do
        if Info[j].count > Info[j+1].count then begin
          temp := Info[j];
          Info[j] := Info[j+1];
          Info[j+1] := temp;
        end;
      { вывод результата }
    for i:=1 to MAX do
      if Info[i].count > 0 then
        writeln(Info[i].zadacha, ' ', Info[i].count);
  end.
```

Теперь посмотрим, как можно было решить задачу без записей, используя массивы. например, в решении, предложенном **Е.Н. Смирновой** (г. Брянск) используется только один массив счетчиков:

```
  var count: array[1..MAX] of integer;
```

который сначала заполняется нулями:

```
  for i:=1 to MAX do count[i] := 0;
```

При этом ввод данных с увеличением счетчиков выглядит так:

```
  readln(N);
  for i:=1 to N do begin
    readln(zNo);
    count[zNo] := count[zNo] + 1;
  end;
```

Остается решить проблему вывода результата. Заметим, что данные неотсортированы, если просто отсортировать массив счетчиков по неубыванию, то будет потеряна информация о номерах задач. Поэтому предлагается такой алгоритм, учитывающий, что значение счетчика уже не нужно после того, как оно выведено на экран:

1) ищем минимальный по величине ненулевой счетчик с номером **nMin**
2) выводим соответствующий номер задачи (**nMin**) и значение счетчика (**count[nMin]**)
3) обнуляем **count[nMin]**, чтобы не учитывать его при следующем проходе
4) При выполнении шага 1 сначала находим первый ненулевой элемент в массиве count и записываем его номер в переменную **nMin**:

```
  nMin := 1;
  while (count[nMin]=0) and (nMin < MAX) do
    nMin := nMin+1;
```

Если такого элемента нет, будет выполняться условие **count[nMin]=0**, это значит, что все результаты выведены. При этом можно досрочно выйти из цикла с помощью оператора break:

```
if count[nMin] = 0 then break;
```

Иначе (если найден ненулевой счетчик), мы ищем далее по массиву минимальный ненулевой элемент:

```
for j:=nMin to max do
  if (count[j]<>0) and (count[j]<count[nMin])
  then nMin:=j;
```

выводим на экран его номер (номер задачи) и значение (количество запросов по этой задаче) и обнуляем этот счетчик:

```
writeln (nMin, ' ', count[nMin]);
count[nMin] := 0;
```

Вот полная программа:

```
program qq;
const MAX = 12;
var count: array[1..MAX] of integer;
    i, j, nMin, N, zNo: integer;
begin
    { начальные установки }
  for i:=1 to MAX do count[i] := 0;
    { ввод данных и увеличение счетчиков }
  readln(N);
  for i:=1 to N do begin
    readln(zNo);
    count[zNo] := count[zNo] + 1;
  end;
    { вывод результатов }
  for i:=1 to MAX do begin
      { поиск первого ненулевого счетчика }
    nMin := 1;
    while (count[nMin]=0) and (nMin < MAX) do
      nMin := nMin+1;
    if count[nMin] = 0 then break;
      { поиск минимального ненулевого счетчика }
    for j:=nMin to max do
      if (count[j]<>0) and (count[j]<count[nMin])
      then nMin:=j;
      { вывод результата }
    writeln (nMin, ' ', count[nMin]);
    count[nMin] := 0;
  end;
end.
```

36) В сравнении с предыдущей задачей, здесь есть одно усложнение. При вводе нужно определять, встречался ли этот фильм в списке раньше. Если встречался, то нужно просто увеличить соответствующий счетчик, если нет – запомнить название фильма и записать в его счетчик единицу. Так же, как и в предыдущей задаче, будем использовать записи (структуры), каждый элемент хранит название фильма и число поданных за него голосов:

```
type TInfo = record
       film: string;
       count: integer;
     end;
```

Введем целую переменную **K** – счетчик уже найденных фильмов, который в начале равен нулю. Когда прочитано очередное название фильма (в символьную строку **s**, нужно проверить если ли уже в списке этот фильм. Если есть, мы просто увеличиваем его счетчик. Для поиска удобно использовать цикл с условием, который просматривает первые K элементов массива:

```
j := 1;
while j <= K do
  if s = Info[j].film then begin
    Info[j].count := Info[j].count + 1;
    j := MaxInt;
  end
  else j := j + 1;
```

Если фильм нашли в списке, кроме увеличения счетчика в переменную j записывается значение **MaxInt**, это сделано для того, чтобы цикл завершился. Таким образом, после окончания цикла переменная j может быть равна **MaxInt** (если фильм уже был в списке) или **K+1** (если фильм не найден и цикл закончился при нарушении условия **j<=K**). В последнем случае добавляем его в список и записываем в счетчик 1:

```
if j = K+1 then begin
  K := K + 1;
  Info[K].film := s;
  Info[K].count := 1;
end;
```

При сортировке нужно использовать не весь массив, а только первые K записей (столько, сколько было фактически найдено разных фильмов):

```
for i:=1 to K-1 do
  for j:=1 to K-i do
    if Info[j].count < Info[j+1].count then begin
      temp := Info[j];
      Info[j] := Info[j+1];
      Info[j+1] := temp;
    end;
```

При выводе, в отличие от предыдущей задачи, не нужно делать проверку на неравенство нулю – если фильм был найден, его счетчик заведомо не меньше 1:

```
for i:=1 to K do
  writeln(Info[i].film, ' ', Info[i].count);
```

Вот полная программа:

```
const MAX = 10;
type TInfo = record
       film: string;
       count: integer;
     end;
var Info: array[1..MAX] of TInfo;
    i, j, N, K: integer;
    s: string;
```

```
    temp: TInfo;
begin
  K := 0;
  readln(N);
  for i:=1 to N do begin
    readln(s);
    j := 1;
    while j <= K do
      if s = Info[j].film then begin
         Info[j].count := Info[j].count + 1;
         j := MaxInt;
      end
      else j := j + 1;
    if j = K+1 then begin
       K := K + 1;
       Info[K].film := s;
       Info[K].count := 1;
    end;
  end;
  for i:=1 to K-1 do
    for j:=1 to K-i do
      if Info[j].count < Info[j+1].count then begin
        temp := Info[j];
        Info[j] := Info[j+1];
        Info[j+1] := temp;
      end;
  for i:=1 to K do
    writeln(Info[i].film, ' ', Info[i].count);
end.
```

37) Сложность этой задачи состоит в том, что объем используемой памяти не должен зависеть от длины последовательности чисел. Следовательно, запоминать числа нельзя. Нежелательно хранить и все возможные произведения введенных чисел (их может быть миллион, от 1 до 1000000, поскольку каждое введенное число находится в интервале от 1 до 1000).

Таким образом, массив заводить нельзя. Введем следующие переменные:

x – вспомогательная переменная, в которую читается очередное число
count – счётчик введенных чисел
C0 – контрольное значение

Общая структура программы выглядит так:

```
count:=0;
readln(x);
while x <> 0 do begin
  count:= count + 1;
  { обработка введенного значения x }
  readln(x);
end;
readln(C0);
```

Структура цикла определяется тем, что

1) до начала цикла нужно ввести первое значение элемента последовательности; если оно равно 0, то цикл выполнять не нужно;

2) если очередное введенное значение равно нулю, нужно закончить цикл и не обрабатывать его (это не элемент последовательности), поэтому оператор **readnl(x)** стоит в конце тела цикла;

Заметим, что можно было использовать и цикл с постусловием, но при этом нужен еще условный оператор:

```
count:=0;
repeat
  readln(x);
  if x > 0 then begin
    count:= count + 1;
    { обработка введенного значения x }
  end;
until x = 0;
readln(C0);
```

Таким образом, главная проблема – обработка введенного значения x так, чтобы в конце цикла мы получили в некоторой переменной (назовем ее **C**) контрольное значение, и его можно было сравнить с введенным значением **C0**. По условию контрольное значение – это **наибольшее** произведение чисел, **делящееся на 6**.

Сначала предположим, что в этом произведении один из сомножителей делится на 6, тогда второй может быть любым. Наибольшее произведение таких пар может быть найдено следующим образом:

1) найти наибольшее число, делящееся на 6.
2) найти наибольшее число, кроме найденного в п. 1 (заметим, что оно также может делиться на 6!!!).

Для поиска наибольшего из вводимых чисел не нужно хранить все числа. Введем две переменные:

max6 – максимальное число, делящееся на 6
max – максимальное из оставшихся чисел

Поскольку все числа больше 0, сначала в эти переменные запишем нули.
Допустим, что очередное число прочитано в переменную x. Тогда поиск **max6** и **max** на очередном шаге цикла выглядит так:

```
if (x mod 6 = 0) and (x > max6) then begin
  if max6 > max then max:= max6;
  max6:= x
end
else
  if x > max then max:= x;
```

Обратите внимание, что в случае, когда найдено новое значение **max6**, нужно проверить, не было ли старое значение больше, чем **max**:

```
if max6 > max then max:= max6;
```

Теперь рассмотрим еще один вариант: произведение делится на 6, но ни один из сомножителей не делится на 6. Это может быть тогда, когда один сомножитель делится на 2, в второй – на 3. Тогда получается, что нужно среди всех введенных чисел, не делящихся на 6, найти наибольшее, делящееся на 2 (в программе будем хранить его в переменной **max2**), и наибольшее, делящееся на 3 (переменная **max3**). Их произведение – второй кандидат на искомый максимум.

Начальные значение **max2** и **max3** можно также взять нулевыми (все вводимые числа больше нуля). На каждом шаге цикла они могут измениться, для этого используем такой условный оператор:

```
if x mod 6 <> 0 then begin { если число не делится на 6 }
  if (x mod 2 = 0) and (x > max2) then max2:= x;
  if (x mod 3 = 0) and (x > max3) then max3:= x;
end;
```

Можно немного упростить этот оператор, убрав условие «неделимости» на 6. Проблема только в том, чтобы число не попало одновременно в группу «делящихся на 2» и «делящихся на 3». Для этого достаточно использовать **else**:

```
  if (x mod 2 = 0) and (x > max2) then max2:= x
  else
    if (x mod 3 = 0) and (x > max3) then max3:= x;
```

После окончания цикла нужно выбрать максимальное из произведений **max6*max** и **max2*max3**, это и будет контрольное значение (переменная **C**):

```
if max6*max > max2*max3 then
     C:= max6*max
else C:= max2*max3;
```

Приведем полную программу:

```
program qq;
var x, count, C, C0, max, max2, max3, max6: integer;
begin
  count:=0; max:=0; max2:=0; max3:=0; max6:=0;
  readln(x);
  while x <> 0 do begin
    count:= count + 1;
    if (x mod 6 = 0) and (x > max6) then begin
       if max6 > max then max:= max6;
       max6:= x
    end
    else if x > max then max:= x;
    if (x mod 2 = 0) and (x > max2) then max2:= x
    else
      if (x mod 3 = 0) and (x > max3) then max3:= x;
    readln(x);
  end;
  readln(C0);
  if max6*max > max2*max3 then
       C:= max6*max
  else C:= max2*max3;
  writeln('Получено чисел: ', count);
  writeln('Полученное контрольное значение: ', C0);
  writeln('Вычисленное контрольное значение: ', C);
  if C = C0 then
       writeln('Контроль пройден.')
  else writeln('Контроль не пройден.');
end.
```

38) Это достаточно простая задача, в ней фактически нужно искать возрастающие последовательности чисел, и среди них искать последовательность с максимальной разностью конечного и начального элементов. Например, в цепочке чисел

21, 22, 13, 5, 1, 2, 3, 5, 1, 23

есть 5 неубывающих последовательностей (они выделены разными цветами), из них две состоят из одного элемента. Для них величины «подъемов» (разности между конечным и начальным числами) равны, соответственно, 22-21=1, 0, 0, 5-1=4 и 23-1=22. Максимальный «подъем» 22 имеет последняя последовательность.

Во-первых, нужно считать введенные числа. Для этого будем использовать переменную-счётчик **N**. Её начальное значение равно 0, с каждым прочитанным числом она увеличивается на 1.

По условию задачи объем расходуемой памяти не должен зависеть от количества чисел, поэтому запоминать их в массиве нельзя. Будем хранить только что прочитанное значение в целой переменной **X**. Тогда цикл чтения, заканчивающийся при вводе нуля, можно записать в виде бесконечного цикла (**while True do** …) с выходом через оператор **break**:

```
while True do begin
  readln(X);                { читаем число }
  if X = 0 then break;      { если 0, то выход из цикла }
  N:= N + 1;                { увеличиваем счётчик чисел}
  { обработка X }
end;
```

Вся последовательность просматривается только 1 раз (вернуться к предыдущим числам тоже нельзя). В этих условиях для того, чтобы определить, продолжается ли дальше неубывающая цепочка, достаточно знать только одно предыдущее значение, которое будем хранить в целой переменной **Xprev**. Кроме того, для поиска максимального «подъема» нам нужны две целых переменных, обозначим их **L** и **LMax**. Одна из них (**L**) обозначает величину подъема текущей возрастающей последовательности, а вторая (**Lmax**) – максимальную на данный момент величину подъема.

Если новое значение больше предыдущего, подъем продолжается, и новую величину подъема можно рассчитать так:

```
L := L + X - Xprev;
```

Это значит, что к предыдущему значению добавляется еще одна «ступенька» – разность между последним и предпоследним числами. Кроме того, удобно сразу проверить, не стало ли **L** больше, чем **Lmax**. Если это так, нужно скопировать значение **L** в **Lmax**, так как мы нашли новую максимальную величину подъема.

Если же возрастающая последовательность закончилась (**X<=Xprev**), нужно сбросить значение **L** в ноль. Таким образом, обработка очередного значения **X** выглядит так:

```
if X > Xprev then begin
  L := L + X - Xprev;
  if L > LMax then LMax := L;
end
else L := 0;
Xprev := X;
```

Заметим, что начальное значение **Xprev** нужно выбрать такое, чтобы на первом шаге НЕ было выполнено условие **X >Xprev** и в переменную **L** было записано нулевое значение (началась новая возрастающая последовательность). Учитывая, что все числа по условию не превышают 1000, можно принять

```
Xprev := 1001;
```

В конце программы остается вывести значения переменных **N** (количество чисел) и **Lmax** (наибольшую высоту подъема). Приведем полную программу:

```
var X, Xprev, N, L, LMax: integer;
begin
  N:= 0;
  LMax:= 0;
  Xprev:= 1001;
  L:= 0
  while True do begin
    readln(X);                { читаем число }
    if X = 0 then break;      { если 0, то стоп }
    N:= N + 1;                { увеличиваем счётчик чисел}
    if X > Xprev then begin
      L := L + X - Xprev;     { изменяем высоту подъема }
      if L > LMax then LMax := L;
    end
    else L := 0;
    Xprev := X;
  end;
  writeLn('Получено ', N, ' чисел');
  writeLn('Наибольшая высота подъема ', LMax);
end.
```

Обратите внимание, что обнулять значение **L** перед началом цикла ввода данных не обязательно. При выбранном начальном значении **Xprev** условие **X>Xprev** не может ыть выполнено, и в **L** запишется 0.

Возможен другой вариант решения (автор **О.В. Алимова**), который отличается в некоторых деталях:

- для окончания цикла не используется оператор **break** (этот оператор не приветствуется теоретиками);
- запоминается начальное значение возрастающей последовательности (в переменной **Xstart**), а не текущая высота подъема.

В переменную **Xstart** в самом начале записывается ноль, поскольку такого «рабочего» значения быть не может.

```
Xstart:= 0;
```

Затем, при чтении первого числа из входного потока, сразу записываем это число в **Xstart**:

```
readln(X);
if Xstart = 0 then Xstart := X;
```

Очевидно, что условный оператор выполнится только один раз, при первом чтении.
Обработка очередного прочитанного значения сводится к следующему:

- увеличить счётчик чисел **N**
- если подъем кончился (только что прочитанное значение меньше или равно предыдущему), найти длину полученной цепочки (вычесть из предыдущего значения стартовое) и проверить, не стала ли она больше, чем **Lmax;** записать последнее прочитанное число в **Xstart** (эта начала следующего участка подъема)
- записать последнее прочитанное число в **Xprev**

Получается такой цикл ввода и обработки данных:

```
repeat
  readln(X);
  if Xstart = 0 then Xstart := X;
  N:= N + 1;
  if X <= Xprev then begin
    if Xprev - Xstart > LMax then
```

```
          LMax := Xprev - Xstart;
        Xstart := X;
      end;
      Xprev:=X;
    until X = 0;
```

Проверим, что будет происходить, когда прочитаем завершающий 0. Счетчик увеличится (это нужно будет учесть), условие **X < Xprev** выполнится, и программу зафиксирует окончание подъема. Таким образом, последнюю возрастающую цепочку мы не потеряем. Затем в **Xstart** и в **Xprev** будет записан 0, но это нас уже не волнует – цикл закончится, поскольку выполнено условие **X = 0**.

Поскольку мы подсчитали и последний 0, при выводе количество чисел нужно уменьшить на 1:

```
writeLn('Получено ', N-1, ' чисел');
```

Вот полная программа:

```
var X, Xprev, Xstart, N, LMax: integer;
begin
  N:= 0;
  LMax:= 0;
  Xstart:= 0;
  Xprev:= 0;
  repeat
    readln(X);
    if Xstart = 0 then Xstart := X;
    N:= N + 1;
    if X <= Xprev then begin
      if Xprev - Xstart > LMax then
         LMax := Xprev - Xstart;
      Xstart := X;
    end;
    Xprev:=X;
  until X = 0;
  writeLn('Получено ', N-1, ' чисел');
  writeLn('Наибольшая высота подъема ', LMax);
end.
```

39) С первого взгляда эта задача кажется достаточно простой: нужно ввести две строки, подсчитать количество латинских букв в каждой из них (записав эти данные в два массива счетчиков **f1** и **f2**), затем найти сумму произведений элементов этих массивов и разделить её на произведение длин строк.

В языке Паскаль индексами элементов массива могут быть символы, поэтому можно объявить массивы счетчиков и переменную **k** так:

```
var k: char;
    f1, f2: array['A'..'Z'] of integer;
```

Сначала массивы счётчиков нужно обнулить:

```
for k:='A' to 'Z' do begin
  f1[k] := 0; f2[k] := 0;
end;
```

Дальше в цикле обрабатываем каждую строку. Например, для строки **s1** получаем

```
for i:=1 to length(s1) do begin
  { обработать символ s1[i] }
```

```
end;
```

Что входит в обработку? Нужно определить индекс нужного счётчика («номер» символа в латинском алфавите), и затем увеличить этот счётчик (элемент массива). Необходимо учесть две особенности:

1) все строчные буквы нужно преобразовывать к прописным (заглавным), так как индексы массивов **f1** и **f2** – заглавные буквы; например, если символьная переменная **k** содержит строчную букву, для преобразования её в соответствующую прописную нужно вычесть из её кода (полученного с помощью функции **Ord**) код буквы «a» и прибавить код буквы «A», полученный новый код преобразовать в символ с помощью функции **Chr**:

```
k := Chr(Ord(s1[i]) - Ord('a') + Ord('A'));
```

   вместо этого можно просто вызвать функцию **UpCase** (преобразовать символ в верхний регистр), если она есть в той версии языка, на которой вы пишете:

```
k := UpCaSe(s1[i]);
```

2) все символы, не являющиеся латинскими буквами, нужно игнорировать (счётчики не меняются).

Поэтому цикл обработки строки s1 выглядит так:

```
for i:=1 to Length(s1) do begin
  k := ' ';
  if ('A' <= s1[i]) and (s1[i] <= 'Z') then
    k := s1[i]
  else
  if ('a' <= s1[i]) and (s1[i] <= 'z') then
    k := Chr(Ord(s1[i]) - Ord('a') + Ord('A'));
  if k <> ' ' then f1[k] := f1[k] + 1;
end;
```

Обратите внимание, что если очередной символ не является латинской буквой (заглавной или строчной), в переменной **k** остается пробел, и счетчики не изменяются.

Для строки **s2** нужно написать аналогичный цикл:

```
for i:=1 to Length(s2) do begin
  k := ' ';
  if ('A' <= s2[i]) and (s2[i] <= 'Z') then
    k := s2[i]
  else
  if ('a' <= s2[i]) and (s2[i] <= 'z') then
    k := Chr(Ord(s2[i]) - Ord('a') + Ord('A'));
  if k <> ' ' then f2[k] := f2[k] + 1;
end;
```

Теперь находим сумму произведений соответствующих счетчиков:

```
sumFF := 0;
for k:='A' to 'Z' do
  sumFF := sumFF + f1[k]*f2[k];
```

и выводим ответ:

```
writeln(sumFF/(length(s1)*length(s2)):10:3);
```

Вот полная программа:

```
program qq;
var i, sumFF: integer;
    s1, s2: string;
    f1, f2: array['A'..'Z'] of integer;
    k: char;
begin
readln(s1);
readln(s2);
for k:='A' to 'Z' do begin
  f1[k] := 0; f2[k] := 0;
end;
for i:=1 to Length(s1) do begin
  k := ' ';
  if ('A' <= s1[i]) and (s1[i] <= 'Z') then
    k := s1[i]
  else
  if ('a' <= s1[i]) and (s1[i] <= 'z') then
    k:= Chr(Ord(s1[i]) - Ord('a') + Ord('A'));
          { или k:= UpCase(s1[i]); }
  if k <> ' ' then f1[k] := f1[k] + 1;
end;
for i:=1 to Length(s2) do begin
  k := ' ';
  if ('A' <= s2[i]) and (s2[i] <= 'Z') then
    k := s2[i]
  else
  if ('a' <= s2[i]) and (s2[i] <= 'z') then
    k := Chr(Ord(s2[i]) - Ord('a') + Ord('A'));
          { или k:= UpCase(s2[i]); }
  if k <> ' ' then f2[k] := f2[k] + 1;
end;
sumFF := 0;
for k:='A' to 'Z' do
  sumFF := sumFF + f1[k]*f2[k];
writeln(sumFF/(length(s1)*length(s2)):10:3);
end.
```

Заметим, что обе строки должны содержать хотя бы по одному символу. Если этот не гарантируется, оператор вывода нужно записать так:

```
if length(s1)*length(s2) > 0 then
  writeln(sumFF/(length(s1)*length(s2)):10:3)
else
  writeln(0);
```

В качестве оптимизации можно заранее вычислить длины обеих строк и записать их в новые переменные.

Однако, не все так просто. Является ли эта программа эффективной? С одной стороны, количество операций линейно зависит от длин строк **s1** и **s2** (это хорошо!), его можно оценить как **C*(length(s1)+length(s2))** при некоторой постоянной **C**. С другой стороны, мы использовали два массива по 26 элементов в каждом (можно придраться к неэффективности по использованию памяти).

Итак, попробуем без массивов. Можно, например, сделать цикл по всем буквам латинского алфавита и считать, сколько раз входит каждая буква в первую и вторую строки (эти переменные-счётчики назовем **count1** и **count2**). После прохода по обеим строкам, добавляем произведение **count1*count2** к сумме (для упрощения считаем, что в языке есть функция **UpCase**):

```
sumFF := 0;
for k:='A' to 'Z' do begin
  count1 := 0;
  for i:=1 to length(s1) do
    if k = UpCase(s1[i]) then
      count1 := count1 + 1;
  count2 := 0;
  for i:=1 to length(s2) do
    if k = UpCase(s2[i]) then
      count2 := count2 + 1;
  sumFF := sumFF + count1 * count2;
end;
```

Вот полная программа:

```
var i, count1, count2, sumFF: integer;
    s1, s2: string;
    k: char;
begin
readln(s1);
readln(s2);
sumFF := 0;
for k:='A' to 'Z' do begin
  count1 := 0;
  for i:=1 to length(s1) do
    if k = UpCase(s1[i]) then
      count1 := count1 + 1;
  count2 := 0;
  for i:=1 to length(s2) do
    if k = UpCase(s2[i]) then
      count2 := count2 + 1;
  sumFF := sumFF + count1 * count2;
end;
if length(s1)*length(s2) > 0 then
  writeln(sumFF/(length(s1)*length(s2)):10:3)
else
  writeln(0);
end.
```

Проанализируем эффективность. Прежде всего, мы избавились от массивов (это хорошо)! Но количество операций стали пропорционально **26*(length(s1)+length(s2))**, поскольку получился вложенный цикл. Это значит, что скорость вычислений уменьшилась (стало хуже!), а эффективность по памяти – увеличилась (стало лучше!).

Ещё один вариант, который не использует массивов и быстро работает для коротких строк. Фактически нас интересуют только те символы, которые встречаются в обеих строках, поэтому можно в цикле рассматривать все символы одной строки, например, **s1**:

```
sumFF := 0;
for i:=1 to length(s1) do begin
```

```
    { если s[i] – латинская буква то
        подсчитать count1 – сколько раз она входит в s1
        подсчитать count2 – сколько раз она входит в s2
        sumFF := sumFF + count1 * count2;
      все }
  end;
```

Здесь есть только одна проблема – а что если в первой строке буквы повторяются? Чтобы ее решить, можно «забивать» уже рассмотренные буквы пробелами, например, так (для упрощения считаем, что в языке есть функция **UpCase**):

```
  for i:=1 to Length(s1) do begin
    k := UpCase(s1[i]);
    if ('A' <= k) and (k <= 'Z') then begin
        { считаем эти буквы в оставшейся части строки s1 }
      count1 := 1;
      for j:=i+1 to Length(s1) do
        if k = UpCase(s1[j]) then begin
          count1 := count1 + 1;
          s1[j] := ' ';          { забиваем букву пробелом }
        end;
        { считаем эти буквы в строке s2 }
      count2 := 0;
      for j:=1 to Length(s2) do
        if k = UpCase(s2[j]) then
          count2 := count2 + 1;
        { увеличиваем сумму }
      sumFF := sumFF + count1 * count2;
    end;
```

Теперь остается только вывести результат на экран. Вот полная программа:

```
  var i, j, count1, count2, sumFF: integer;
      s1, s2: string;
      k: char;
  begin
  readln(s1);
  readln(s2);
  sumFF := 0;
  for i:=1 to length(s1) do begin
    k := UpCase(s1[i]);
    if ('A' <= k) and (k <= 'Z') then begin
      count1 := 1;
      for j:=i+1 to Length(s1) do
        if k = UpCase(s1[j]) then begin
          count1 := count1 + 1;
          s1[j] := ' ';
        end;
      count2 := 0;
      for j:=1 to Length(s2) do
        if k = UpCase(s2[j]) then
          count2 := count2 + 1;
      writeln(k, ' ', count1, ' ', count2);
      sumFF := sumFF + count1 * count2;
    end;
```

```
end;
if length(s1)*length(s2) > 0 then
  writeln(sumFF/(length(s1)*length(s2)):10:3)
else
  writeln(0);
end.
```

Проанализируем эффективность. Количество операций стало пропорционально произведению длин строк (получился вложенный цикл), его можно оценить как **C*length(s1)*length(s2)**. Для длинных строк произведение длин во много раз больше, чем сумма длин, поэтому программа будет работать медленнее, чем предыдущий вариант. А для коротких строк – быстрее.

40) В этой задаче сначала нужно заполнить массив слов (обучающий блок). Естественно объявить массив так:

```
var words: array[1..27] of string;
```

и читать его в цикле:

```
for i:=1 to 27 do
  readln(words[i]);
```

Затем, как обычно, читаем количество строк с данными и обрабатываем их в цикле. Обработка сводится к тому, чтобы поместить записанное в строке число в переменную (назовем ее **number**) и добавить это число к сумме (переменная **sum**):

```
readln(N);
sum:=0;
for i:=1 to N do begin
  readln(s);
  { получить число и записать его в number }
  sum:= sum + number;
end;
```

Сложность в том, что нужно складывать только числа в интервале от 1 до 99, а остальные – игнорировать. Это значит, что если в строке встретится слово, не входящее в словарь, нужно записать в переменную **number** значение 0.

Используя тот факт, что слова в строке разделены пробелами, можно «резать» строку по этим пробелам, «откусывая» начальное слово и переводя его в число. Цикл обработки строки может быть записан так:

```
number:=0;
p:=1;
while p > 0 then begin
   p:= Pos(' ', s); { ищем номер пробела }
   if p = 0 then    { если пробела нет,  }
      s1 := s       { ... то слово - это вся строка }
   else begin       { если пробел есть, то ... }
     s1 := Copy(s, 1, p-1); { выделить первое слово }
     Delete(s, 1, p);       { и удалить его из строки }
   end;
   { !!! изменить number !!! }
end.
```

Самый важный момент – как изменить переменную **number**. Во-первых, нужно попытаться найти выделенное первое слово (строку **s1**) в словаре. Для этого используем цикл, в котором сравниваем **s1** с каждым элементом массива **words**:

```
j := 1; { начали с первого элемента }
while (j <= 27) and (s1 <> words[j]) do
  j:= j + 1; { перейти к следующему слову }
```

Цикл завершается, когда слово найдено (в этом случае **j<=27**) или же массив полностью просмотрен и совпадения не найдено (при этом **j>27**).

В первом случае (если строка **s1** найдена в словаре) нужно добавить к **number** соответствующее значение, например, так:

```
if j < 10 then                  { число от 1 до 9 }
  number:=number + j
else
if j < 19 then                  { число от 11 до 19 }
  number:=number + j + 1
else
  number:=number + (j-18)*10;   { десятки 10, 20, ... 90 }
```

или так:

```
if j < 19 then
     number:=number + j + (j div 10)
else number:=number + (j-18)*10;
```

Если строка **s1** не найдена в словаре (число не входит в диапазон 1..99), то нужно записать в **number** нуль, и в переменную **p** также нужно записать нуль, чтобы остановить цикл **while**. Таким образом, изменение **number** можно записать так:

```
if j <= 27 then
  if j < 19 then
       number:=number + j + (j div 10)
  else number:=number + (j-18)*10
else begin
  number:= 0; p:= 0;
end
```

Итак, все принципиальные моменты мы обсудили, теперь нужно собрать все в одну программу:

```
var sum, p, i, j, N, number: integer;
    s, s1: string;
    words: array[1..27] of string;
begin
  for i:=1 to 27 do    { читаем словарь }
    readln(words[i]);
  readln(N);           { читаем количество строк }
  sum:=0;
  for i:=1 to N do begin
    readln(s);         { читаем очередную строку }
    number := 0;
    p := 1;
    while p > 0 do begin
      p := Pos(' ',s);         { найти пробел }
      if p = 0 then s1 := s    { в строке одно слово }
```

```
        else begin                { выделить первое слово }
          s1 := Copy(s, 1, p-1);
          Delete(s, 1, p);
        end;
        j := 1;           { ищем слово в словаре }
        while (j <= 27) and (s1 <> words[j]) do j:=j+1;
        if j <= 27 then { если слово найдено… }
          if j < 19 then
               number:=number + j + (j div 10)
          else number:=number + (j-18)*10
        else begin        { если слово не найдено… }
          p:= 0; number:= 0;
        end
      end;
      sum:=sum+number;  { добавить число к сумме }
    end;
    writeln(sum);
  end.
```

Рассмотрим другие правильные варианты решения. Например, для сокращения основной программы (и для того, чтобы сделать ее более понятной) можно выделить операцию получения числа, соответствующего заданному слову, в функцию:

```
function Word2Value(s1: string): integer;
var j: integer;
begin
  j := 1;
  while (j <= 27) and (s1 <> words[j]) do j:=j+1;
  Word2Value:= 0;
  if j <= 27 then
    if j < 19 then
         Word2Value:= j + (j div 10)
    else Word2Value:= (j-18)*10;
end;
```

Тогда получается такое решение:

```
var sum, p, i, v, N,number: integer;
   s, s1: string;
   words: array[1..27] of string;
{-----------------------------------------------}
function Word2Value(s1: string): integer;
var j: integer;
begin
  j := 1;
  while (j <= 27) and (s1 <> words[j]) do j:=j+1;
  Word2Value:= 0;
  if j <= 27 then
    if j < 19 then
         Word2Value:= j + (j div 10)
    else Word2Value:= (j-18)*10;
end;
{-----------------------------------------------}
begin
  for i:=1 to 27 do
```

```
      readln(words[i]);
    readln(N);
    sum:=0;
    for i:=1 to N do begin
      readln(s);
      p := 1;
      number := 0;
      while p > 0 do begin
        p := Pos(' ',s);
        if p = 0 then s1 := s
        else begin
          s1 := Copy(s, 1, p-1);
          Delete(s, 1, p);
        end;
        v := Word2Value(s1);    { получить число из слова }
        if v > 0 then           { если известное число...}
          number:=number + v
        else begin             { если неизвестное число...}
          number := 0; p := 0;
        end;
      end;
      sum:=sum + number;
    end;
    writeln(sum);
  end.
```

Возможен еще один вариант, который, согласно критериям оценивания для этой задачи, считается правильным и эффективным. Нужно выделить массив не на 27, а на 99 строк,

```
var words: array[1..99] of string;
```

загрузить заданные 27 строк:

```
for i:=1 to 9 do readln(words[i]);
for i:=1 to 9 do readln(words[10+i]);
for i:=1 to 9 do readln(words[10*i]);
```

а затем составить *полный словарь* возможных «правильных» строк:

```
for i:=2 to 9 do
  for j:=1 to 9 do
    words[10*i+j]:= words[10*i] + ' ' + words[j];
```

Теперь основной цикл получается очень простой: сравниваем введенную строку со всеми строками из словаря, и если нашли совпадение, добавляем к сумме найденное число и выходим из цикла, используя оператор **break**:

```
sum:= 0;
for i:=1 to N do begin
  readln(f,s);
  for j:=1 to 99 do
    if s = words[j] then begin
      sum:=sum + j;
      break; { этот оператор можно не писать }
    end;
end;
```

Вот полная программа:

```
var sum, i, j, N: integer;
    s: string;
    words: array[1..99] of string;
begin
  for i:=1 to 9 do readln(words[i]);
  for i:=1 to 9 do readln(words[10+i]);
  for i:=1 to 9 do readln(words[10*i]);
  for i:=2 to 9 do
    for j:=1 to 9 do
      words[10*i+j] := words[10*i] + ' ' + words[j];
  readln(N);
  sum:=0;
  for i:=1 to N do begin
    readln(s);
    for j:=1 to 99 do
      if s = words[j] then begin
        sum:=sum + j;
        break;
      end;
  end;
  writeln(sum);
end.
```

Внутренний цикл **for** (в котором сравниваются строки) можно заменить на цикл с условием, тогда не нужен оператор **break**:

```
j := 1;
while (j <= 99) and (s <> words[j]) do
  j:= j + 1;
if j < 100 then sum:=sum + j;
```

Получилось очень короткое и красивое решение, но оно не лишено недостатков. Например, можно считать избыточным расход памяти (99 строк в массиве вместо 27). Однако, такое решение (согласно критериям оценивания для этой задачи) считается правильным и эффективным.

41) Решение этой задачи явно подразумевает три этапа:

1) чтение данных и подсчет букв, с которых начинаются слова, с помощью массива счётчиков;
2) сортировка букв и соответствующих счётчиков
3) вывод результата.

В Паскале массив счётчиков можно сделать с символьными индексами:

```
var count: array['a'..'z'] of integer;
```

В самом начале все счётчики нужно обнулить:

```
for c:='a' to 'z' do count[c]:=0;
```

Первый этап (чтение данных) может быть выполнен двумя способами: чтением по строкам и посимвольным чтением. Сначала покажем вариант чтения по строкам. Введем переменную **s** типа **string**:

```
var s: string;
```

и организуем бесконечный цикл, который заканчивается по оператору **break** тогда, когда введена строка, начинающаяся с символа *:

```
while True do begin
  readln(s);
  if (Length(s) > 0) and (s[1] = '*')
   then break;
  { обработка строки }
end;
```

Обратите внимание, что сначала нужно проверить, что длина строки больше нуля, то есть строка содержит хотя бы один символ. Иначе при вводе пустой строки программа завершится аварийно (будет обращение к элементу **s[1]**, которого нет).

Обработка состоит в том, что нужно найти в строке первую букву каждого слова. Что такое начало слова? Это ситуация, когда рядом стоят пробел и вслед за ним строчная латинская буква. В этом случае нужно увеличить счётчик, соответствующий этой букве. Есть одно исключение: строка может начинаться с буквы. Можно обрабатывать этот случай отдельно, а можно просто добавить в начало строки пробел:

```
s := ' ' + s;
for i:=1 to Length(s)-1 do
  if (s[i] = ' ') and (s[i+1] in ['a'..'z']) then
    count[s[i+1]]:= count[s[i+1]] + 1;
```

Второй этап – сортировка букв в порядке убывания значений счётчиков. Тут есть две проблемы: по-первых, недостаточно просто отсортировать элементы массива **count** по убыванию – при этом мы потеряем связь буквы со счётчиком. Поэтому придется выделить еще один массив, в котором будут храниться буквы:

```
var letters: array['a'..'z'] of char;
...
for c:='a' to 'z' do letters[c]:=c;
```

Вторая проблема: «*если количество слов, начинающихся на какие-то буквы, совпадает, эти буквы следует выводить в алфавитном порядке*». Методы сортировки, сохраняющие это свойство, называются устойчивыми. Например, метод пузырька – устойчивый метод, а метод выбора минимального элемента – нет (поэтому использовать его в этой задаче нельзя). Итак, применяем метод пузырька, переставляя элементы сразу двух массивов:

```
for c:='a' to 'y' do begin
  for d:='y' downto c do begin
    d1:= Succ(d); { следующий символ за d }
    if count[d] < count[d1] then begin
      i:= count[d]; count[d] := count[d1]; count[d1]:= i;
      temp:= letters[d]; letters[d]:=letters[d1];
      letters[d1]:=temp;
    end;
  end;
end;
```

Вывод результата не представляет сложности: если счётчик не равен нулю, выводим его значение вместе с соответствующей буквой:

```
for c:='a' to 'z' do
  if count[c] > 0 then
```

```
      writeln(letters[c], ' ', count[c]);
```

Вот полная программа:

```
var count: array['a'..'z'] of integer;
    letters: array['a'..'z'] of char;
    s: string;
    c, temp, d, d1: char;
    i: integer;
begin
   for c:='a' to 'z' do begin
     count[c]:=0;
     letters[c]:=c;
   end;
     { чтение данных, подсчёт начальных букв }
   while True do begin
     readln(s);
     if (Length(s) > 0) and (s[1] = '*') then break;
     s := ' ' + s;
     for i:=1 to Length(s)-1 do
       if (s[i] = ' ') and (s[i+1] in ['a'..'z']) then
         count[s[i+1]] := count[s[i+1]] + 1;
   end;
     { сортировка }
   for c:='a' to 'y' do begin
     for d:='y' downto c do begin
       d1:= Succ(d);
       if count[d] < count[d1] then begin
         i:= count[d]; count[d] := count[d1]; count[d1]:= i;
         temp:= letters[d]; letters[d]:=letters[d1];
         letters[d1]:=temp;
       end;
     end;
   end;
     { вывод результата }
   for c:='a' to 'z' do
     if count[c] > 0 then
       writeln(letters[c], ' ', count[c]);
end.
```

Во многих языках программирования индексы не могут быть символьными значениями, приходится использовать массивы с целыми индексами. Для преобразования символа в его числовой код в Паскале применяется функция **Ord**, а для обратного преобразования – функция **Chr**. Вот готовая программа:

```
var count: array[1..26] of integer;
    letters: array[1..26] of char;
    s: string;
    temp: char;
    i, k: integer;
begin
   for i:=1 to 26 do begin
     count[i]:=0;
     letters[i]:=Chr(Ord('a')+i-1);
   end;
```

```
    { чтение данных, подсчёт начальных букв }
  while True do begin
    readln(s);
    if (Length(s) > 0) and (s[1] = '*') then break;
    s := ' ' + s;
    for i:=1 to Length(s)-1 do
      if (s[i] = ' ') and (s[i+1] in ['a'..'z']) then begin
        k:=Ord(s[i+1]) - Ord('a') + 1;
        count[k] := count[k] + 1;
      end;
  end;
    { сортировка }
  for i:=1 to 26 do begin
    for k:=25 downto i do begin
      if count[k] < count[k+1] then begin
        i:= count[k]; count[k]:= count[k+1]; count[k+1]:= i;
        temp:= letters[k]; letters[k]:=letters[k+1];
        letters[k+1]:=temp;
      end;
    end;
  end;
    { вывод результата }
  for i:=1 to 26 do
    if count[i] > 0 then
      writeln(letters[i], ' ', count[i]);
end.
```

Теперь рассмотрим второй вариант ввода данных – посимвольное чтение. Начало слова – это ситуация, когда предыдущий символ (в программе он хранится в переменной **c1**) – пробел, а следующий за ним – строчная латинская буква. В этом фрагменте **c** и **c1** – переменные типа **char**:

```
c1:=' ';
repeat
  read(c);
  if (c1 = ' ') and (c in ['a'..'z']) then begin
    k:=Ord(c) - Ord('a') + 1;
    count[k] := count[k] + 1;
  end;
  c1 := c;
  if c in [#10,#13] then c1 := ' '; { перевод строки }
until c = '*';
```

Первая строчка в этом фрагменте нужна для того, чтобы обработать самое первое слово, перед которым может не быть пробелов – мы искусственно добавляем первый пробел, записывая его в переменную **c1**.

Строчка

```
  if c in [#10,#13] then c1 := ' '; { перевод строки }
```

предназначена для того, чтобы правильно обработать переход на новую строку (при этом во входной поток поступают символы с десятичными кодами 13 и 10). Следующая строка может начинаться сразу с латинской буквы, а не с пробела, поэтому после чтения символа с кодом 10 или 13 в переменную **c1** нужно записать пробел.

Вот полная программа:

```
var count: array[1..26] of integer;
    letters: array[1..26] of char;
    s: string;
    c, c1, temp: char;
    i, k: integer;
begin
   for i:=1 to 26 do begin
     count[i]:=0;
     letters[i]:=Chr(Ord('a')+i-1);
   end;
     { чтение данных, подсчёт начальных букв }
   c1:=' ';
   repeat
     read(c);
     if (c1 = ' ') and (c in ['a'..'z']) then begin
       k:=Ord(c) - Ord('a') + 1;
       count[k] := count[k] + 1;
     end;
     c1 := c;
     if c in [#10,#13] then c1 := ' '; { перевод строки }
   until c = '*';
     { сортировка }
   for i:=1 to 26 do begin
     for k:=25 downto i do begin
       if count[k] < count[k+1] then begin
         i:= count[k]; count[k]:= count[k+1]; count[k+1]:= i;
         temp:= letters[k]; letters[k]:=letters[k+1];
         letters[k+1]:=temp;
       end;
     end;
   end;
     { вывод результата }
   for i:=1 to 26 do
     if count[i] > 0 then
       writeln(letters[i], ' ', count[i]);
end.
```

Наконец, можно использровать массив структур вместо двух массивов **count** и **letters**:

```
type structW = record
       count: integer;
       letter: char;
     end;
var W: array[1..26] of structW;
    temp: structW;
    c, c1: char;
    i, k: integer;
begin
   for i:=1 to 26 do begin
     W[i].count:=0;
     W[i].letter:=Chr(Ord('a')+i-1);
   end;
```

```
      { чтение данных, подсчёт начальных букв }
    c1 := ' ';
    repeat
      read(c);
      if (c1 = ' ') and (c in ['a'..'z']) then begin
        k:=Ord(c) - Ord('a') + 1;
        W[k].count := W[k].count + 1;
      end;
      c1 := c;
      if c in [#10,#13] then c1 := ' '; { перевод строки }
    until c = '*';
      { сортировка }
    for i:=1 to 26 do begin
      for k:=25 downto i do begin
        if W[k].count < W[k+1].count then begin
          temp:= W[k]; W[k]:=W[k+1]; W[k+1]:=temp;
        end;
      end;
    end;
      { вывод результата }
    for i:=1 to 26 do
      if W[i].count > 0 then
        writeln(W[i].letter, ' ', W[i].count);
end.
```

Кроме того, **Н.М. Айзикович** (**лицей №410, г. Санкт-Петербург**) предложил идею решения, использующего только один массив. Напомним, что проблема в том, что нужно вывести все ненулевые счетчики для букв в порядке возрастания. Для этого можно не сортировать массив **count**, а поступить следующим образом:

1) определить, с какой буквы начиналось наибольшее количество слов: найти максимальное значение счетчика и записать его в переменную **max** (в этом решении используется массив **count** с целочисленными индексами):

```
max:= count[1];
for i:=2 to 26 do
  if count[i] > max then max:= count[i];
```

2) затем в цикле перебираем (в переменной **k**) возможные значения счетчиков в порядке убывания, от **max** до 1; для каждого из таких значений проходим весь массив счетчиков и выводим те счетчики, которые равны k, и соответствующие буквы:

```
for k:=max downto 1 do
  for i:=1 to 26 do
    if count[i] = k then
      writeln(Chr(Ord('a')+i-1), ' ', count[i]);
```

Здесь запись **Chr(Ord('a')+i-1)** служит для того, чтобы получить символ с номером **i** в латинском алфавите (определить код буквы **a**, добавить к нему **i** и вычесть 1, поолучить символ с полученным таким образом кодом).

Таким образом, массив **letters** в этом варианте решения не нужен. Структуры – тоже. Фактически сортировка выполнена «на месте», без перестановки элементов, только при выводе. Приведем полное решение:

```
var count: array[1..26] of integer;
    s: string;
```

```
    temp: char;
    i, k, max: integer;
begin
   for i:=1 to 26 do count[i]:=0;
   while True do begin
     readln(s);
     if (Length(s) > 0) and (s[1] = '*') then break;
     s := ' ' + s;
     for i:=1 to Length(s)-1 do
       if (s[i] = ' ') and (s[i+1] in ['a'..'z']) then begin
         k:=Ord(s[i+1]) - Ord('a') + 1;
         count[k] := count[k] + 1;
       end;
   end;
   max:= count[1];
   for i:=2 to 26 do
     if count[i] > max then max:= count[i];
   for k:=max downto 1 do
     for i:=1 to 26 do
       if count[i] = k then
          writeln(Chr(Ord('a')+i-1), ' ', count[i]);
end.
```

Ещё одно решение предложила **О.В. Алимова (СПбГУ)**. В нём используется один массив счётчиков с символьными индексами. Поскольку по условию в тексте нет никаких символов, кроме строчных английских букв, пробелов и завершающего знака ‘*', можно не делать дополнительные проверки (например, на заглавные буквы, цифры и т.п.). Сначала заполняем нулями массив счётчиков:

```
for c := 'a' to 'z' do mas[c] := 0;
```

Затем читаем и обрабатываем строки, заканчивая на строке, содержащей одну звездочку:

```
readln(s);
while s <> '*' do begin
  { обработать строку }
  readln(s);
end;
```

Обработка строки сводится к следующему:

- добавляем пробел в конец строки

```
s := s + ' ';
```

- удаляем все сдвоенные пробелы и пробел в начале строки, если он есть:

```
while pos('  ',s) <> 0 do
  delete(s,pos('  ',s),1);
if pos(' ',s) = 1 then delete(s,1,1);
```

- теперь первый символ строки – это первая буква первого слова (или строка пустая, если в ней не было слов)
- в цикле делаем следующее: ищем пробел следующий за словом, увеличиваем счётчик для буквы, с которой начинается первое слово и вырезаем из строки слово вместе с следующим за ним пробелом; попутно (если нужно) изменяем

значение переменной **max**, в которой хранится максимальное на данный момент значение счётчика:

```
k := pos(' ',s);
while k <> 0 do begin
  mas[s[1]] := mas[s[1]] + 1;
  if mas[s[1]] > max then
    max := mas[s[1]];
  delete(s,1,k);
  k := pos(' ',s);
end;
```

- этот цикл заканчивается, когда слова кончились и очередной пробел не найден (**k=0**)

Приведем программу полностью:

```
var mas: array['a'..'z'] of integer;
    max,i,k:integer;
    c:char;
    s:string;
BEGIN
  for c := 'a' to 'z' do mas[c] := 0;
  max := 0;
  readln(s);
  while s <> '*' do begin
    s := s + ' ';
    while pos('  ',s) <> 0 do
      delete(s,pos('  ',s),1);
    if pos(' ',s) = 1 then delete(s,1,1);
    k := pos(' ',s);
    while k <> 0 do begin
      mas[s[1]] := mas[s[1]] + 1;
      if mas[s[1]] > max then
        max := mas[s[1]];
      delete(s,1,k);
      k := pos(' ',s);
    end;
    readln(s);
  end;
  for k:= max downto 1 do
    for c := 'a' to 'z' do
      if mas[c] = k then writeln(c:4, mas[c]:4);
END.
```

42) На вид это простая задача, но все дело портит то, что числа могут быть и положительные и отрицательные. Заметьте, что согласно условию нулевых значений быть не может. Поэтому возможны три варианта:

1) все числа положительные, при этом минимальное произведение – это произведение двух минимальных из введенных значений, как в примере из условия; заметим, что эти два минимальных могут быть равны;
2) все числа отрицательные, при этом все произведения двух чисел положительны, и минимальное произведение – это произведение двух минимальных по модулю из введенных значений, причем эти два числа могут быть равны;

3) среди чисел есть положительные и отрицательные; при этом минимальное произведение отрицательно и равно произведению максимального положительного числа на минимальное отрицательное.

Поэтому нам нужно определить

1) минимальное и максимальное числа; если они разного знака, то ответ – их произведение;
2) два минимальных числа: на случай, если все числа положительны;
3) два максимальных числа: на случай, если все числа отрицательные.

По условию частиц не меньше двух. Поэтому после ввода количества частиц (**N**) можно прочитать первые две скорости и найти минимальную и максимальную из них (переменные **min** и **max**), а также второй минимум и второй максимум (переменные **min2** и **max2**):

```
readln(min);
readln(max);
if min < max then begin
   min2:= max; max2:= min;
end
else begin
   min2:= min; max2:= max;
   min:= max2; max:= min2;
end;
```

Здесь с помощью условного оператора мы меняем местами значения **min** и **max**, если они вводились в порядке убывания, и «расставляем» второй минимум и второй максимум соответствующим образом.

Теперь читаем остальные **N-2** скорости. Именно в этом цикле чтения и будет происходить основная обработка данных. Сразу, за 1 проход, ищем два минимальных и два максимальных (см. первую разобранную задачу в файле **C4.doc**):

```
if v < min then begin          { два минимальных }
  min2 := min; min := v;
end
else if v < min2 then min2 := v;
if v > max then begin         { два максимальных }
  max2 := max; max := v;
end
else if v > max2 then max2 := v;
```

Вывести нужно минимальное из значений **min*min2**, **max*max2** и **min*max**:

```
pMin := min*min2;
if max*max2 < pMin then pMin := max*max2;
if max*min  < pMin then pMin := min*max;
writeln(pMin);
```

Вот полная программа:

```
var N, i, v, pMin, min, min2, max, max2: integer;
    neg: boolean;
begin
  readln(N);
  readln(min);
  readln(max);
```

```
    if min < max then begin
       min2 := max; max2 := min;
    end
    else begin
       min2 := min; max2 := max;
       min := max2; max := min2;
    end;
    for i := 1 to N-2 do begin
      readln(v);
      if v < min then begin
        min2 := min; min := v;
      end
      else if v < min2 then min2 := v;
      if v > max then begin
        max2 := max; max := v;
      end
      else if v > max2 then max2 := v;
    end;
    pMin := min*min2;
    if max*max2 < pMin then pMin := max*max2;
    if max*min  < pMin then pMin := max*min;
    writeln(pMin)
  end.
```

Заметим, что возможен более простой вариант этой задачи: найти *максимальное* произведение. Она решается еще проще: нужно найти два максимуму и два минимума, так же, как и в этой задаче, и найти максимальное из произведений **min*min2**, **max*max2**.

Ещё одно решение предложил **Goolkin's Nose**:

Минимальное произведение равно произведению минимального отрицательного и максимального положительного чисел, либо двух околонулевых. В этом случае, чтобы не объявлять пару лишних переменных для околонулевых отрицательных/положительных, просто найдем по модулю два максимально близких к нулю числа **abs1** и **abs2**. Также ищем минимальное отрицательное и максимальное положительное в переменных **min** и **max**. Если присутствуют нужные **min** и **max**, перемножаем их. Иначе перемножаем **abs1**, **abs2**.

```
  var
    i, input, n: integer;
    min, max, abs1, abs2: integer;
  begin
    min:=10002;max:=-10002;
    abs1:=10002; abs2:=10002;
    readln(n);
    for i:=1 to n do begin
      readln(input);
      if input>max then max:=input;
      if input<min then min:=input;
      if abs(input)<=abs(abs1) then begin
        abs2:=abs1;
        abs1:=input;
      end else if abs(input)<=abs(abs2) then
```

```
        abs2:=input;
    end;
    if (min>0)or(max<0) then writeln('+', abs1*abs2)
    else writeln(min*max);
end.
```

43) По условию задачи нужно найти минимальную четную сумму из всех сумм пар введенных значений. Подумаем, как может получиться четная сумма при сложении двух чисел. Возможны два варианта:

1) четное + четное;
2) нечетное + нечетное.

Кроме того, если четной суммы нет (это значит, что введено всего два числа, одно четное, второе – нечетное), нужно вывести их сумму, которая одновременно будет минимальной. Таким образом, можно сделать важный вывод: если **N=2**, нужно просто вывести сумму двух последующих введенных чисел.

Если чисел больше 2, нужно хранить в памяти и постоянно корректировать два минимальных четных числа (для этого будем использовать переменные **ch1** и **ch2**) и два минимальных нечетных числа (переменные **nch1** и **nch2**); тогда результат – это минимальная из сумм **ch1+ch2** и **nch1+nch2**.

Итак, сначала читаем их входного потока количество чисел **N** и сразу первые два числа последовательности в переменные **v** и **v1**:

```
readln(N);
readln(v);
readln(v1);
```

Записываем в переменные **ch1**, **ch2**, **nch1** и **nch2** число 60001, которое больше 60000, поэтому любое число входной последовательности при сравнении окажется меньше этого начального значения:

```
ch1:=60001; ch2:=60001;
nch1:=60001; nch2:=60001;
```

Кроме того, сумма любых двух чисел из реальной последовательности меньше, чем каждое из начальных значений (спасибо за замечание *С. Давыдову*).

Важно: здесь предполагается, что число 60001 помещается в ячейки **ch1**, **ch2**, **nch1** и **nch2**. Например, можно использовать тип **longint**, если в вашей версии транслятора переменная типа **integer** имеет размер 2 байта.

Теперь нужно записать числа, находящиеся в переменных **v** и **v1**, в соответствующие рабочие переменные: **ch1**, **ch2**, **nch1** и **nch2**. Если оба числа четных, то есть остатки от их деления на 2 равны 0, записываем их в **ch1** и **ch2** в порядке возрастания (меньшее – в **ch1**).

```
if (v mod 2 = 0) and (v1 mod 2 = 0) then begin
  if v < v1 then begin
      ch1 := v; ch2 := v1;
  end
  else begin
      ch1 := v1; ch2 := v;
  end;
end;
```

Аналогично нечётные числа записываем их в **nch1** и **nch2** в порядке возрастания (меньшее – в **nch1**). Если одно число четное, а второе – нечетное, записываем их в **ch1** и **nch1**:

```
if v mod 2 <> v1 mod 2 then begin
```

```
      if v mod 2 = 0 then begin
         ch1 := v; nch1 := v1;
      end
      else begin
         nch1 := v; ch1 := v1;
      end;
   end;
```

Теперь в цикле читаем остальные **N-2** чисел входной последовательности:

```
   for i := 1 to N-2 do begin
     readln(v);
     ...          { обработать v }
   end;
```

Обработка сводится к тому, что мы определяем, четное число прочитано в **v** или нет, а затем ищем, соответственно среди четных или нечетных чисел, два минимума (см. первую разобранную задачу в файле **C4.doc** и решение задачи 42 выше):

```
   if v mod 2 = 0 then begin { если введено четное число }
     if v < ch1 then begin
        ch2 := ch1; ch1 := v;
     end
     else if v < ch2 then ch2 := v;
   end
   else { если введено нечетное число }
     if v < nch1 then begin
        nch2 := nch1; nch1 := v;
     end
     else if v < nch2 then nch2 := v;
```

Остается вывести результат. Если **N=2**, то после окончания цикла выводим просто сумму **v+v1** (сумму первых двух введенных чисел). Иначе выбираем минимальное из **ch1+ch2** и **nch1+nch2**:

```
   if N = 2 then writeln(v + v1)
   else
     if ch1+ch2 < nch1+nch2 then
          writeln(ch1+ch2)
     else writeln(nch1+nch2)
```

Заметим, что вывод работает верно и в том случае, когда введено одно нечетное число и два больших четных. Например (**N=3**):

```
15
29000
29000
```

В этом случае значения переменных будут такие:

```
ch1 = 29000        ch2 = 29000
nch1 = 15          nch2 = 60001
```

Видим, что в **nch2** осталось начальное значение, равное 60001, что больше суммы двух любых правильных чисел входной последовательности. Поэтому имеем

ch1+ch2 = 29000+29000 = 58000 < nch1+nch2 = 15+60001 = 60016

В результате будет выведен правильный ответ 58000.

Вот полная программа:

```
   var N, i, v, v1, ch1, ch2, nch1, nch2: integer;
   begin
     readln(N);
     readln(v);
     readln(v1);
```

```
    ch1:=60001; ch2:=60001;
    nch1:=60001; nch2:=60001;
    if (v mod 2 = 0) and (v1 mod 2 = 0) then begin
       if v < v1 then begin
          ch1 := v; ch2 := v1;
       end
       else begin
          ch1 := v1; ch2 := v;
       end;
    end;
    if (v mod 2 = 1) and (v1 mod 2 = 1) then begin
       if v < v1 then begin
          nch1 := v; nch2 := v1;
       end
       else begin
          nch1 := v1; nch2 := v;
       end;
    end;
    if v mod 2 <> v1 mod 2 then begin
       if v mod 2 = 0 then begin
          ch1 := v; nch1 := v1;
       end
       else begin
          nch1 := v; ch1 := v1;
       end;
    end;
    for i := 1 to N-2 do begin
      readln(v);
      if v mod 2 = 0 then begin
        if v < ch1 then begin
          ch2 := ch1; ch1 := v;
        end
        else if v < ch2 then ch2 := v;
      end
      else
        if v < nch1 then begin
          nch2 := nch1; nch1 := v;
        end
        else if v < nch2 then nch2 := v;
    end;
    if N = 2 then writeln(v + v1)
    else
      if ch1+ch2 < nch1+nch2 then
           writeln(ch1+ch2)
      else writeln(nch1+nch2)
  end.
```

Однако это решение можно еще упростить (этот вариант предложил **А. Тарасов**, МОБУ СОШ №3 с. Красноусольский Республики Башкортостан). Дело в том, что условные операторы, которые обрабатывают особый случай при N = 2, можно просто убрать из программы. Вот что получится:

```
  var N, i, v, ch1, ch2, nch1, nch2: integer;
  begin
```

```
    ch1:=60001; ch2:=60001;
    nch1:=60001; nch2:=60001;
    readln(N);
    for i := 1 to N do begin
      readln(v);
      if v mod 2 = 0 then begin
        if v < ch1 then begin
          ch2 := ch1; ch1 := v;
        end
        else if v < ch2 then ch2 := v;
      end
      else
        if v < nch1 then begin
          nch2 := nch1; nch1 := v;
        end
        else if v < nch2 then nch2 := v;
    end;
    if ch2 = nch2 then writeln(ch1+nch1)
    else
      if ch1+ch2 < nch1+nch2 then
          writeln(ch1+ch2)
      else writeln(nch1+nch2)
  end.
```

Действительно, если мы ввели только два чётных числа, они окажутся в переменных **ch1** и **ch2**, и их сумма будет заведомо меньше, чем **nch1+nch2=60001+60001**, при этом выводится нужная сумма **ch1+ch2**. Аналогичная ситуация возникает при вводе двух нечётных чисел.

Если же введено одно чётное число и одно нечётное, то в переменных **ch2** и **nch2** останутся равные начальные значения 60001, в этом случае просто выводим выводим сумму **ch1+nch1**.

44) Уточним условие задачи «по-русски». Есть аббревиатура, состоящая из трех символов, это начальные буквы фамилии, имени и отчества (ФИО). Есть список в формате

<фамилия> <имя> <отчество>

в котором есть не более 10 подходящих адресатов, причем адресаты в списке могут повторяться. Нужно вывести данные всех людей, ФИО которых подходит под аббревиатуру, в порядке убывания частоты встречаемости в этом списке.

Мы знаем, что если требуется сортировка (или вывод списка в определенном порядке), как правило, нужно использовать массивы. Намёк на это есть в условии, где сказано, что подходящих адресатов не более 10. Вводим массив для хранения подходящих ФИО и соответствующие счетчики:

```
  const MAX = 10;
  var FIO: array[1..MAX] of string;
      count: array[1..MAX] of integer;
```

Выделим в памяти место для переменных

```
  var i, k, N, nFIO: integer;
      abbr, s: string;
```

Здесь

N – количество элементов списка (вводится в самом начале),

nFIO – количество найденных подходящих личностей,

abbr – заданная аббревиатура,
остальные переменные – вспомогательные.
В начале программы читаем аббревиатуру и размер списка:

```
readln(abbr);
readln(N);
```

и обнуляем счетчики:

```
nFIO := 0;
for i:=1 to MAX do count[i] := 0;
```

Далее следует основной цикл

```
for i:=1 to N do begin
  readln(s);
  если аббревиатура совпадает с abbr то
    искать ФИО в массиве FIO
    если не нашли то
      добавить в массив FIO
    иначе
      увеличить счётчик для этого ФИО
    все
  все
end;
```

Здесь все строки, записанные синим цветом – это псевдокод, который нужно перевести на язык программирования.
Для построения аббревиатуры, соответствующей введенной строке, удобно использовать функцию, которая принимает строку и возвращает аббревиатуру. Для этого нужно «сцепить» первые буквы всех слов в строке.
Начало слова удобно определить по сочетанию символов «пробел, а за ним – не пробел». Для того, чтобы не рассматривать отдельно первый символ (перед которым нет пробела), добавим один пробел в начало строки. Аббревиатуру собираем в локальной переменной **a**:

```
function Abbrev(s: string): string;
var i: integer;
    a: string;
begin
  s := ' ' + s;    { добавляем пробел в начало }
  a := '';         { пустая строка }
  for i:=2 to Length(s) do
    if (s[i-1] = ' ') and (s[i] <> ' ') then { начало слова }
      a := a + s[i];
  Abbrev := a;     { вернуть результат функции }
end;
```

Итак, предположим, что аббреавиатура совпадает. Тогда ищем только что введенную строку среди первых **nFIO** элементов массива **FIO**:

```
k := 1;
while (k <= nFIO) and (s <> FIO[k]) do
  k := k + 1;
```

Этот цикл останавливается в двух случаях: если строки нет в массиве (новая персона) – при этом **k>nFIO**, и если такая строка найдена – при этом **k<=nFIO**. В первом случае нужно добавить строку в массив **FIO** и записать в соответствующий счетчик число 1 (встретили в первый раз), а во втором нужно просто увеличить счетчик (соответствующий элемент массива **count**):

```
if k > nFIO then begin
  nFIO := nFIO + 1;
```

```
    FIO[nFIO] := s;
    count[nFIO] := 1;
  end
  else count[k] := count[k] + 1;
```

Теперь остается вывод на экран в порядке убывания частоты встречаемости. Конечно, можно отсортировать массив **FIO** с помощью любого алгоритма, не забывая переставлять соответствующие значения счетчиков в массиве **count**. Но можно обойтись без перестановки. Для этого придётся ввести еще одну переменную **maxCount** – в ней будем хранить максимальное значение из всех счётчиков массива **count**. До основного цикла эта переменная обнуляется, а при каждом изменении какого-либо счётчика проверяем, не нужно ли менять и **maxCount**:

```
  if count[k] > maxCount then
    maxCount:= count[k];
```

После цикла сначала выводим все ФИО, которые встречаются **maxCount** раз (больше не может быть!), затем те, которые встречаются **maxCount-1** раз и т.д.:

```
  for i:=maxCount downto 1 do
    for k:=1 to MAX do
      if count[k] = i then
        writeln(FIO[k], ' ', count[k]);
```

Вот полная программа:

```
const MAX = 10;
var FIO: array[1..MAX] of string;
    count: array[1..MAX] of integer;
    i, k, N, nFIO, maxCount: integer;
    abbr, s: string;
{ функция, которая определяет аббревиатуру }
function Abbrev(s: string): string;
var i: integer;
    a: string;
begin
  s := ' ' + s;
  a := '';
  for i:=2 to Length(s) do
    if (s[i-1] = ' ') and (s[i] <> ' ') then
      a := a + s[i];
  Abbrev := a;
end;
{ начало основной программы }
begin
  readln(abbr);
  readln(N);
  nFIO := 0;
  for i:=1 to MAX do count[i] := 0;
  maxCount:= 0;
  for i:=1 to N do begin
    readln(s);
    if abbr = Abbrev(s) then begin
       k := 1;
       while (k <= nFIO) and (s <> FIO[k]) do
         k := k + 1;
       if k > nFIO then begin
         nFIO := nFIO + 1;
```

```
          FIO[nFIO] := s;
          count[nFIO] := 1;
        end
        else count[k] := count[k] + 1;
        if count[k] > maxCount then
          maxCount:= count[k];
    end;
  end;
  for i:=maxCount downto 1 do
    for k:=1 to nFIO do
      if count[k] = i then
        writeln(FIO[k], ' ', count[k]);
end.
```

Приведем также решение с сортировкой массива **count** по убыванию (**М.Г. Можаева, г. Череповец**). Одновременно нужно переставлять и элементы массива **FIO**, чтобы не «отрывать» ФИО от числа повторений:

```
  for i:=1 to nFIO-1 do
    for j:=1 to nFIO-i do
      if count[j] < count[j+1] then begin
        s:= FIO[j]; FIO[j]:= FIO[j+1]; FIO[j+1]:= s;
        p:= count[j]; count[j]:= count[j+1]; count[j+1]:= p;
      end;
```

Обратите внимание, что сортируются только первые **nFIO** элементов, остальные нам не нужны. Несмотря на то, что алгоритм сортировки имеет квадратичную сложность (по **nFIO**), снижать баллы за неэффективность было бы неправильно. Дело в том, что количество операций этого вложенного цикла не зависит от **N** и вообще довольно невелико: не более 45 шагов внутреннего цикла. Вот полная программа:

```
const MAX = 10;
var FIO: array[1..MAX] of string;
    count: array[1..MAX] of integer;
    i, j, k, p, N, nFIO: integer;
    abbr, s: string;
{ функция, которая определяет аббревиатуру }
function Abbrev(s: string): string;
var i: integer;
    a: string;
begin
  s := ' ' + s;
  a := '';
  for i:=2 to Length(s) do
    if (s[i-1] = ' ') and (s[i] <> ' ') then
      a := a + s[i];
  Abbrev := a;
end;
{ начало основной программы }
begin
  readln(abbr);
  readln(N);
  nFIO := 0;
  for i:=1 to MAX do count[i] := 0;
  for i:=1 to N do begin
```

```
      readln(s);
      if abbr = Abbrev(s) then begin
         k := 1;
         while (k <= nFIO) and (s <> FIO[k]) do
           k := k + 1;
         if k > nFIO then begin
           nFIO := nFIO + 1;
           FIO[nFIO] := s;
           count[nFIO] := 1;
         end
         else count[k] := count[k] + 1;
      end;
    end;
      { сортировка массивов count и FIO }
    for i:=1 to nFIO-1 do
      for j:=1 to nFIO-i do
        if count[j] < count[j+1] then begin
          s:= FIO[j]; FIO[j]:= FIO[j+1]; FIO[j+1]:= s;
          p:= count[j]; count[j]:= count[j+1]; count[j+1]:= p;
        end;
      { вывод результата после сортировки }
    for k:=1 to nFIO do
       writeln(FIO[k], ' ', count[k]);
  end.
```

45) В этой задаче нужно помнить время освобождения каждой ячейки, поэтому нужно завести массив. Его размер задан в условии – может быть не более 1000 ячеек:

```
const MAXCELLS = 1000;
var pass: array [1..MAXCELLS] of integer;
```

В начале работы программы этот массив (точнее, первые K элементов, где K – число ячеек) нужно заполнить нулями (все ячейки свободны):

```
for i:=1 to K do
  cellFreeTime[i] := 0;
```

Основной цикл можно записать на псевдокоде так:

```
for i:=1 to N do begin
  Readln(s); { прочитать строку }
  { выделить фамилию, начальное и конечное время }
  { найти свободную ячейку }
  if ячейка найдена then begin
    { вывести фамилию и номер ячейки }
  end
end;
```

Сразу возникает вопрос – как хранить время? Конечно, можно записывать в массив число минут, прошедших с начала дня. Тогда время «09:12» будет храниться как число 9*60+12=552. Однако в этой конкретной задаче нам не нужно выполнять вычисления с данными типа «время», поэтому можно просто вырезать из строки двоеточие (получить «0912» и перевести полученную запись в целое число 912 с помощью процедуры **Val**.

Тогда процедура ParseData, которая выделяет из строки фамилию, начальное и конечное время, может выглядеть так:

```
procedure ParseData(var s: string;
                    var startTime, endTime: integer);
var p, r: integer;
    sStartTime, sEndTime: string;
begin
  p := Pos(' ', s); { найти первый пробел }
  sStartTime := Copy(s, p+1, 5); { начальное время – строка }
  Delete(sStartTime, 3, 1);      { удалить двоеточие }
  Val(sStartTime, startTime, r); { преобразовать в число }
  sEndTime   := Copy(s, p+7, 5); { конечное время – строка }
  Delete(sEndTime, 3, 1);        { удалить двоеточие }
  Val(sEndTime, endTime, r);     { преобразовать в число }
  s := Copy(s, 1, p-1);          { оставить только фамилию }
end;
```

В этой процедуре все параметры объявлены с ключевым словом var, это значит, что они изменяемые и с их помощью процедура возвращает результаты работы в вызывающую программу. Строка **s** – это и входной параметр (строка, прочитанная из файла), и один из результатов – фамилия пассажира. В переменны **startTime** и **endTime** процедура возвращает начальное и конечное время (целые числа).

Поиск первой свободной ячейки оформим в виде функции, которая возвращает номер выделенной ячейки:

```
function FindCell(startTime, endTime: integer): integer;
var cellNo: integer;
begin
  FindCell := 0;
  for cellNo:=1 to K do
    if cellFreeTime[cellNo] <= startTime then begin
       FindCell := cellNo;
       cellFreeTime[cellNo] := endTime;
       break;
    end;
end;
```

Эта функция перебирает все K ячеек, начиная с первой, и ищет такую, для которой время освобождения меньше или равно времени прибытия очередного пассажира (параметр **startTime**). Вспомним, что если ячейка ни разу не была занята, в соответствующем элементе массива будет записан 0, и любое реальное время **startTime** будет не меньше, чем это значение (ячейка будет выбрана).

Если свободная ячейка найдена, в переменную **FindCell** записывается результат функции – номер ячейки, а в соответствующий элемент массива **cellFreeTime** записывается время освобождения ячейки. Поскольку остальные ячейки уже не нужны, цикл прерывается с помощью оператора **break**.

Если свободной ячейки нет, в переменной **FindCell** остается значение 0, записанное в первой строке функции.

Теперь можно собрать всю программу:

```
program qq;
const MAXCELLS = 1000;
var cellFreeTime: array[1..MAXCELLS] of integer;
    N, K, i, cellNo, startTime, endTime: integer;
```

```
    s: string;
{ Разбор входной строки }
procedure ParseData(var s: string;
                    var startTime, endTime: integer);
var p, r: integer;
    sStartTime, sEndTime: string;
begin
  p := Pos(' ', s);
  sStartTime := Copy(s, p+1, 5);
  sEndTime   := Copy(s, p+7, 5);
  Delete(sStartTime, 3, 1);
  Val(sStartTime, startTime, r);
  Delete(sEndTime, 3, 1);
  Val(sEndTime, endTime, r);
  s := Copy(s, 1, p-1);
end;
{ Поиск первой свободной ячейки }
function FindCell(startTime, endTime: integer): integer;
var cellNo: integer;
begin
  FindCell := 0;
  for cellNo:=1 to K do
    if cellFreeTime[cellNo] <= startTime then begin
       FindCell := cellNo;
       cellFreeTime[cellNo] := endTime;
       break;
    end;
end;
{ Основная программа }
begin
  Readln(N);
  Readln(K);
  for i:=1 to K do cellFreeTime[i] := 0;
  for i:=1 to N do begin
     Readln(s);
     ParseData(s, startTime, endTime);
     cellNo := FindCell(startTime, endTime);
     if cellNo > 0 then
       writeln(s, ' ', cellNo);
  end;
end.
```

Заметим, что эта программа выводит данные по ходу дела – вводит одну строки и сразу выводит результат, елси найдена свободная ячейка. Если требуется, чтобы сначала программа ввела все данные, а потом выдала все результаты, на каждом шаге цикла строки, предназначенные для вывода нужно запоминать в массиве. Так как по условию число пассажиров не превышает 1000, размер этого массива выбираем равным 1000. Переменная **count** – это счётчик пассажиров, которые использовали камеру хранения.

```
program Passengers;
const MAXPASS = 1000;
      MAXCELLS = 1000;
var pass: array[1..MAXPASS] of string;
    cellFreeTime: array[1..MAXCELLS] of integer;
```

```
    N, K, i, cellNo, count, startTime, endTime: integer;
    s, name, sCell: string;
{ Разбор входной строки }
procedure ParseData(var s: string;
                    var startTime, endTime: integer);
var p, r: integer;
    sStartTime, sEndTime: string;
begin
  p := Pos(' ', s);
  sStartTime := Copy(s, p+1, 5);
  sEndTime   := Copy(s, p+7, 5);
  Delete(sStartTime, 3, 1);
  Val(sStartTime, startTime, r);
  Delete(sEndTime, 3, 1);
  Val(sEndTime, endTime, r);
  s := Copy(s, 1, p-1);
end;
{ Поиск первой свободной ячейки }
function FindCell(startTime, endTime: integer): integer;
var cellNo: integer;
begin
  FindCell := 0;
  for cellNo:=1 to K do
    if cellFreeTime[cellNo] <= startTime then begin
       FindCell := cellNo;
       cellFreeTime[cellNo] := endTime;
       break;
    end;
end;
{ Основная программа }
begin
  Readln(N);
  Readln(K);
  for i:=1 to K do cellFreeTime[i] := 0;
  count := 0;
  for i:=1 to N do begin
     Readln(s);
     ParseData(s, startTime, endTime);
     cellNo := FindCell(startTime, endTime);
     if cellNo > 0 then begin
       Inc(count);          { увеличить счётчик }
       Str(cellNo, sCell); { преобразовать в строку }
       pass[count] := s + ' ' + sCell;{ запомнить в массиве }
     end;
  end;
  { вывод результата }
  for i:=1 to count do
    writeln(pass[i]);
end.
```

Ещё один вариант решения, более короткий, предложила **М.А. Зайцева** (лицей № 1580 при МГТУ им. Н.Э.Баумана). Идея состоит в том, чтобы хранить и сравнивать моменты времени как символьные строки, без преобразования. Действительно, цифры в кодовой таблице ASCII расположены подряд по возрастанию, так что

'00:00' < '00:01' < ...'23:58' < '23:59'

Поэтому не нужно переводить время в формат целого числа, и анализ строки упрощается:

```
p:= Pos(' ', s);
fam:= Copy(s,1,p-1); { выделяем фамилию }
Delete(s,1,p);      { удаляем ее вместе с пробелом }
startTime:= Copy(s,1,5); { время сдачи багажа }
endTime:= Copy(s,7,5);   { время выдачи багажа }
```

Дальше остается только найти ячейку, у которой время освобождения меньше, чем время сдачи текущего багажа startTime.

Вот полная программа:

```
const MAXPASS = 1000;
      MAXCELLS = 1000;
var
  N, K: integer;
  i, p, cellNo, count: integer;
  s, fam, sCell: string;
  startTime, endTime: string;
  cellFreeTime: array [1..MAXCELLS] of string;
  pass: array[1..MAXPASS] of string;
begin
  readln(N);
  readln(K);
    { записываем нулевые значения времени }
  for cellNo:= 1 to k do
     cellFreeTime[cellNo]:='00:00';
  count:= 0;
  for i:= 1 to N do begin
    readln(s);
      { разбор строки }
    p:= Pos(' ', s);
    fam:= Copy(s,1,p-1); { выделяем фамилию }
    Delete(s,1,p);       { удаляем ее вместе с пробелом }
    startTime:= copy(s,1,5); { время сдачи багажа }
    endTime:= copy(s,7,5);   { время выдачи багажа }
      { поиск свободной ячейки }
    cellNo:= 1;
    while cellNo <= k do begin
      if startTime >= cellFreeTime[cellNo] then begin
        cellFreeTime[cellNo]:= endTime;
        Inc(count);
        Str(cellNo, sCell);
        pass[count] := fam + ' ' + sCell;
        break;
      end;
      Inc(cellNo);
    end;
end;
    { вывод результата }
  for i:=1 to count do
    writeln(pass[i]);
end.
```

46) Нам нужно определить площадь треугольника, одно из оснований которого лежит на оси OX. Пусть длина этого основания равна B, а высота треугольника равна H. Тогда площадь треугольника равна S = B·H/2.

Из этих рассуждений следует, что треугольник с максимальной площадью образован двумя точками на оси OX, которые дальше всего остоят друг от друга, и точкой, которая наиболее удалена от оси OX, то есть имеет максимальную координату y (по модулю).

Таким образом, нужно найти:

1) **xMin** – минимальное значение x-координаты среди всех точек, для которых y-координата равна нулю;
2) **xMax** – максимальное значение x-координаты среди всех точек, для которых y-координата равна нулю;
3) **yMax** – максимальный модуль y-координаты среди всех точек.

Тогда S = (**xMax** – **xMin**)·**yMax**/2.

Единственная сложность состоит в том, чтобы записать в переменные **xMax** и **xMin** некоторые начальные значения, которые позволят определить, что еще ни одной точки на оси OX не найдено. Например, можно записать в них два нуля, но как тогда различить ситуации «ни одна точка на оси OX не найдена» и «найдена одна точка на оси OX с координатой x=0»? Можно ввести еще одну логическую переменную (назовём её **Found**), которая сначала равна **False** (ни одна точка на оси OX не найдена), и получает значение **True**, когда такая точка найдена.

```
program Treug;
var i, N, x, y: integer;
    xMin, xMax, yMax: integer;
    Found: boolean;
begin
  Found:= False;
  xMin := 0; xMax := 0; yMax := 0;
  Readln(N);
  for i:=1 to N do begin
    Readln(x,y);
    if y = 0 then begin
      if not Found or (x < xmin) then
        xMin:= x;
      if not Found or (x > xmax) then
        xMax:= x;
      Found:= True;
    end
    else
    if abs(y) > ymax then
      ymax:= abs(y);
  end;
  writeln((xMax - xMin) * yMax / 2);
end.
```

Ещё один вариант решения предложил **Д.Ф. Муфаззалов** (Уфа, УГАТУ). Его идея состоит в том, что сначала ищется первая точка, лежащая на оси OX, а потом обрабатываются все остальные. Это позволяет избавиться от логической переменной:

```
var i,x,y,n,h,minx,maxx:integer; s:real;
begin
  readln(n);
  s:=0; h:=0;
  repeat
    readln(x,y);
    if abs(y)>h then h:=abs(y);
    dec(n)
  until  ((y=0) or (n=0));
  if y=0 then begin
    minx:=x;
    maxx:=x;
    for i:=n downto 1 do begin
      readln(x,y);
      if y = 0 then begin
        if x < minx then minx:=x
        else
         if x > maxx then maxx:=x;
      end
      else
        if (abs(y)>h)then  h:=abs(y);
    end;
    s:=(maxx-minx)*h/2;
  end;
  writeln(s);
end.
```

Д.Ф. Муфаззалов предложил также аналогичное решение на языке C++:

```
#include <math.h>
#include <iostream.h>
void main()
{
int n, minx, maxx, y,x,h=0;
cin>>n; float s=0;
do {
  cin>>x>>y;
  if (y!=0)
   if (abs(y)>h) h=abs(y);
  n--;
  }
while (y!=0&&n>0);
if (y==0) {
  minx=x;
  maxx=x;
  for (;n>0;n--) {
   cin>>x>>y;
   if (y==0)
     {
     if (x<minx) minx=x;
     else
       if (x>maxx) maxx=x;
     }
   else
```

```
        if (abs(y)>h) h=abs(y);
      }
      s=(maxx-minx)*h/2.0;
     }
   cout<<s;
   }
```

47) Нам нужно определить площадь треугольника, одно из оснований которого лежит на оси OX. Пусть длина этого основания равна B, а высота треугольника равна H. Тогда площадь треугольника равна S = B·H/2. Важно, что у треугольника нет обеих точек с осью OY, то есть, x-координаты (абсциссы) всех вершин должны быть одного знака, или все положительные, или все отрицательные. Поэтому нужно рассматривать отдельно треугольники, расположенные слева от оси OY (с отрицательными абсциссами), и треугольники, расположенные справа от оси OY (с положительными абсциссами).

Таким образом, эта задача представляет собой как бы «двойную» задачу 46. Так, для левой полуплоскости, нужно найти:

4) **xMin1** – минимальное значение x-координаты среди всех точек, для которых y-координата равна нулю, а x-координата отрицательная;

5) **xMax1** – максимальное значение x-координаты среди всех точек, для которых y-координата равна нулю, а x-координата отрицательная;

6) **yMax1** – максимальный модуль y-координаты среди всех точек, а x-координата отрицательная.

Тогда максимальная площадь **S1** = (**xMax1** – **xMin1**) · **yMax1** / 2.

Единственная сложность состоит в том, чтобы записать в переменные **xMax1** и **xMin1** некоторые начальные значения, которые позволят определить, что еще ни одной точки на отрицательно части оси OX не найдено. Можно ввести еще одну логическую переменную (назовём её **Found1**), которая сначала равна **False** (ни одна точка на отрицательной части оси OX не найдена), и получает значение **True**, когда такая точка найдена.

Аналогичный поиск нужно провести для правой пролуплоскости, и затем выбрать максимальную площадь из двух.

```
program Treug;
var i, N, x, y: integer;
    xMin1, xMax1, yMax1: integer;
    xMin2, xMax2, yMax2: integer;
    Found1, Found2: boolean;
    S1, S2: real;
begin
  Found1:= False;
  Found2:= False;
  xMin1 := 0; xMax1 := 0; yMax1 := 0;
  xMin2 := 0; xMax2 := 0; yMax2 := 0;
  Readln(N);
  for i:=1 to N do begin
    Readln(x,y);
    if x < 0 then begin
      if y = 0 then begin
        if not Found1 or (x < xMin1) then xMin1:= x;
        if not Found1 or (x > xMax1) then xMax1:= x;
```

```
          Found1:= True;
        end
        else
        if abs(y) > yMax1 then yMax1:= abs(y);
      end;
      if x > 0 then begin
        if y = 0 then begin
          if not Found2 or (x < xMin2) then xMin2:= x;
          if not Found2 or (x > xMax2) then xMax2:= x;
          Found2:= True;
        end
        else
        if abs(y) > yMax2 then yMax2:= abs(y);
      end;
    end;
    S1 := (xMax1 - xMin1) * yMax1 / 2;
    S2 := (xMax2 - xMin2) * yMax2 / 2;
    if S1 > S2 then
         writeln(S1)
    else writeln(S2);
  end.
```

48) Обратите внимание, что в задаче не указано ограничение на количество участников, это говорит о том, что не нужно хранить в массиве все результаты. Действительно, нас интересуют только три лучших, поэтому логично определить массив из трёх структур:

```
Type TInfo = record
       name: string;
       res: integer;
     end;
var Info: array[1..3] of TInfo;
```

Конечно, здесь можно использовать и отдельные переменные для хранения трёх имён и трех лучших результатов, но работать со структурами удобнее.

В самом начале все результаты нужно обнулить:

```
for i:=1 to 3 do Info[i].res := 0;
```

Затем вводим число записей в таблице:

```
Read(N);
```

Далее организум цикл (**`for i:=1 to N do...`**), на каждом шаге читаем очередной результат и имя в переменные

```
var res0: integer;
    name0: string;
```

Затем сравниваем с лучшими известными результатами и корректируем таблицу лучших – сдвигаем все результаты, чтобы освободить место для нового призёра:

```
if res0 > Info[1].res then begin
      Info[3]:= Info[2]; Info[2]:= Info[1];
      Info[1].name := name0;
      Info[1].res := res0;
end
else if res0 > Info[2].res then begin
      Info[3]:= Info[2];
```

```
      Info[2].name := name0;
      Info[2].res := res0;
end
else if res0 > Info[3].res then begin
      Info[3].name := name0;
      Info[3].res := res0;
end;
```

Этот алгоритм хорошо работает, когда каждый участник присутствует в таблице только один раз. В нашем же случае в таблице могут быть несколько результатов для каждого участника, поэтому перед приведённым выше блоком нужно проверить, не улучшил ли кто-то из уже записанныхв таблицу свой результат.

```
if name0 = Info[1].name then begin
  if res0 > Info[1].res then Info[1].res := res0;
end
else if name0 = Info[2].name then begin
  if res0 > Info[1].res then begin
    Info[2]:= Info[1];
    Info[1].name := name0;
    Info[1].res := res0;
  end
  else if res0 > Info[2].res then Info[2].res := res0;
end
```

Обратите внимание на то, что проверять совпадение имени для участника, который находится на третьем месте (**`if name0 = Info[3].name then...`**) не обязательно, поскольку в случае, если он улучшил свой результат, прошлый результат будет автоматически вытеснен из таблицы призёров общим алгоритмом.

Остается привести полную программу:

```
program Tetris;
type TInfo = record
       name: string;
       res: integer;
     end;
var Info: array[1..3] of TInfo;
    i, N: integer;
    res0: integer;
    name0: string;
begin
  for i:=1 to 3 do Info[i].res := 0;
  Read(N);
  for i:=1 to N do begin
    Readln(res0, name0);
    if name0 = Info[1].name then begin
      if res0 > Info[1].res then Info[1].res := res0;
    end
    else if name0 = Info[2].name then begin
      if res0 > Info[1].res then begin
        Info[2]:= Info[1];
        Info[1].name := name0;
        Info[1].res := res0;
      end
```

```
        else if res0 > Info[2].res then Info[2].res := res0;
      end
      else if res0 > Info[1].res then begin
        Info[3]:= Info[2]; Info[2]:= Info[1];
        Info[1].name := name0;
        Info[1].res := res0;
      end
      else if res0 > Info[2].res then begin
        Info[3]:= Info[2];
        Info[2].name := name0;
        Info[2].res := res0;
      end
      else if res0 > Info[3].res then begin
        Info[3].name := name0;
        Info[3].res := res0;
      end;
    end;
    for i:=1 to 3 do
      writeln(i, ' место. ', Info[i].name,
                 ' (', Info[i].res, ')')
  end.
```

Недостаток этого решения стоит в том, что оно содержит много условных операторов, в которых легко запутаться. **Д.Ф. Муфаззалов** (г. Уфа) предложил другое решение, основанное на идее вытеснения «лишнего» из массива призёров в том случае, если новый прочтанный результат лучше какого-то из предыдущих. Вместо структур будем использовать два массива, для хранения результатов и имен:

```
var name:array[1..3] of string;
    res: array[1..3] of integer;
```

Получив новые данные в переменные **res0** и **name0**, сначала ищем в списке призеров запись с тем же именем:

```
posName:= 1;
while (posName < 3) and
      (name[posName] <> name0) do inc(posName);
```

Этот цикл останавливается, если нашли призёра с тем же именем, а если такого нет, **posName** будет равно 3 (может быть вытеснен последний призёр).

Теперь выше в списке ищем призёра, у которого результат меньше, чем **res0**:

```
k:= 1;
while (k < posName) and
      (res[k] >= res0) do inc(k);
```

Если все результаты больше, чем **res0**, цикл остановится при **k=posName**.

Есл призёр с номером **k** имеет результат меньше, чем res0, нужно вставть новый результат на место **k**.

```
if res[k] < res0 then begin
  for j:=posName downto k+1 do begin
    res[j]:= res[j-1];
    name[j]:= name[j-1];
  end;
  res[k]:= res0;
```

```
    name[k]:= name0;
  end;
```

Обратите внимание, что сдвиг таблицы идет не до самого низа, а до позиции **posName**: в том случае, когда данный участник уже был в списке призёров, его результат в позиции **posName** будет удалён. Если этого участника ранее не было в призёрах, вытесняется последний элемент таблицы (именно на него указывает в этом случае переменная **posName**). Приведем полную программу:

```
var i, j, k, posName, N:longint;
    res0: integer;
    name0: string;
    name:array[1..3] of string;
    res: array[1..3] of integer;
begin
readln(N);
for i:=1 to 3 do res[i]:=0;
for i:=1 to N do begin
  readln(res0, name0);
    { ищем участнка в списке }
  posName:= 1;
  while (posName < 3) and (name[posName] <> name0) do
    inc(posName);
    { ищем результат меньше, чем новый }
  k:= 1;
  while (k < posName) and (res[k] >= res0) do
    inc(k);
    { если получен лучший результат,... }
  if res[k] < res0 then begin
    for j:=posName downto k+1 do begin
      res[j]:= res[j-1];
      name[j]:= name[j-1];
    end;
    res[k]:= res0;
    name[k]:= name0;
  end;
end;
for i:=1 to 3 do
  writeln(i,' место. ', name[i], ' (', res[i], ')');
end.
```

49) В этой задаче нам нужно для каждой четверти координатной плоскости определить:

1) количество точек
2) точку, ближайшую к какой-нибудь оси координат
3) минимальное расстояние от этой точки до ближайшей оси

Для хранения этих данных выделим массивы из 4-х элементов:

```
var count, R, xR, yR: array[1..4] of integer;
```

В начале работы обнуляем счётчики:

```
for i:=1 to 4 do count[i]:=0;
```

Получив координаты точки **(x,y)**, нужно определить, к какой четверти она принадлежит. Вспомним, что точки, лежащие на осях, то есть такие, для которых **x*y=0**, считаются не принадлежащими какой-либо четверти, поэтому их обрабатывать не нужно:

```
if x*y <> 0 then begin
  if (x > 0) and (y > 0) then k:= 1 else
  if (x < 0) and (y > 0) then k:= 2 else
  if (x < 0) and (y < 0) then k:= 3 else k:= 4;
  { здесь работаем с четвертью k }
end;
```

Обработка сводится к тому, что мы

1) увеличиваем счётчик точек
2) проверяем, не является ли эта точка первой (при этом **count[k]=1**) или ближайшей к осям; если да, запоминаем её координаты.

```
count[k]:= count[k] + 1;
if (count[k] = 1) or
   (abs(x) < R[k]) or (abs(y) < R[k]) then begin
  if abs(x) < abs(y) then { выбрать min(abs(x),abs(y)) }
      R[k]:= abs(x)
  else R[k]:= abs(y);
  xR[k]:= x; yR[k]:= y;
end;
```

Обратите внимание, что строгие неравенства в условии

```
(abs(x) < R[k]) or (abs(y) < R[k])
```

обеспечивают выбор *первой* из точек, если несколько точек находятся в какой-то четверти на одинаковом расстоянии от осей (так и задано в условии).

После окончания цикла нужно выбрать четверть с наибольшим количеством точек, а если таких несколько, то из них выбрать четверть с минимальной величиной **R**. Это можно сделать следующим образом. Начнём с первой четверти (**k:=1**) и будем запоминать новый номер четверти **i**, если выполняется одно из условий:

1) **count[i]>count[k]**
2) **(count[i] = count[k]) and (R[i] < R[k])**

Получается такой цикл выбора четверти:

```
k:=1;
for i:=2 to 4 do
  if (count[i]>count[k]) or
     (count[i] = count[k]) and (R[i] < R[k]) then
    k:= i;
```

Приведем полную программу:

```
var count, R, xR, yR: array[1..4] of integer;
    i, k, N, x, y: integer;
begin
  for i:=1 to 4 do count[i] := 0;
  readln(N);
  for i:=1 to N do begin
    readln(x, y);
    if x*y <> 0 then begin
```

```
        if (x > 0) and (y > 0) then k:= 1 else
        if (x < 0) and (y > 0) then k:= 2 else
        if (x < 0) and (y < 0) then k:= 3 else k:= 4;
        count[k]:= count[k] + 1;
        if (count[k] = 1) or
           (abs(x) < R[k]) or (abs(y) < R[k]) then begin
          if abs(x) < abs(y) then {выбрать min(abs(x),abs(y))}
              R[k]:= abs(x)
          else R[k]:= abs(y);
          xR[k]:= x; yR[k]:= y;
        end;
      end;
    end;
    k:=1;
    for i:=2 to 4 do
      if (count[i]>count[k]) or
         (count[i] = count[k]) and (R[i] < R[k]) then
        k:= i;
    writeln('K = ', k);
    writeln('M = ', count[k]);
    writeln('A = (', xR[k], ',', yR[k], ')');
    writeln('R = ', R[k]);
  end.
```

50) Итак, нужно найти номера всех отсчетов, которые войдут в максимальное произведение. Все числа неотрицательные, поэтому очередное число
   1) увеличивает произведение, если оно больше 1
   2) не изменяет его, если оно равно 1 (по условию, в этом случае значение можно брать, а можно и не брать, так как «*Если таких подмножеств несколько, то выбрать можно любое из них*»)
   3) уменьшает его, если оно меньше 1.

Вывод: нужно вывести номера всех входных значений, которые больше или равны 1 (или строго больше 1).

Сложность этой задачи в том, что этих значений может быть очень много, хранить их в массиве нельзя (программа будет неэффективна по использованию памяти). Если бы данные вводились из файла, можно было бы использовать такой алгоритм:

```
var F: Text;
...
for i:=1 to N do begin
  read(F, x);           { чтение очередного числа }
  if x >= 1 then write(i, ' ');
end;
```

Не забудем, что возможен и ещё один вариант: среди чисел нет ни одного, большего или равного 1, на этот случай нужно искать максимальное число и вывести его номер после цикла тогда, когда оно меньше 1:

```
var max: double;
    iMax: integer;
...
max:= 0; iMax:= 0;
for i:=1 to N do begin
  read(F, x);           { чтение очередного числа }
```

```
    if x >= 1 then write(i, ' ');
    if x > max then begin
      max := x; iMax := i;
    end;
  end;
  if max < 1 then writeln(iMax);
```

Но по условию неявно (или явно?) предполагается, что данные читаются с клавиатуры, поэтому выводить результат нужно тогда, когда ввод уже закончен и все **N** (большое количество!) чисел обработаны. Как запомнить нужные номера так, чтобы фактически их не хранить?

Чтобы решить эту проблему, нужно внимательно прочитать условие и обратить внимание на две «мелочи»:

1) все числа вводятся с точностью до 0.1
2) все числа различные.

Это значит, что чисел, которые меньше 1 и нас не интересуют (когда есть большие или равные единице), может быть не более 10:

0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 и 0,9

Для того, чтобы запомнить их номера, достаточно массива из 10 элементов:

```
  var small: array[0..9] of integer;
```

В элементе **small[i]** будем хранить номер измерения, равного **i/10**, если оно было во входной последовательности, или 0, если такого числа не было. Получив число **x** и выяснив, что оно меньше 1, вычислим номер соответствующего эемента массива **small** как **round(x*10)**.

```
  if x < 1 then
    small[round(x*10)]:= i;
```

Поскольку **x** вводится с точностью до десятых, величина **x*10** – целое число, формально мы приводим его к целому типу с помощь функции округления **round**.

Все остальные номера, не записанные в массив **small**, нужно (после окончания чтения данных) выводить на экран:

```
  for i:=1 to N do
    if { номер i есть в массиве small } then
      write(i, ' ');
```

Для того, чтобы найти этот номер (или убедиться, что его нет), нужен проход по массиву **small**. Логическая переменная **isSmall** примет истинное значение, если номер найден:

```
  var isSmall: boolean;
  ...
  isSmall:= False;
  for j:=0 to 9 do
    if small[j] = i then begin
      isSmall:= True;
      break;
    end;
```

Теперь можно собрать всю программу:

```
var small: array[0..9] of integer;
    N, i, j, iMax: integer;
    isSmall: boolean;
    x, max: real;
begin
  readln(N);
  for i:=0 to 9 do small[i]:= 0;
    { ввод данных, поск максимума }
  max := 0;
  for i:=1 to N do begin
    readln(x);
    if x < 1 then
      small[round(x*10)]:= i;
    if x > max then begin
      max := x; iMax := i;
    end;
  end;
    { вывод результата в общем случае }
  for i:=1 to N do begin
    isSmall:= False;
    for j:=0 to 9 do
      if small[j] = i then begin
        isSmall:= True;
        break;
      end;
    if not isSmall then write(i, ' ');
  end;
    { особый случай: все <= 1 }
  if max < 1 then writeln(iMax);
end.
```

Для ускорения работы программы в случае, когда все нужно вывести все номера (нет ни одного, меньшего или равного единице), можно считать номера, которые записываются в массив **small**, и, если этот счётчик равен нулю, выводить все номера подряд, от 1 до N. Поскольку переменная **iMax** после окончания цикла ввода используется только тогда, когда все числа маленькие, искать это значение нужно только среди маленьких чисел, то есть внутри условного оператора **if(x<1)**:

```
var small: array[0..9] of integer;
    N, i, j, iMax, countSmall: integer;
    isSmall: boolean;
    x, max: real;
begin
  readln(N);
  for i:=0 to 9 do small[i]:= 0;
  countSmall := 0;
  max := 0;
  for i:=1 to N do begin
    readln(x);
    if x < 1 then begin
      small[round(x*10)]:= i;
      Inc(countSmall);
      if x > max then begin
        max := x; iMax := i;
```

```
        end;
      end;
    end;
      { если все < 1 }
    if countSmall = N then writeln(iMax)
      { если все >= 1 }
    else if countSmall = 0 then
      for i:=1 to N do write(i, ' ')
    else
      { вывод результата в общем случае }
      for i:=1 to N do begin
        isSmall:= False;
        for j:=0 to 9 do
          if small[j] = i then begin
            isSmall:= True;
            break;
          end;
        if not isSmall then write(i, ' ');
      end;
  end.
```

Ещё один вариант решения «проблемы маленьких чисел» продложил **С.М. Семёнов** (Владивостокский государственный университет экономики и сервиса). Идея состоит в том, чтобы заполнять массив **small** номерами маленьких элементов по мере их поступления *с начала массив*а. Как и в предыдущей программе, **countSmall** – количество уже найденных маленьких чисел:

```
  var small: array[1..10] of integer;
      N, i, j, iMax, k, countSmall: integer;
      isSmall: boolean;
      x, max: real;
  begin
    readln(N);
    countSmall := 0;
    max := 0;
    for i:=1 to N do begin
      readln(x);
      if x < 1 then begin
        Inc(countSmall);
        small[countSmall]:= i;
        if x > max then begin
          max := x; iMax := i;
        end;
      end;
    end;
      { если все < 1 }
    if countSmall = N then writeln(iMax)
      { если все >= 1 }
    else if countSmall = 0 then
      for i:=1 to N do write(i, ' ')
    else begin
      { вывод результата в общем случае }
      k:= 1; { начинать просмотр маленьких чисел с номера k }
      for i:=1 to N do begin
        isSmall:= False;
```

```
        for j:=k to countSmall do begin
          if small[j] > i then begin
            k:= j;
            break
          end;
          if small[j] = i then begin
            isSmall:= True;
            k:= j+1;
            break
          end
        end;
        if not isSmall then write(i, ' ');
      end
    end
  end.
```

Наиболее элегантное решение было предложено пользователем *Lavanda* (http://egekp.unoforum.ru/?1-16-0-00000069-000-0-0-1396421949). Идея состоит в том, чтобы хранить в отдельном массиве номера маленьких чисел, которые не нужно выводить на экран. Маленьких чисел может бытьне более 9, выделяем массив из 10 элементов, чтобы последний всегда был равен нулю – он будет «барьером» при выводе.

```
var a:array[1..10] of integer;
...
for i:=1 to 10 do a[i]:=0;
```

Прочитав очередное число, обрабатываем его, если оно меньше 1: ищем максимум (на тот случай, когда все числа < 1) и запоминаем его номер в массиве **a**:

```
readln(x);
if x < 1 then begin
  if x > max then begin { новый максимум }
    max:= x;
    nmax:= i
  end;
  m:= m + 1; { новое маленькое число }
  a[m]:= i   { записали номер в массив }
end;
```

При выводе начинаем с 1 и пропускаем те номера, которые есть в массиве **a**:

```
j:=1;
for i:=1 to n do
  if (a[j] = 0) or (i < a[j]) then
     write(i, ' ')
  else
     j:=j+1 { пропуск номера }
```

Здесь **j** – это номер следующего элемента массива **a**. Поскольку последний элемент массива (10-й) всегда будет нулевым, он будет «барьером» и выхода за границу массива не произойдет.

```
var n,i,m,j,nmax:integer;
    x,max:real;
    a:array[1..10] of integer;
begin
  readln(n);
  for i:=1 to 10 do a[i]:=0;
  max:=0; nmax:=0;
```

```
    m:=0;
    for i:=1 to n do begin
      readln(x);
      if x < 1 then begin
        if x > max then begin { новый максимум }
          max:=x; nmax:=i
        end;
        m:=m+1; a[m]:=i { записали номер в массив }
      end;
    end;
    if n = m then writeln(nmax)
    else begin
      j:=1;
      for i:=1 to n do
        if (a[j] = 0) or (i < a[j]) then
           write(i, ' ')
        else
           j:=j+1 { пропуск номера }
    end;
    writeln;
  end.
```

В приведённых выше решениях предполагается, что нужно сначала ввести все данные, и только потом выводить результат. Для случая, когда разрешается чередовать ввод и вывод, лучшее решение предложил **Андрей Проскурнёв**. Его идея состоит в том, что мы во время ввода сразу выводим номера всех чисел, которые больше 1 (они гарантированно войдут в нужное произведение), а в конце, если все числа оказались меньше 1, выводим наибольшее из них.

```
  var
    all_below_one :boolean;
    i, n, max_x_idx :integer;
    x, max_x :real;
  begin
    all_below_one := true;
    max_x := 0.0;
    max_x_idx := 1;
    readln(n);
    for i := 1 to n do begin
      readln(x);
      if x > 1 then begin
        all_below_one := false;
        write(i, ' ');
      end
      else
        if all_below_one
           and (x > max_x) then begin
          max_x := x;
          max_x_idx := i;
        end;
    end;
    if all_below_one then begin
      write(max_x_idx, ' ');
    end;
```

```
end.
```

51) В задаче нужно найти максимальное произведение двух положительных чисел, которое делится на 7, но не делится на 49. Легко сообразить, что в произведении одно число делится на 7, но не делится на 49, а второе – не делится на 7. Таким образом, можно выделить две группы интересующих нас чисел:

- числа, которые делятся на 7, но не на 49
- числа, которые не делятся на 7

В каждой группе нужно выбрать наибольшее, их произведение должно быть равно контрольному значению.

Для ввода с подсчётом введённых чисел будем использовать такой цикл

```
count:= 0;
while True do begin
  read(x);
  if x = 0 then break;
  Inc(count); { увеличить счётчик }
  ...
end;
```

Приведем полную программу

```
var x, max7, max, R, R1, count: integer;
begin
  max:= 0; max7:= 0;
  count:= 0;
  while True do begin
    read(x);
    if x = 0 then break;
    Inc(count);
    if (x mod 7 = 0) and (x mod 49 <> 0) and
       (x > max7) then max7:= x;
    if (x mod 7 <> 0) and (x > max) then max:= x;
  end;
  read(R);
  R1:= max7*max;
  if R1 = 0 then R1:= 1;
  writeln('Введено чисел: ', count);
  writeln('Контрольное значение: ', R );
  writeln('Вычисленное значение: ', R1);
  if R1 = R then
       writeln('Значения совпали')
  else writeln('Значения не совпали')
end.
```

52) В задаче нужно для каждой пары чисел (D,K) найти остаток от деления K на D (не наоборот!) и подсчитать, какой остаток встречался чаще всего. Учитывая, что число детей на каждом утреннике не больше 100, остаток может быть в диапазоне от 0 до 99. Для того, чтобы считать, сколько раз встретилось каждое число из этого диапазона, заведём массив счётчиков:

```
const MAX=100;
var count: array[0..MAX-1] of integer;
```

Сначала этот массив нужно обнулить:

```
for i:=0 to MAX-1 do count[i]:= 0;
```

Затем, прочитав очередную пару чисел, увеличиваем нужный счётчик:

```
ist := K mod D;
count[ost]:= count[ost] + 1;
```

После окончания цикла нужно найти номер максимального элемента массива count, причем не забыть, что нас интересуют только ненулевые остатки, то есть нужно начать цикл перебор не с 0, а с 1 (элемент **count[0]** нас не интересует):

```
nMax:=1;
for i:=2 to MAX-1 do
  if count[i] >= count[nMax] then
  nMax:=i;
```

Не забудем, что по условию «если несколько чисел записывались одинаково часто, надо вывести большее из них». Поэтому в условном операторе нужно именно нестрогое неравенство (**<=**, а не **<**)!

Наконец, вспомним, что если все остатки нулевые, то нужно вывести 0:

```
if count[nMax] = 0 then
  nMax := 0;
writeln(nMax);
```

Приведем полную программу:

```
const MAX=100;
var count: array[0..MAX-1] of integer;
    i, ost, nMax, N, D, K: integer;
begin
readln(N);
for i:=1 to N do begin
  readln(D, K);
  ost := K mod D;
  count[ost] := count[ost] + 1;
end;
nMax:=1;
for i:=2 to MAX-1 do
  if count[i] >= count[nMax] then
  nMax:=i;
if count[nMax] = 0 then
  nMax := 0;
writeln(nMax);
end.
```

Эта программа эффективна по памяти, потому что размер массива **count** не зависит от N. Она также эффеативна по скорости, потому что от N зависит только время ввода данных, оно изменяется пропорционально N. Это значит, что «при увеличении размера входных данных N в t раз (t – любое число) время её работы увеличивается не более чем в t раз».

Можно было сэкономить на одном элементе массива **count**, потому что элемент **count[0]** мы не используем. Тогда в цикле ввода нужно было бы проверять на неравенство остатка нулю:

```
var count: array[1..MAX-1] of integer;
...
for i:=1 to N do begin
  readln(D, K);
  ost := K mod D;
  if ost > 0 then
    count[ost] := count[ost] + 1;
end;
```

53) Для решения этой задачи можно, например, определить, когда нужно перехать с дороги A на дорогу B так, чтобы общее время было минимальным. Пусть мы переехали по переезду A_iB_i. Тогда время общее время движения равно

$$T_i = SA_i + t + SB_i$$

Здесь SA_i – это время проезда от A_0 до A_i;

t – время переезда между A_i до B_i;

SB_i – это время проезда от B_i до B_N.

Из всех возможных значений T_i нужно выбрать минимальное, это и есть ответ.

Напишем сначала именно такое (неэффективное) решение. Для того, чтобы построить массив **SA**, будем накапливать частичные суммы:

$SA_i = 0$

$SA_1 = A_1 = SA_0 + A_1$

$SA_2 = A_1 + A_2 = SA_1 + A_2$

...

Все это укладывается в формулу $SA_i = SA_{i-1} + A_i$, которую будем использовать в цикле для заполнения массива **SA**:

```
SA[0] := 0;
for i:=1 to N do
  SA[i] := SA[i-1] + A[i];
```

Далее аналогично заполняем массив **SB**,только с конца:

$SB_N = 0$

$SB_{N-1} = B_N = B_N + SB_N$

$SB_{N-2} = B_{N-1} + B_N = B_{N-1} + SB_{N-1}$

...

Все это укладывается в формулу $SB_{i-1} = B_i + SB_i$, которую будем использовать в цикле для заполнения массива **SB**:

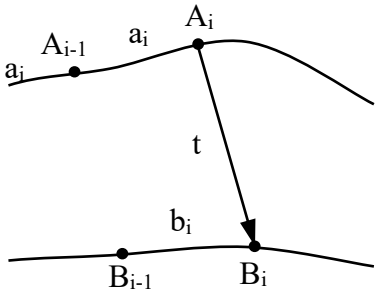
```
SB[N] := 0;
for i:=N downto 1 do
  SB[i-1] := B[i] + SB[i];
```

Теперь остается найти минимальное время:

```
Tmin := SB[0];
for i:=1 to N do begin
  Ti := SA[i] + SB[i];
  if Ti < Tmin then Tmin := Ti
end;
writeln(Tmin + t);
```

Поскольку время переезда t всегда присутствует, мы добавляем его прямо при выводе ответа.

Как мы уже говорили, это неоптимальное решение, поскольку мы использовали массивы и предполагали, что все данные уже загружены в память и доступны. За него, скорее всего, дадут 3 балла, то есть, 1 балл (только!) будет снят за неоптимальность.



Прелесть этой задачи состоит в том, что она имеет простое оптимальное решение, значительно более короткое, чем предыдущее.

Предположим, что мы знаем оптимальное время TA_{i-1} и время TB_{i-1}, за которое можно проехать из A_0 в A_{i-1} и B_{i-1} соответственно. Поскольку мы не можем возвращаться с дороги B на дорогу A, имеем

$$TA_{i-1} = a_1 + a_2 + \ldots + a_{i-1}$$

Сразу отметим, что эту величину можно накапливать постепенно, не используя массив.

Как же получить теперь TB_i? У нас есть два варианта проезда в B_i:

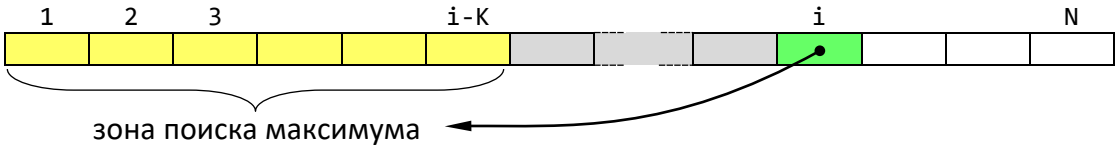
1) через A_{i-1} и A_i с общей длиной маршрута $TA_{i-1} + a_i + t$
2) через B_{i-1} с общей длиной маршрута $TB_{i-1} + b_i$

Из этих двух значений нужно выбрать минимальное. Повторяем такой выбор на каждом шаге, в результате TB_N – это ответ к задаче.

Так как все предыдущие значения TA_k и TB_k, кроме ближайших, нам не нужны, можно не заводить массивы, а обойтись простыми переменными:

```
var  N, i, t, TA, TB, ai, bi: integer;
begin
  Readln( N );
  Readln( t );
  TA := 0;
  TB := t;
  for i:=1 to N do begin
    Readln( ai, bi );
    TA := TA + ai;
    if TB+bi < TA+t then
         TB:= TB + bi
    else TB:= TA + t;
  end;
  writeln(TB);
end.
```

54) Сначала поймем, что требуется найти. Предположим, что все результаты измерения записаны в массив:



Нужно найти пару элементов с максимальной суммой, отстоящих друг от друга не менее, чем на K (в данной задаче K = 7) позиций в массиве, то есть разность их индексов должна быть больше или равна K. Предположим, что второй элемент в паре (стоящий ближе к концу массива) имеет индекс **i**. Тогда соответствующий первый элемент из той же пары нужно искать в диапазоне индексов от 1 до **i-K** (см. желтую зону на рисунке). Таким образом, мы можем найти максимальную сумму из всех пар, в которых второй элемент имеет индекс **i**:

```
max := Buf[1];
for j:=2 to i-K do
  if Buf[j] > max then max := Buf[j];
writeln(max + Buf[i]);
```

Здесь и далее предполагается, что массив, в котором хранятся результаты измерений, имеет имя **Buf**. Из всех таких пар (при всех **i** от **K+1** до **N**) нужно выбрать пару с максимальной суммой (будем хранить её в переменной **maxSum**):

```
for i:=K+1 to N do begin
  max := Buf[1];
  for j:=2 to i-K do
    if Buf[j] > max then max := Buf[j];
  if i = K+1 then
    maxSum := max + Buf[i]
  else
    if max + Buf[i] > maxSum then
       maxSum := max + Buf[i];
```

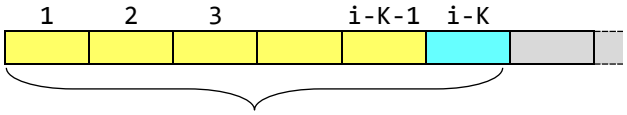
```
  end;
  writeln(maxSum);
```

Обратите внимание, что мы не могли сразу присвоить начальное значение переменной maxSum, потому что диапазон значений входных данных неизвестен. Поэтому это начальное значение записывается на первом шаге цикла:

```
  if i = K+1 then
    maxSum := max + Buf[i]
  else ...
```

Это решение правильное, но неэффективное. Во-первых, все данные хранятся в массиве, то есть, нужно выделить массив достаточного размера, а по условию N может быть очень велико (решение неэффективно по использованию памяти). Во-вторых, для каждого **i** мы заново просматриваем всю начальную часть массива (ищем максимум), это значит, что программа неэффективна по количеству операций (времени выполнения).

Сначала избавимся от лишних операций при поиске максимума в начальной части массива. Предположим, что мы уже знаем максимальный элемент (обозначим его **max**) в диапазоне индексов от 1 до **i-K-1**. При переходе к следующему значению **i** нам нужно найти максимум из элементов с индексами от 1 до **i-K**, то есть к уже рассмотренным элементам (для которых мы знаем максимум **max**) добавляется один элемент **Buf[i-K]**, выделенный голубым цветом на рисунке:



Поэтому новый максимум – это наибольшее значение из **max** и **Buf[i-K]**:

```
  if Buf[i-K] > max then
    max := Buf[i-K];
```

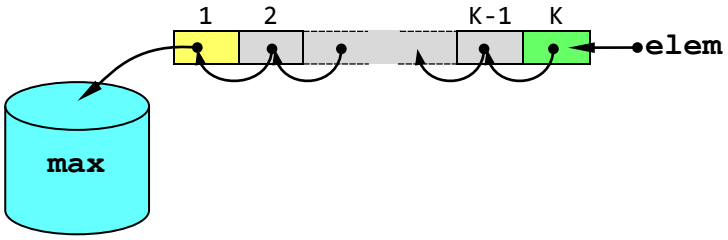
Получается такое решение, которое уже не использует внутренний цикл (по переменной **j**) и может считаться эффективным по времени выполнения:

```
  for i:=K+1 to N do begin
    if i = K+1 then begin
      max := Buf[1];
      maxSum := max + Buf[i];
    end
    else begin
      if Buf[i-K] > max then
        max := Buf[i-K];
      if max + Buf[i] > maxSum then
        maxSum := max + Buf[i];
    end
  end;
```

Следующая задача – избавиться от массива, в котором хранятся все введённые значения. Заметим, что в предыдущем варианте мы фактически не использовали всю начальную часть массива для поиска максимума, а добавляли к рассмотренным ранее элементам очередной элемент с индексом **i-K**.

Представим себе, что мы на каждом шаге читаем из входного потока один элемент, который в предыдущих вариантах решения хранился в элементе массива с индексом **i**. Тогда получается, что на этом шаге нужно добавить к элементам, среди которых ищется максимум, элемент, прочитанный на K шагов раньше, то есть, нужно хранить (в массиве) только K предыдущих элементов. Размер этого массива K не зависит от N, которое может быть очень большим. После использования значения с номером **i-K** для поиска максимума нужно запомнить в массиве только что прочитанное значение.

Структура данных, с помощью которой можно решить эту задачу, называется *очередью*. Элементы в очередь добавляются с одного конца (это будет конец массива), а извлекаются – с другого (с начала массива). После удаления первого элемента (который «включается» в поиск максимума) все элементы очереди, начиная со второго, сдвигаются на одну позицию к началу массива, а в последний (освободившийся) элемент записывается только что прочитанное значение (оно хранится в переменной **elem**):



Сначала нужно заполнить очередь: прочитать в неё первые K измеренных значений:

```
var Buf: array[1..K] of integer;
...
for i:=1 to K do
  read(Buf[i]);
```

Затем читаем и обрабатываем оставшиеся N-K значений:

```
for i:=K+1 to N do begin
  read(elem);
...
end;
```

На первом шаге этого цикла записываем в переменную **max** первый элемент очереди, а в переменную maxSum – сумму **max** и только что прочитанного значения **elem**; на остальных шагах цикла обновляем эти максимальные значения:

```
if i = K+1 then begin
  max := Buf[1];
  maxSum := Buf[1] + elem;
end
else begin
  if Buf[1] > max then
    max := Buf[1];
  if max+elem > maxSum then
    maxSum := max + elem;
end
```

После этоно нужно сдвинуть все элементы очереди на одну позицию к началу (удалить уже использованный первый элемент) и добавить в конец очереди новое хзначение **elem**:

```
for j:=1 to K-1 do
  Buf[j] := Buf[j+1];
Buf[K] := elem;
```

В конце цикла нужное значение оказывается в переменной **maxSum**. Приведём полное правильное решение, эффективное как по времени выполнения (оно линейно зависит от N), так и по используемой памяти (не зависит от N):

```
const K = 7;
var i, j, N, max, maxSum, elem: integer;
    Buf: array[1..K] of integer;
begin
  read(N);
    { заполняем очередь }
  for i:=1 to K do read(Buf[i]);
```

```
    { обрабатываем оставшиеся данные }
  for i:=K+1 to N do begin
    read(elem);
      { начальные значения для max и maxSum }
    if i = K+1 then begin
      max := Buf[1];
      maxSum := Buf[1] + elem;
    end
      { обновление max и maxSum }
    else begin
      if Buf[1] > max then max := Buf[1];
      if max+elem > maxSum then
         maxSum := max + elem;
    end;
      { обработка очереди }
    for j:=1 to K-1 do      { сдвиг очереди }
      Buf[j] := Buf[j+1];
    Buf[K] := elem;         { добавить элемент в очередь }
  end;
  writeln(maxSum);
end.
```

Это решение можно ещё немного улучшить, если избавиться от сдвига элементов в очереди на каждом шаге цикла. Для этого будем считать, что очередь замкнута в кольцо, и её голова (место первого элемента) смещается при чтении нового значения. Удобнее всего нумеровать элементы массива с нуля и при чтении **i**-ого по счёту значения считать головой очереди элемент массива с номером **i mod K**. Этот элемент на следующем шаге становится концом очереди, поэтому именно в него нужно записать очередное прочитанное значение. Тогда преобразование очереди на каждом шаге выглядит так:

```
next := Buf[i mod K];
Buf[i mod K] := elem;
```

Здесь **next** – это переменная для хранения первого элемента очереди, который будет участвовать в поиске максимума, а **elem** – значение, только что прочитанное из входного потока. Программа приобретает такой вид:

```
const K = 7;
var i, N, max, maxSum, next, elem: integer;
    Buf: array[0..K-1] of integer;
begin
  read(N);
    { заполняем очередь }
  for i:=0 to K-1 do read(Buf[i]);
    { обрабатываем оставшиеся данные }
  for i:=K to N-1 do begin
    read(elem);
      { обработка очереди }
    next := Buf[i mod K];
    Buf[i mod K] := elem;
      { начальные значения для max и maxSum }
    if i = K then begin
      max := next;
      maxSum := next + elem;
    end
      { обновление max и maxSum }
```

```
      else begin
        if next > max then max := next;
        if max+elem > maxSum then
           maxSum := max + elem;
      end;
    end;
    writeln(maxSum);
  end.
```

Можно было «втащить» начальное заполнение очереди внутрь цикла:

```
  const K = 7;
  var i, N, max, maxSum, next, elem: integer;
      Buf: array[0..K-1] of integer;
  begin
    read(N);
    for i:=0 to N-1 do begin
      read(elem);
        { обработка очереди }
      next := Buf[i mod K];
      Buf[i mod K] := elem;
        { начальные значения для max и maxSum }
      if i = K then begin
        max := next;
        maxSum := next + elem;
      end;
        { обновление max и maxSum }
      if i > K begin
        if next > max then max := next;
        if max+elem > maxSum then
           maxSum := max + elem;
      end;
    end;
    writeln(maxSum);
  end.
```

В этом решении при **i=K** происходит присвоение начальных значений, при **i>k** – обновление максимумов, а при **i<K** ни один условный оператор внутри цикла не срабатывает, и только новый элемент добавляется в очередь.

55) Эта задача полностью аналогична задаче 54 (см. подробный разбор выше), но здесь оговорено, что все результаты измерений – натуральные числа, не превышающие 1000. Это позволяет сразу задать начальные значения для переменных **min** (минимальное из предшествующих чисел, участвующих в поиске минимума) и **minRes** (минимальная сумма квадратов):

```
  min := 1001;       { любое число, большее, чем 1000 }
  minRes := 2000001; { любое число, большее, чем 2*1000*1000 }
```

Полная программа получается такая:

```
  const K = 5;
  var i, N, min, minRes, next, elem: integer;
      Buf: array[0..K-1] of integer;
  begin
    read(N);
      { заполняем очередь }
    for i:=0 to K-1 do read(Buf[i]);
      { начальные значения для min и minRes }
```

```
    min := 1001;
    minRes := 2000001;
      { обрабатываем оставшиеся данные }
    for i:=K to N-1 do begin
      read(elem);
        { обработка очереди }
      next := Buf[i mod K];
      Buf[i mod K] := elem;
        { обновление min и minRes }
      if next < min then min := next;
      if min*min+elem*elem < minRes then
         minRes := min*min+elem*elem;
    end;
    if minRes > 2000000 then
         write(-1)
    else writeln(minRes)
  end.
```

Как и в последнем варианте решения задачи 54, можно не выносить начальное заполнение очереди в отдельный цикл:

```
  const K = 5;
  var i, N, min, minRes, next, elem: integer;
      Buf: array[0..K-1] of integer;
  begin
    read(N);
    min := 1001;
    minRes := 2000001;
    for i:=0 to N-1 do begin
        { обновляем min, если i > K }
      if i >= K then
        if Buf[i mod K] < min then
          min := Buf[i mod K];
        { читаем новый элемент и добавляем в очередь }
      read(elem);
      Buf[i mod K] := elem;
        { обновляем minRes }
      if (i >= K) and
         (min*min+elem*elem < minRes) then
         minRes := min*min+elem*elem;
    end;
    if minRes > 2000000 then
         write(-1)
    else writeln(minRes)
  end.
```

56) Построим несколько вариантов правильных программ, начиная с простейшего и заканчивая оптимальным, за который можно получить 4 балла. Поскольку всего мы получем не более 10000 измерений, можно завести массив такого размера и прочитать в него все данные а потом их обрабатывать:

```
  var N, i: integer;
      data: array[1..10000] of real;
  begin
    readln(N);
    for i:=1 to N do
```

```
    readln(data[i]);
  { обработка массива }
end.
```

Обработка заключается в том, что нужно найти минимальное произведение двух элементов, отстоящих друг от друга на расстояние не менее 6, то есть для каждого элемента **data[i]** нужно рассмотреть произведения

data[i]*data[i+6], data[i]*data[i+7], …, data[i]*data[N]

и выбрать из них минимальное. Это можно сделать с помощью двойного цикла, обозначив индекс второго элемента через **j**:

```
const K = 6;
var min: real;
    i, j: integer;
...
min := 1000001; { любое число, большее 1000*1000 }
for i:=1 to N-K do
  for j:=i+K to N do
    if data[i]*data[j] < min then
      min := data[i]*data[j];
```

Здесь константа K обозначает минимальное расстояние, а начальное значение для переменной **min** выбрано так, чтобы оно было больше, чем 1000*1000, потому что по условию величина каждого измерения не больше 1000. Приведём полное решение:

```
const K = 6;
var data: array[1..10000] of real;
    min: real;
    N, i, j: integer;
begin
  readln(N);
  for i:=1 to N do
    readln(data[i]);
  min := 1000001;
  for i:=1 to N-K do
    for j:=i+K to N do
      if data[i]*data[j] < min then
        min := data[i]*data[j];
  writeln(min);
end.
```

Согласно критериям оценивания, приведенным в условии, это решение неэффективно по памяти, поскольку все данные записываются в массив. Кроме того, программа неэффективна по времени выполнения. Действительно, в двойном цикле

```
for i:=1 to N-K do
  for j:=i+K to N do
    ...
```

число шагов как внешнего, так и внутреннего циклов зависит от N, так что общее количество выполняемых операций пропорционально N^2, а не N. За такое решение можно получить только 2 балла.

Сначала подумаем, ка сделать программу эффективной по времени. Для этого обозначим через **i** номер *второго элемента* в паре (а не первого, как обычно). При этом минимальное произведение **data[j]*data[i]** будем искать «глядя назад», то есть, рассматривая все элементы от **data[1]** до **data[i-6]**:

```
for i:=K+1 to N do
  for j:=1 to i-K do
    if data[j]*data[i] < min then
```

```
        min := data[j]*data[i];
```

Пока мы ничего не выиграли. Однако заметим, что для каждого **i** нам нужно знать только минимальное из всех значений **data[j]** при **j** от 1 до **i-6**, а его легко вычислять по мере прохода по массиву (в следующей программе это переменная **minPrev**):

```
var minPrev: real;
...
minPrev := 1001;
for i:=K+1 to N do begin
  if data[i-K] < minPrev then
    minPrev := data[i-K];
  if minPrev*data[i] < min then
      min := minPrev *data[i];
end;
```

Начальное значение **minPrev** берём больше, чем маскимальное значение измерения (1000). В таком цикле время выполнения пропорционально N, то есть, решение эффективно по времени, но неэффективно по памяти (получаем 3 балла):

```
const K = 6;
var data: array[1..10000] of real;
    min, minPrev: real;
    N, i: integer;
begin
  readln(N);
  for i:=1 to N do
    readln(data[i]);
  min := 1000001;
  minPrev := 1001;
  for i:=K+1 to N do begin
    if data[i-K] < minPrev then { минимум из предыдущих }
      minPrev := data[i-K];
    if minPrev*data[i] < min then
      min := minPrev*data[i];
  end;
  writeln(min);
end.
```

Для того, чтобы получить максимальный балл (4 балла), нужно избавиться от массива, в котором хранятся результаты измерений. Заметим, что на каждом шаге цикла в предыдущей программе нам нужно только значение **data[i-K]**, то есть достаточно хранить в памяти только последние K введенных значения. Для этого заводим массив размером K:

```
var data: array[1..K] of real;
```

и сначала читаем в него K первых результатов:

```
readln(N);
for i:=1 to K do
  readln(data[i]);
```

Затем на каждом шаге цикла сравниваем **minPrev** с первым элементов массива:

```
if data[1] < minPrev then
  minPrev := data[1];
```

читаем очередной элемент

```
readln(x);
```

ищем минимум

```
if minPrev*x < min then
  min := minPrev*x;
```

сдвигаем массив данных так, чтобы первый элемент «ушёл», в последний осободился:

```
for j:=1 to K-1 do
  data[j] := data[j+1];
```

и записываем в последний элемент массива только что прочитанное значение:

```
data[K] := x;
```

В итоге получается такая программа, которая эффективна и по времени, и по памяти:

```
const K = 6;
var data: array[1..K] of real;
    min, minPrev, x: real;
    N, i, j: integer;
begin
  readln(N);
  for i:=1 to K do
    readln(data[i]);
  min := 1000001;
  minPrev := 1001;
  for i:=K+1 to N do begin
    if data[1] < minPrev then
      minPrev := data[1];
    readln(x);
    if minPrev*x < min then
      min := minPrev*x;
    for j:=1 to K-1 do
      data[j] := data[j+1];
    data[K] := x;
  end;
  writeln(min);
end.
```

Однако, её тоже можно улучшить. Дело в том, что сдвигать массив на каждом шаге цикла не обязательно. Достаточно записывать каждый полученный элемент на освободившееся место в массиве. Если начать нумерацию элементов массива не с 1, а с нуля,

```
var data: array[0..K-1] of real;
```

то получается совсем удобно: номер нужного элемента массива вычисляется как остаток от деления номера измерения **i** на K:

```
const K = 6;
var data: array[0..K-1] of real;
    min, minPrev, x: real;
    N, i: integer;
begin
  readln(N);
  for i:=1 to K do
    readln(data[i mod K]);
  min := 1000001;
  minPrev := 1001;
  for i:=K+1 to N do begin
    if data[i mod K] < minPrev then
      minPrev := data[i mod K];
    readln(x);
    if minPrev*x < min then
      min := minPrev*x;
    data[i mod K] := x;
  end;
  writeln(min);
```

```
end.
```

57) Эта задача полностью аналогична задаче 37. Наибольшее нечётное число получится при сложении наибольших чётного и нечётного числа, которые и нужно определить. Для проверки чётности будем использовать остаток от деления на 2.
Приведём правильное и эффективное решение. Обратите внимание, что начальные значения для переменных **chet** и **nechet** должны быт меньше минимального допустимого числа. Поскольку все числа по условию положительны, можно принять эти начальные значения равными нулю.

```
var chet, nechet, R, R0, i, N, x: longint;
begin
  chet := 0;
  nechet := 0;
  readln(N);
  for i := 1 to N do begin
    readln(x);
    if (x mod 2 = 0) and (x > chet) then chet := x;
    if (x mod 2 <> 0) and (x > nechet) then nechet := x;
  end;
  if (chet > 0) and (nechet > 0) then
       R := chet + nechet
  else R := -1;
  readln(R0);
  if R > 0 then
    writeln('Вычисленное контрольное значение: ', R);
  if (R > 0) and (R = R0)
    then writeln('Контроль пройден')
    else writeln('Контроль не пройден');
end.
```

58) Начнём с простейшего решения – задачи А. Сначала выделяем массив на максимальное возможное количество элементов (по условию – 10000):

```
var a: array[1..10000] of integer;
```

В начале программы читаем все данные в массив:

```
readln(N);
for i:=1 to N do
  read(a[i]);
```

Теперь в двойном цикле (по переменным **i** и **j**) нужно рассмотреть все возможные пары элементов (**a[i]**,**a[j]**), такие что **j-i** ≥ 8. Поэтому при выбранном значении меньшего индекса **i** второй индекс, j, будет изменяться от **i+8** до **N**. В свою очередь, меньший индекс (**i**) будет изменяться от 1 до **N-8**:

```
max:= 0;
for i:= 1 to N-8 do
  for j:= i+8 to N do
    if a[i]*a[j] > max then
       max := a[i]*a[j];
writeln(max)
```

Вот полное решение задачи А:

```
var N: integer;
    a: array[1..10000] of integer;
    i, j, max: integer;
begin
  readln(N);
```

```
    for i:=1 to N do read(a[i]);
    max:= 0;
    for i:= 1 to N-8 do
      for j:= i+8 to N do
        if a[i]*a[j] > max then
           max := a[i]*a[j];
    writeln(max)
end.
```

Такое решение «стоит» 2 балла. Попробуем улучшить его. Можно ли не перебирать все пары? Оказывается, можно. Например, рассмотрим все пары, в которых второй элемент – это элемент с номером **j**: (**a[i]**,**a[j]**). Поскольку все числа неотрицательны, из всех таких пар максимальное произведение имеет пара (**m**,**a[j]**), где **m** – максимальное из чисел **a[1]**, …, **a[j-8]**. Такой максимум можно искать постепенно, добавляя в зону поиска следующий элемент массива, **a[j-8]**, при рассмотрении нового значения **j**.

Вот решение на 3 балла (неэффективное по памяти, потому что все данные хранятся в массиве):

```
var N: integer;
    a: array[1..10000] of integer;
    m, j, max: integer;
begin
  readln(N);
  for j:=1 to N do read(a[j]);
  max:= 0;
  m:= 0;
  for j:= 9 to N do begin
    if a[j-8] > m then m := a[j-8];
    if m*a[j] > max then max := m*a[j];
  end;
  writeln(max)
end.
```

Более того, ВСЕ данные в массиве хранить вообще НЕ нужно. Достаточно запомнить только последние 8 значений в таком массиве:

```
const d = 8;
var a: array[1..d] of integer;
```

Здесь константа **d** – это смещение, равное 8. Сначала читаем первые 8 чисел в массив:

```
for j:=1 to d do read(a[j]);
```

Теперь читаем остальные данные (по одному числу)

```
for j:= d+1 to N do begin
  read(x);
  { обработка x }
end;
```

Обработка выполняется так же, как и в предыдущей версии программы, но вместо a[j] мы используе x, а вместо a[j-8] – значение a[1]:

```
if a[1] > m then m := a[1];
if m*x > max then max := m*x;
```

После этого значение **a[1]** нам уже не нужно, мы сдвигаем весь массив влево на 1 элемент и записываем в конец массива только что полученное значение **x**:

```
for i:=1 to d-1 do a[i]:=a[i+1];
a[d]:=x;
```

В итоге получается такая программа:

```
const d;
var N: integer;
```

```
    a: array[1..d] of integer;
    max, m, x, j, i: integer;
begin
  readln(N);
  for j:=1 to d do read(a[j]);
  max:= 0;
  m:= 0;
  for j:= d+1 to N do begin
    read(x);
    if a[1] > m then m := a[1];
    if m*x > max then max := m*x;
    for i:=1 to d-1 do a[i]:=a[i+1];
    a[d]:=x;
  end;
  writeln(max)
end.
```

Это решение задачи на 4 балла.

Возможен ещё вариант без сдвига массива. Здесь нумерация массива a выполняется с нуля, и на каждом шаге основного цикла идёт работа с элементом **a[j mod d]**, который представляет собой число, полученное на 8 шагов раньше текущего (только что прочитанного). Фактически вместо того, чтобы сдвигать элементы массива **a** и работать все время с первым из них, мы с помощью операции остатка сдвигаем номер рабочего элемента.

```
const d = 8;
var N: integer;
    a: array[0..d-1] of integer;
    max, m, x, j, i: integer;
begin
  readln(N);
  for j:=0 to d-1 do read(a[j]);
  max:= 0;
  m:= 0;
  for j:= d to N-1 do begin
    read(x);
    if a[j mod d] > m then m := a[j mod d];
    if m*x > max then max := m*x;
    a[j mod d]:=x;
  end;
  writeln(max)
end.
```

59) Эта задача полностью аналогична предыдущей, но требуется найти не максимум, а минимум. Это может вызвать некоторые сложности с выбором начального значения для переменной **min**. Оно должно быть таким, чтобы любая возможная сумма двух элементов последовательности была меньше, чем это начальное значение. Поскольку все числа не превышают 1000, можно выбрать это значение любым больше 2000, например, 2001.

Решение задачи A:

```
const d = 4;
var N: integer;
    a: array[1..10000] of integer;
    i, j, min: integer;
begin
```

```
  readln(N);
  for i:=1 to N do read(a[i]);
  min:= 2001;
  for i:= 1 to N-d do
    for j:= i+d to N do
      if a[i]+a[j] < min then
         min := a[i]+a[j];
  writeln(min)
end.
```

В задаче Б начальное значение переменной **m** должно быть больше любого элемента последовательности, например, 1001. Решение задачи Б:

```
const d = 4;
var N: integer;
    a: array[0..d-1] of integer;
    min, m, x, j, i: integer;
begin
  readln(N);
  for j:=0 to d-1 do read(a[j]);
  min:= 2001;
  m:= 1001;
  for j:= d to N-1 do begin
    read(x);
    if a[j mod d] < m then m := a[j mod d];
    if m+x < min then min := m+x;
    a[j mod d]:=x;
  end;
  writeln(min)
end.
```

60) Эта задача аналогична предыдущим. Начальное значение переменной **min** нужно выбрать таким, чтобы оно было больше, чем любое допустимое значение минимума. Поскольку все числа не превышают 1000, можно выбрать это значение любым больше 1000*1000, например, 1000001. Можно взять даже 1000000, поскольку это число чётное, то есть не может быть правильным подходящим произведением.

Решение задачи А:

```
const d = 6;
var N: integer;
    a: array[1..10000] of integer;
    i, j, min: integer;
begin
  readln(N);
  for i:=1 to N do read(a[i]);
  min:= 1000000; { 1000*1000 }
  for i:= 1 to N-d do
    for j:= i+d to N do
      if (a[i]*a[j] mod 2 = 1) and
         (a[i]*a[j] < min) then
        min := a[i]*a[j];
  if min = 1000000 then
       writeln(-1)
  else writeln(min)
end.
```

В задаче B есть одна особенность, связанная с дополнительным условием (чётность/нечётность произведения). Дело в том, что минимальное нечётное произведение – это всегда произведение двух нечётных чисел. Поэтому нужно искать минимальное **нечётное** число из предыдущих, отстоящих не менее чем на 6 отсчётов (переменная **m**). Начальное значение переменной **m** мы возьмем равным 10000 – это **чётное** значение; если оно останется в переменной **m**, это будет означать, что предыдущего нечётного числа **не было**. Минимум обновляется, если

а) только что прочитанное число **x** нечётное;

б) было найдено предыдущее нечётное число **m**;

в) произведение **x*m** меньше, чем ранее найденный минимум.

Решение задачи B:

```
const d = 6;
var N: integer;
    a: array[0..d-1] of integer;
    min, m, x, j, i: integer;
begin
  readln(N);
  for j:=0 to d-1 do read(a[j]);
  min:= 1000000;
  m:= 1000;
  for j:= d to N-1 do begin
    read(x);
    if ((a[j mod d] mod 2 = 1) and
        (a[j mod d] < m)) then m := a[j mod d];
    if (x mod 2 = 1) and (m mod 2 = 1) and
       (m*x < min) then min := m*x;
    a[j mod d]:=x;
  end;
  if min = 1000000 then
     writeln(-1)
  else
     writeln(min)
end.
```

61) Эта задача аналогична задаче 57.

Решение задачи A сводится к тому, что мы читаем все данные в массив, а затем проверяем все возможные пары, сумма которых кратна 3. Обратите внимание, что во внутреннем цикле переменная **j** изменяется от 1 до **i-1**, а не до N, чтобы не взять случайно в пару одно и то же число.

```
var R, R0, i, j, N: longint;
    a: array[1..10000] of integer;
begin
  readln(N);
  for i := 1 to N do readln(a[i]);
    { поиск контрольного значения }
  R := 0;
  for i := 1 to N do
    for j := 1 to i-1 do
      if ((a[i]+a[j]) mod 3 = 0) and (a[i]+a[j] > R) then
        R:= a[i]+a[j];
  readln(R0);
  if R = 0 then R:=1;
  writeln('Вычисленное контрольное значение: ', R);
```

```
    if R = R0
      then writeln('Контроль пройден')
      else writeln('Контроль не пройден');
  end.
```

Чтобы построить эффективное решение на полный балл (без массива), нужно сообразить, как найти максимальную сумму, кратную 3, не храня все числа в памяти. Сумма двух чисел может быть кратна трём в двух случаях:

а) оба числа делятся на 3

б) одно число при делении на 3 даёт в остатке 1, а второе – 2.

Поэтому нам нужно при обработке потока данных найти

а) два максимальных числа, кратных трём, **max3a** и **max3b**;

б) максимальное число, которое при делении на 3 даёт в остатке 1, **max1**;

в) максимальное число, которое при делении на 3 даёт в остатке 2, **max2**.

Поскольку все входные данные положительные, начальные значение этих четырёх переменных можно взять равными нулю.

Контрольное значение – это максимальная из сумм **max3a+max3b** и **max1+max2**. Нужно только не забыть, что подходящей пары может не существовать. Если хотя бы одна из переменных **max1** или **max2** осталавь равной нулю, то нет пары **max1+max2**. Аналогично если **max3b** (второе по величине число, делящееся на 3) осталось равно нулю, нет пары **max3a+max3b**. Поэтому контрольное значение строим так:

```
  R := 1;
  if max1*max2 > 0 then R:= max1+max2;
  if (max3b > 0) and  (max3a+max3b > R) then
    R:=max3a+max3b;
```

Если в R осталась 1, то ни одной подходящей пары нет (потому что любая сумма двух положительных чисел больше 1).

Вот полная программа:

```
  var R, R0, i, max1, max2, max3a, max3b, N, x: longint;
  begin
    readln(N);
    max1 := 0; max2 := 0;
    max3a := 0; max3b := 0;
    for i := 1 to N do begin
      readln(x);
      if x mod 3 = 0 then begin
        if x > max3a then begin
          max3b:= max3a;
          max3a:= x
        end
        else
          if x > max3b then max3b:=x;
      end;
      if x mod 3 = 1 then
        if x > max1 then max1:=x;
      if x mod 3 = 2 then
        if x > max2 then max2:=x;
    end;
    R := 1;
    if max1*max2 > 0 then R:= max1+max2;
    if (max3b > 0) and  (max3a+max3b > R) then
      R:=max3a+max3b;
    readln(R0);
```

```
    if R > 0 then
      writeln('Вычисленное контрольное значение: ', R);
    if (R > 0) and (R = R0)
      then writeln('Контроль пройден')
      else writeln('Контроль не пройден');
  end.
```

Интересное и более короткое решение предложил **М.П. Стручков** (доработано автором этих материалов). Идея состоит в том, чтобы хранить в массиве (назовём его «**a**») три максимальных числа:

0) максимальное из чисел, делящихся на 3;
1) максимальное из чисел, дающих остаток 1 при делении на 3;
2) максимальное из чисел, дающих остаток 2 при делении на 3;

Для только что прочитанного числа **x** определяем остаток **k** от деления на 3.
Тогда сумма-кандидат, делящаяся на 3, которую нужно проверить, – это:

x+a[0], если **k=0**
x+a[2], если **k=1**
x+a[1], если **k=2**

Конечно, если второе слагаемое в сумме больше нуля, то есть нужное парное число к только что полученному **x** уже встретилось в последовательности ранее. Остаток **k1**, соответствующий числу-паре, можно было бы вычислить как **3-k**, но эта формула не подходит при **k=0**. Поэтому можно написать так

```
  if k = 0 then k1:=0 else k1:=3-k;
```

или так:

```
  k1:= (3-k) mod 3;
```

```
  var a:array[0..2] of integer;
      N,i,k,k1,x,max,R:integer;
  begin
    max:=-1;
    readln (N);
    for i:=0 to 2 do a[i]:=0;
    for i:=1 to N do begin
      readln(x);
      k:= x mod 3;
      k1:=(3-k) mod 3;
      if (a[k1]>0) and (a[k1]+x>max) then
        max:=a[k1]+x;
      if x > a[k] then a[k]:=x;
    end;
    if max = -1 then max:=1;
    readln (R);
    writeln('Вычисленное контрольное значение: ', max);
    if max = R then
         writeln ('Контроль пройден')
    else writeln ('Контроль не пройден');
  end.
```

62) Для того чтобы декодировать сообщение, нужно сначала получить битовую цепочку, переводя введённое число **N** в двоичную систему. Причём потом эту цепочку нам придётся разворачивать, поэтому можно сразу получить развёрнутую цепочку, добавляя остаток от деления в конец числа. Для хранения двоичной последовательности будем использовать символьную строку **s**. Переменные, которые будут нужны:

```
var N: integer;       { переданное число }
    binCode: string;  { двоичный код }
    s: string;        { декодированное сообщение }
    count: integer;   { счётчик гласных }
```

Переводим в двоичный код делением на 2, остатки записываем в конец числа, так что оно сразу «разворачивается»:

```
binCode:= '';
while N > 0 do begin
  binCode:= binCode + chr(ord('0')+(N mod 2));
  N:= N div 2;
end;
```

Здесь **ord('0')** – это код символа **'0'**, тогда **ord('0')+(N mod 2)** – это код очередной цифры (0 или 1), а **chr(ord('0')+(N mod 2))** – это соответствующий символ.

Дальше стандартное декодирование:

1. определяем начальный символ строки;
2. если это 0, получили букву E, увеличиваем счётчик гласных, откусываем первую букву строки;
3. если первый символ 1, смотрим на второй, если он 0, то это буква P, если 1 – то буква A (тут нужно увеличить счётчик.

```
if binCode[1] = '0' then begin
  s:= s + 'E';
  count:= count + 1;
    Delete(binCode, 1, 1);
end
else begin
  if binCode[2] = '0' then s:= s + 'P'
  else begin
    s:= s + 'A';
    count:= count + 1;
  end;
  Delete(binCode, 1, 2);
end;
```

Когда будут декодированы все цифры, в конце останется единица, которая добавляется к полезной цепочке согласно алгоритму кодирования. Это значит, что длина цепочки станет равна 1 и второго символа там нет. Эту ситуацию нужно предусмотреть: условие цикла выглядит так:

```
while Length(binCode) > 1 do begin
  ...
end;
```

Вот полное решение задачи:

```
var N, count: integer;
    binCode, s: string;
begin
  readln(N);
  binCode:= '';
  while N > 0 do begin
    binCode:= binCode + chr(ord('0')+(N mod 2));
    N:= N div 2;
  end;
  s:= '';
```

```
    while Length(binCode) > 1 do begin
      if binCode[1] = '0' then begin
        s:= s + 'E';
        count:= count + 1;
        Delete(binCode, 1, 1);
      end
      else begin
        if binCode[2] = '0' then s:= s + 'P'
        else begin
          s:= s + 'A';
          count:= count + 1;
        end;
        Delete(binCode, 1, 2);
      end;
    end;
    writeln(s);
    writeln(count);
  end.
```

Альтернативное решение, использующее битовые логические операции, предложила **А.А. Фахуртдинова** (г. Москва). Вспомним, что любое значение хранится в памяти компьютера в двоичном коде.

Декодировать начнем с младшего разряда. При этом учтем, что коды символов будут перевернуты: E-0; A-11; P-01. Таким образом, если младший бит равен 0, то есть

```
(N and 1) = 0
```

то это буква E: нужно добавить её к строке **s**, увеличить счётчик гласных выполнить сдвиг вправо на 1 бит с помощью оператора **shr**, таким образом имитируя работу сдвигового регистра:

```
if (N and 1) = 0 then begin {E}
  s := s + 'E';
  count := count + 1;
  N := N shr 1;
end
```

Если же младший бит не равен 0, проверяем последние два бита по маске $3 = 11_2$, если они оба равны 1, то это буква P, иначе – буква A

```
...
else begin
   if (N and 3) = 3 then begin {A}
     s := s + 'A';
     count := count + 1;
   end
   else {P}
     s := s + 'P';
   N := N shr 2;
end;
```

Обратите внимание, что при получении букв P и A сдвиг вправо выполняется на 2 бита:

```
N := N shr 2;
```

Когда будут декодированы все символы, декодируемое число станет равно 1 (это «стоп-бит», который был дописан в конец числа и оказался в начале двоичного кода после переворачивания). Тогда условие цикла будет таким:

```
While a <> 1 do
  ...
end;
```

Вот полное решение задачи:

```
Var N:longint;
    s:string;
    count:integer;
begin
  readln(N);
  s := ''; count := 0;
  While N <> 1 do begin
    if (N and 1) = 0 then begin {E}
      s := s + 'E';
      count := count + 1;
      N := N shr 1;
    end
    else begin
      if (N and 3) = 3 then begin {A}
        s := s + 'A';
        count := count + 1;
      end
      else {P}
        s := s + 'P';
      N := N shr 2;
    end;
  end;
  Writeln(s);
  Writeln(count)
end.
```

Ещё одно решение предложил **Ю.В. Ковалевич**. Так как нам необходимо перевести входящее натуральное число в двоичный код, а затем его перевернуть, то мы можем сразу анализировать цепочку остатков, полученную при делении на 2. В «перевёрнутом виде» E – 0; P – 01; A – 11.

Применяем стандартный алгоритм разделения числа на цифры. (Нашли цифру, уменьшили число, проверили, остались ли цифры в числе). Но мы должны убрать последнюю (добавленную) цифру 1, поэтому в условии продолжения цикла записываем N > 1, а не N > 0.

При анализе остатков сначала проверяем наличие нуля. Если нашли в конце 0 – это буква «E». Если остаток равен 1, то нужно узнать какой следующий остаток будет. Поэтому уменьшаем N в 2 раза и анализируем следующий остаток, по нему определяя, какая буква будет записана в результирующую строку – «P» или «A». Не забываем накапливать результирующую строку и счетчик гласных букв.

```
var N, count: integer;
    s: string;
begin
  readln(n);
  count:=0;
  while N > 1 do begin
    if N mod 2=0 then begin
       s:= s + 'E';
       count:= count + 1;
    end
    else begin
      N:= N div 2;
      if N mod 2=0 then
```

```
          s:= s + 'P'
        else begin
          s:= s + 'A';
          count:= count + 1;
        end;
      end;
      N:= N div 2;
    end;
    writeln(s);
    writeln(count);
  end.
```

63) Эта задача аналогична задаче 60, только требуется найти максимальное чётное произведение вместо минимального нечётного. Начальное значение переменной **max** нужно выбрать таким, чтобы оно было меньше, чем любое допустимое значение максимума. Поскольку все числа неотрицательны, можно выбрать это значение любым больше меньше 0, например, -1. Вариант с -1 хорош ещё и потому, что именно -1 нужно вывести, если чётного произведения найти не удалось (нет ни одного чётного числа).

Решение задачи A:

```
const d = 9;
var N: integer;
    a: array[1..10000] of integer;
    i, j, max: integer;
begin
  readln(N);
  for i:=1 to N do read(a[i]);
  max:= -1;
  for i:= 1 to N-d do
    for j:= i+d to N do
      if (a[i]*a[j] mod 2 = 0) and
         (a[i]*a[j] > max) then
        max := a[i]*a[j];
  writeln(max)
end.
```

В задаче B есть одна особенность, связанная с дополнительным условием (чётность/нечётность произведения). Дело в том, что максимальное чётное произведение – это произведение максимального чётного числа на максимум из оставшихся (чётных и нечётных) чисел. Поэтому нужно искать максимальное **чётное** число из предыдущих, отстоящих не менее чем на 9 отсчётов (переменная **mEven**) и максимальное их всех предыдущих (также отстоящих не менее чем на 9 отсчётов), эту переменную назовём **m**.

Если только что прочитанное число **x** нечётное, сравнивать с максимальным произведением нужно **x*mEven**, поскольку второй сомножитель может быть только чётным. Если же прочитали чётное **x**, проверяем произведение **x*m**, потому что второй сомножитель может быть любым.

Решение задачи B:

```
const d = 9;
var N: integer;
    a: array[0..d-1] of integer;
    max, m, mEven, x, j, i: integer;
begin
  readln(N);
```

```
    for j:=0 to d-1 do read(a[j]);
    max:= -1;
    m:= -1; mEven:= -1;
    for j:= d to N-1 do begin
      read(x);
      if ((a[j mod d] mod 2 = 0) and
          (a[j mod d] > mEven)) then mEven:= a[j mod d];
      if (a[j mod d] > m) then m := a[j mod d];
      if x mod 2 = 1 then begin
        if x*mEven > max then max := mEven*x;
      end
      else
        if x*m > max then max := m*x;
      a[j mod d]:=x;
    end;
    writeln(max)
  end.
```

64) Особенность этой задачи состоит в том, что нужно определить наибольшее произведение, которое **НЕ делится** на какое-то число (здесь – 26).

Решение задачи А (с записью чисел в массив) не вызывает проблем:

```
  var N: integer;
      a: array[1..10000] of integer;
      i, j, max, R: integer;
  begin
    readln(N);
    for i:=1 to N do read(a[i]);
    read(R);
    max:= -1;
    for i:= 1 to N-1 do
      for j:= i+1 to N do
        if (a[i]*a[j] mod 26 <> 0) and
           (a[i]*a[j] > max) then
             max := a[i]*a[j];
    if max > 0 then
      writeln('Вычисленное контрольное значение: ', max);
    if R = max then
      writeln('Контроль пройден.')
    else
      writeln('Контроль не пройден.');
  end.
```

При решении задачи В записыват числа в массив нельзя, они поступают из входного потока и обрабатываются по одному.

Понятно, что все числа, которые делятся на 26, не нужно учитывать вообще.

Кроме того, число 26 раскладывается на простые сомножители единственным способом: **26 = 13·2**. Тогда, если числа **a** и **b** не делятся на 26, их произведение **a ·b** может делиться на 26, если одно из чисел делится на 2, а второе – на 13. Такие варианты нам нужно исключить.

Таким образом, нас интересуют только те пары **(a,b)**, для которых выполняется одно из трёх условий:

а) ни одно из двух чисел не делится ни на 2, ни на 13;
б) одно из двух чисел делится на 2, второе НЕ делится на 13;
в) одно из двух чисел делится на 13, второе НЕ делится на 2.

Таким образом, все поступающие числа, не делящиеся на 26, можно разделить на 3 группы:

Группа 0. Числа, которые не делятся ни на 2, ни на 13.

Группа 2. Числа, которые делятся на 2, но не делятся на 13.

Группа 13. Числа, которые делятся на 13, но не делятся на 2.

Поскольку нам нужно найти наибольшее из таких произведений, при обработка входного потока достаточно сохранять в памяти два наибольших числа из каждой группы. Для этого будет использовать переменные **m0_1**, **m0_2** (для Группы 0) , **m2_1**, **m2_2** (для Группы 2) и **m13_1**, **m13_2** (для Группы 13). При этом вычисленное значение R – это наибольшее из чисел

```
m0_1*m0_2, m0_1*m2_1, m0_1*m13_1,
m2_1*m2_2, m13_1*m13_2
```

Для того чтобы найти два наибольших числа в потоке, используем стандартный алгоритм. Например, для чисел, которые делятся на 2, но не делятся на 13, он выглядит так:

```
if a > m2_1 then begin
  m2_2 := m2_1; m2_1 := a;
end
else if a > m2_2 then m2_2 := a;
```

Если полученное число **a** больше, чем максимум, копируем «старый» максимум на место второго максимума (в переменную **m2_2**) и записываем в «первый» максимум полученное значение **a**. Если a не больше «старого» максимума, проверяем, не будет ли оно теперь вторым максимумом, в этом случае записываем его в переменную **m2_2**. Аналогичные условные операторы нужно записать и для остальных групп рассматриваемых чисел.

При вычислении контрольного значения есть единственная тонкость – нужно учесть, что в какой-то группе может вооще не оказаться ни одного числа. Поскольку все входные данные положительные, в переменные, которые хранят наибольшие значения в каждой группе, можно сначала записать нули. Тогда если вычисленное контрольное значение окажется равным нулю, это будет означает, что определить его не удалось.

Решение задачи B:

```
var N: integer;
    m0_1, m0_2, m2_1, m2_2,
    m13_1, m13_2, a, i, max, R: integer;
begin
  readln(N);
  m0_1 := 0; m0_2 := 0;
  m2_1 := 0; m2_2 := 0;
  m13_1 := 0; m13_2 := 0;
  for i:=1 to N do begin
    read(a);
    if a mod 26 <> 0 then begin
      if a mod 2 = 0 then begin
        if a > m2_1 then begin
          m2_2 := m2_1; m2_1 := a;
        end
        else if a > m2_2 then m2_2 := a;
      end
      else if a mod 13 = 0 then begin
        if a > m13_1 then begin
          m13_2 := m13_1; m13_1 := a;
        end
```

```
          else if a > m13_2 then m13_2 := a;
        end
        else begin
          if a > m0_1 then begin
            m0_2 := m0_1; m0_1 := a;
          end
          else if a > m0_2 then m0_2 := a;
        end
      end
    end;
    read(R);

    max := 0;
    if m0_1*m0_2 > max then max := m0_1*m0_2;
    if m0_1*m2_1 > max then max := m0_1*m2_1;
    if m0_1*m13_1 > max then max := m0_1*m13_1;

    if m2_1*m2_2 > max then max := m2_1*m2_2;
    if m13_1*m13_2 > max then max := m13_1*m13_2;

    if max > 0 then
      writeln('Вычисленное контрольное значение: ', max);
    if R = max then
      writeln('Контроль пройден.')
    else
      writeln('Контроль не пройден.');
  end.
```

Красивое и короткое решение предложил **В. Бабий** (Информатик БУ). Идея состоит в том, чтобы вычислять максимум прямо в цикле. Здесь используются две вспомогательные переменные:

m2 – максимальное из полученных ранее значений, НЕ делящееся на 2

m13 – максимальное из полученных ранее значений, НЕ делящееся на 13.

При получении очередного значения **a** из входного потока нужно проверить два произведения, **m2*a** и **m13*a**. Если какое-то из них не делится на 26 и больше максимума **max**, найденного ранее, максимум заменяется.

Решение задачи B:

```
  var N: integer;
      m2, m13, a, i, max, R: integer;
  begin
    readln(N);
    m2 := 0; m13 := 0;  max := 0;
    readln(a);
    for i:=2 to N do begin
      if (a > m2) and (a mod 2 <> 0) then
         m2 := a;
      if (a > m13) and (a mod 13 <> 0) then
         m13 := a;
      readln(a);
      if (m2*a > max) and (m2*a mod 26 <> 0) then
         max := m2*a;
      if (m13*a > max) and (m13*a mod 26 <> 0) then
```

```
        max := m13*a;
    end;
    read(R);
    if max > 0 then
      writeln('Вычисленное контрольное значение: ', max);
    if R = max then
      writeln('Контроль пройден.')
    else
      writeln('Контроль не пройден.');
end.
```

Ещё одно решение задачи B на 4 балла (**А. Фахуртдинова, Москва**).

Попробуем проверить, какие комбинации чисел нам подходя. Построим таблицу, где нулями и единицами отметим делимость первого и второго числа в паре на 13 и на 2:

	число 1		число 2	
	делится на 13	делится на 2	делится на 13	делится на 2
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	1	0	0
5	0	1	0	1
6	1	0	0	0
7	1	0	1	0

Из этой таблицы видно, что во всех устраивающих нас варианта либо

- оба числа НЕ делятся на 13 (случаи 1, 2, 4, 5)
- оба числа НЕ делятся на 2 (случаи 1, 3, 6, 7)

Таким образом, идея решения состоит в том, чтобы найти

- первый и второй максимумы, не кратные 13 (**x1**, **x2**)
- первый и второй максимумы, не кратные 2 (**y1**, **y2**).

И за контрольное значение принимается большее из произведений **x1*x2** или **y1*y2**.

```
var x,x1,x2,y1,y2,n,i:integer;
pr, r:longint;
begin
  x1:=0; x2:=0; y1:=0; y2:=0;
  readln(n);
  for i:=1 to n do begin
    readln(x);
    if x mod 13 <> 0 then
      if x > x1 then begin
         x2:=x1; x1:=x
      end
      else if x > x2 then x2:=x;
    if x mod 2 <> 0 then
      if x>y1 then begin
           y2:=y1; y1:=x
      end
      else if x>y2 then y2:=x
  end;
  if x1*x2 > y1*y2 then pr:= x1*x2
  else pr:= y1*y2;
  readln(r);
  writeln('Вычисленное контрольное значение: ',pr);
```

```
    if pr = r then
         writeln('Контроль пройден')
    else writeln('Контроль не пройден')
end.
```

65) **Решение задачи А** можно построить очень просто: записать в память все данные и выполнить полный перебор всех возможных вариантов. Можно использовать, например, матрицу – двухмерный массив, в котором 6 строк и 2 столбца

```
var a: array[1..6,1..2] of integer;
```

Читаем данные и выполняем полный перебор вариантов:

```
var a: array[1..6,1..2] of integer;
  r, i1, i2, i3, i4, i5, i6: integer;
  sum, maxSum: integer;
begin
  for r:=1 to 6 do
    readln(a[r,1], a[r,2]);
  maxSum := 0;
  for i1:=1 to 2 do
    for i2:=1 to 2 do
      for i3:=1 to 2 do
        for i4:=1 to 2 do
          for i5:=1 to 2 do
            for i6:=1 to 2 do begin
              sum := a[1,i1] + a[2,i2] + a[3,i3]
                   + a[4,i4] + a[5,i5] + a[6,i6];
              if (sum mod 4 <> 0) and (sum > maxSum) then
                maxSum := sum;
            end;
   writeln(maxSum);
end.
```

Решение задачи Б. Чтобы понять идею решения, сначала не будем учитывать ограничение «сумма не должна делиться на 4». В этом случае ответом будет просто сумма наибольших чисел из каждой пары, которую легко найти:

```
var N, i, x1, x2, sum: integer;
begin
  Readln(N);
  sum := 0;
  for i:=1 to N do begin
    readln(x1, x2);
    if x1 > x2 then
         sum := sum + x1
    else sum := sum + x2;
  end;
  writeln(sum);
end.
```

Что нужно изменить, если полученная таким образом сумма делится на 4? Попробуем заменить одно число. Например, если сумма **sum** делится на 4 и в эту сумму входит число **a**, которое тоже делится на 4, то достаточно заменить **a** на другое число, которое не делится на 4. Если же число **a** НЕ делится на 4, его нужно заменить числом, которое при делении на 4 даёт другой остаток, в этом случае вся сумма не будет делится на 4. Чтобы сумма осталась максимально возможной, для замены нужно выбрать такую пару из входных данных, в которой разница между двумя числами минимальная. Таким образом,

нам нужно во время обработки данных входного потока искать пару чисел, для которой выполняются два условия:

1) числа в паре дают разные остатки при делении на 4;
2) разность **delta** между двумя числами в паре минимальна.

Первое из этих условий можно записать в виде

```
x1 mod 4 <> x2 mod 4
```

или, что то же самое,

```
(x1-x2) mod 4 <> 0
```

то есть разность двух чисел в паре не делится на 4.

По условию все числа положительные и не больше 10000, поэтому их разность всегда будет меньше 10000. Следовательно, начальное значение для **delta** можно взять любое, не меньшее, чем 10000, например, 10001 (можно даже взять и 10000). Получается такая программа:

```
Readln(N);
sum := 0;
delta := 10001;
for i:=1 to N do begin
  readln(x1, x2);
  if x1 > x2 then
       sum := sum + x1
  else sum := sum + x2;
  if ((x1-x2) mod 4 <> 0) and
     (abs(x1-x2) < delta) then
     delta := abs(x1-x2);
end;
```

Если сумма **sum**, полученная в результат работы цикла, не делится на 4, следует просто вывести эту сумму. Если делится, нам нужно определить, есть ли возможность замены. Если такой возможности нет, то условный оператор, выделенный в предыдущей программе жёлтым маркером, ни разу не сработал, и в переменной **delta** осталось начальное значение 100001. Если замена возможна, в результате такой замены сумму нужно уменьшить на разность между двумя числами в паре, в которой эта замена делается, то есть на **delta**:

```
if sum mod 4 <> 0 then
  writeln(sum)
else
  if delta = 10001 then
    writeln(0)
  else writeln(sum-delta);
```

Можно сделать то же самое немного более красиво, используя только один оператор вывода:

```
if sum mod 4 = 0 then
  if delta = 10001 then
       sum := 0
  else sum := sum - delta;
writeln(sum);
```

Приведём полное решение задачи Б:

```
var N, i, x1, x2, sum: integer;
    d, delta: integer;
begin
  Readln(N);
  sum := 0;
  delta := 10001;
```

```
    for i:=1 to N do begin
      readln(x1, x2);
      if x1 > x2 then
           sum := sum + x1
      else sum := sum + x2;
      d := abs(x1-x2);
      if (d mod 4 <> 0) and (d < delta) then
         delta := d;
    end;
    if sum mod 4 = 0 then
      if delta = 10001 then
           sum := 0
      else sum := sum - delta;
    writeln(sum);
  end.
```

Аналогичное решение на Python:

```
N = int(input())
sum = 0
delta = 10001
for i in range(N):
  x1, x2 = map(int, input().split())
  sum += max(x1,x2)
  d = abs(x1-x2)
  if d % 4 != 0 and d < delta:
     delta = d
if sum % 4 == 0:
  if delta == 10001:
        sum = 0
  else: sum = sum - delta
print(sum);
```

66) Поскольку все числа положительные, идеальное решение – это сумма всех чисел. Но при этом может получиться, что эта сумма делится на 4, что запрещено. В этом случае нужно убрать одно число так, чтобы сумма осталась наибольшей из возможных и не делилась на 4. Понятно, что нужно убрать минимальное число, которое не делится на 4. В этом случае количество чисел уменьшится на 1.

В простейшем случае можно прочитать все числа в массив (их количество известно, $N \le 1000$)

```
var A: array[1..1000] of integer;
    N, i, min: integer;
    sum: longint;
...
for i:=1 to N do
  readln(A[i]);
```

и затем искать в массиве наименьшее число, не кратное 4, попутно вычисляя сумму всех чисел:

```
sum := 0;
min = 10001; { любое, большее 10000 }
for i:=1 to N do begin
  sum := sum + A[i];
  if (A[i] mod 4 <> 0) and (A[i] < min) then
     min := A[i];
```

```
  end;
```

Однако понятно, что поиск минимального и суммирование элементов последовательности – это однопроходные алгоритмы, и для этого не нужно хранить все значения в массиве, их можно обрабатывать последовательно по мере поступления данных:

```
  min := 10001; { любое, большее 10000 }
  sum := 0;
  for i:=1 to N do begin
    readln(x);
    sum := sum + x;
    if (x mod 4 <> 0) and (x < min) then
       min := x;
  end;
```

После завершения цикла проверяем делимость суммы на 4, если делится, выводим результат как есть (сумма всех N значений), если не делится, вычитаем найденное минимальное значение и уменьшаем на 1 количество чисел в сумме. Если в сумме нет ни одного числа, не делящегося на 4 (значение **min** осталось равным 10001), выводим нули:

```
    if sum mod 4 <> 0 then
      writeln(N, ' ', sum)
    else
      if min = 10001 then writeln('0 0')
      else writeln(N-1, ' ', sum-min);
```

Вот полная программа, которая эффективна по памяти и по времени:

```
var N, x, min, i: integer;
    sum: longint;
begin
  read(N);
  min := 10001; { любое, большее 10000 }
  sum := 0;
  for i:=1 to N do begin
    readln(x);
    sum := sum + x;
    if (x mod 4 <> 0) and (x < min) then
       min := x;
  end;
  if sum mod 4 <> 0 then
    writeln(N, ' ', sum)
  else
    if min = 10001 then writeln('0 0')
    else writeln(N-1, ' ', sum-min);
end.
```

67) В этой задаче нужно ввести массив счётчиков, с помощью которые будем считать, сколько раз встречается каждая цифра:

```
  var count: array[0..9] of integer;
```

В начале программе на всякий случай не забудем обнулить счётчики:

```
  for i:=0 to 9 do count[i] := 0;
```

После того, как прочитали очередное число, нужно разбить его на цифры и изменить значения счётчиков. Для этого удобно использовать процедуру, которая изменяет глобальный массив **count**.

```
  procedure Digits(x: integer);
  var d: integer;
```

```
  begin
    while x > 0 do begin
      d := x mod 10;
      count[d] := count[d] + 1;
      x := x div 10;
    end;
  end;
```

Основной цикл получается очень простой: читаем число и обрабатываем егопроцедурой **Digits**.

```
  for i:=1 to N do begin
    readln(x);
    Digits(x);
  end;
```

Теперь нужно найти минимум в массиве **count**:

```
  max := 0;
  for i:=0 to 9 do
    if count[i] > max then max := count[i];
```

и вывести все цифры, для которых счётчик равен этому максимуму:

```
  for i:=9 downto 0 do
    if count[i] = max then
      write(i, ' ');
```

Обратите внимание, что для того, чтобы цифры были выведены, начиная со старшей, переменная цикла уменьшается с 9 до 0.

Приведём полную программу:

```
  var N, x, i, max: integer;
      count: array[0..9] of integer;
  procedure Digits(x: integer);
  var d: integer;
  begin
    while x > 0 do begin
      d := x mod 10;
      count[d] := count[d] + 1;
      x := x div 10;
    end;
  end;
  begin
    read(N);
    for i:=0 to 9 do count[i] := 0;
    for i:=1 to N do begin
      readln(x);
      Digits(x);
    end;
    max := 0;
    for i:=0 to 9 do
      if count[i] > max then max := count[i];
    for i:=9 downto 0 do
      if count[i] = max then
        write(i, ' ');
  end.
```

68) В этой задаче нужно ввести массив счётчиков, с помощью которые будем считать, сколько раз встречается каждая цифра в начале чисел:

```
  var count: array[1..9] of integer;
```

Поскольку ни одно число не может начинаться с нуля, счётчики нужны только для цифр от 1 до 9 включительно. В начале программе на всякий случай не забудем обнулить счётчики:

```
for i:=1 to 9 do count[i] := 0;
```

После того, как прочитали очередное число, нужно получить его первую цифру. Для этого можно делить его на 10 до до тех пор, пока не получится число, меньшее 10:

```
while x >= 10 do
  x := x div 10;
```

После этого нужно увеличить соответствующий счётчик. Для этого удобно использовать процедуру, которая изменяет глобальный массив **count**.

```
procedure Digits(x: integer);
begin
  while x > 10 do
    x := x div 10;
  count[x] := count[x] + 1;
end;
```

Основной цикл получается очень простой: читаем число и обрабатываем его процедурой **Digits**.

```
for i:=1 to N do begin
  readln(x);
  Digits(x);
end;
```

Теперь нужно найти минимум в массиве **count**, учитывая только те элементы, для которых **count[i] > 0**. При этом будем сразу запоминать номер найденного минимального элемента (то есть цифру!) в переменной **iMin**:

```
min := N+1;
for i:=1 to 9 do
  if (count[i] > 0) and (count[i] < min) then begin
    min := count[i];
    iMin := i;
  end;
```

Начальное значение переменной **min** равно **N+1**, потому что счётчики не могут быть больше, чем **N** (в крайнем случае все N чисел начинаются с одной и той же цифры).

Из-за того, что условный оператор срабатывает при строгом неравенстве **count[i] < min**, в переменной **iMin** окажется номер **первого** элемента с минимальным значением, то есть младшей цифры. Теперь нужно вывести найденную (наименьшую) цифру, то есть значение **iMin**.

Приведём полную программу:

```
var N, x, i, min, iMin: integer;
    count: array[1..9] of integer;
procedure Digits(x: integer);
begin
  while x >= 10 do
    x := x div 10;
  count[x] := count[x] + 1;
end;
begin
  read(N);
  for i:=1 to 9 do count[i] := 0;
  for i:=1 to N do begin
    readln(x);
    Digits(x);
```

```
    end;
    min := N+1;
    for i:=1 to 9 do
      if (count[i] > 0) and (count[i] < min)
        then begin
        min := count[i];
        iMin := i;
      end;
    writeln(iMin);
  end.
```

Красивое решение этой задачи, использующее символьные переменные, предложила **Е.В. Беляева (**г. Ульяновск):

```
  var a:array['1'..'9'] of integer;
      i,c,n,min:integer;
      k:string;
      j,p:char;
  begin
    readln(n);
    for i:=1 to n do begin
      readln(c);
      str(c,k);
      inc(a[k[1]])
    end;
    min:=n;
    for j:='1' to '9' do
      if (a[j]<min) and (a[j]<>0) then begin
        min:=a[j];
        p:=j
      end;
    writeln(p)
  end.
```

69) Заметим, что все числа не больше 1000, поэтому сумма цифр может быть от 0 до 27. В этой задаче нужно ввести массив счётчиков, с помощью которые будем считать, сколько раз встречается каждая сумма:

```
  var count: array[0..27] of integer;
```

В начале программе не забудем обнулить счётчики:

```
  for i:=0 to 27 do count[i] := 0;
```

После того, как прочитали очередное число, нужно разбить его на цифры и найти сумму этих цифр. Для этого будем применим стандартный алгоритм, использующий операции деления нацело и взятия остатка:

```
  sum := 0;
  while x > 0 do begin
    sum := sum + x mod 10;
    x := x div 10;
  end;
```

После этого нужно увеличить соответствующий счётчик. Для этого удобно использовать процедуру, которая изменяет глобальный массив **count**.

```
  procedure SumDigits(x: integer);
  var sum: integer;
  begin
    sum := 0;
    while x > 0 do begin
```

```
      sum := sum + x mod 10;
      x := x div 10;
    end;
    count[sum] := count[sum] + 1;
  end;
```

Основной цикл получается очень простой: читаем число и обрабатываем его процедурой **SumDigits**.

```
  for i:=1 to N do begin
    readln(x);
    SumDigits(x);
  end;
```

Теперь нужно найти максимум в массиве **count** и сразу запомнить в переменной **iMax** номер найденного максимального элемента (то есть сумму цифр!):

```
    max := 0;
    for i:=27 downto 0 do
      if count[i] > max then begin
        max := count[i];
        iMax := i;
      end;
```

Обратите внимание, что перебор элементов происходит с конца массива. Из-за того, что условный оператор срабатывает при строгом неравенстве **count[i] > max**, в переменной **iMax** окажется номер **последнего** элемента с максимальным значением, то есть наибольшей суммы цифр. Теперь нужно вывести значение **iMax**.

Приведём полную программу:

```
var N, x, i, iMax, max: integer;
    count: array[0..27] of integer;
procedure SumDigits(x: integer);
var sum: integer;
begin
  sum := 0;
  while x > 0 do begin
    sum := sum + x mod 10;
    x := x div 10;
  end;
  count[sum] := count[sum] + 1;
end;
begin
  read(N);
  for i:=0 to 27 do count[i] := 0;
  for i:=1 to N do begin
    readln(x);
    SumDigits(x);
  end;
  max := 0;
  for i:=27 downto 0 do
    if count[i] > max then begin
      max := count[i];
      iMax := i;
    end;
  writeln(iMax);
end.
```

70) (**Д. Муфаззалов, Уфа**) В этой задаче нужно ввести массив счётчиков, с помощью которые будем считать, сколько раз встречается каждая цифра. В начале программы необходимо обнулить счётчики. После того, как прочитали очередное число, нужно разбить его на цифры и изменить значения счётчиков. Вместе с этим будем находить общее количество цифр во вводимых числах, предварительно обнулив его. Сортировку счетчиков выполним «на месте», без перестановки элементов, только при выводе.

Решение на языке Паскаль:

```
var
  a: array[0..9] of word;
  j, n, i, num, count: integer;
begin
  readln(n);
  count := 0;
  for i:=0 to 9 do a[i]:=0;
  for i := 1 to n do
  begin
    readln(num);
    while num > 0 do
    begin
      a[num mod 10] := a[num mod 10] + 1;
      num := num div 10;
      count := count + 1;
    end
  end;
  for i := 1 to count do
    for j := 9 downto 0 do
      if (a[j] = i) then write(j, ' ');
end.
```

Решение на языке C++:

```
#include <iostream>
using namespace std;
int main()
{
    int a[10]= {0}, num, i, j, count=0;
    cin>>i;
    for (; i--;)
    {   cin>>num;
        while (num)
        {   a[num%10]++;
            num/=10;
            count++;
        }
    }
    for (i=0; i++<=count;)
      for (j=10; j--;)
       if (a[j]==i)  cout<<j<<' ';
}
```

71) (**Д. Муфаззалов, Уфа**) Количество появления каждой цифры найдем как в задаче 70, учитывая основание системы счисления. Всего таких различных цифр может быть не более 16. Симметричную строку можно составить лишь в том случае, если количество цифр, появившихся нечетное количество раз, не превышает 1. Проходить по всему

массиву при подсчете количества необязательно, можно выйти их цикла, когда оно станет равно 2.

Решение на языке Паскаль:

```
var
  count: array [0..15] of integer;
  c, n, num, check, i: integer;

begin
  check := 0;
  readln(n);
  for i := 0 to 15 do count[i] := 0;
  for i := 1 to n do
  begin
    readln(Num);
    while num > 0 do
    begin
      c := num mod 16;
      count[c] := count[c] + 1;
      num := num div 16;
    end
  end;
  i := 0;
  while (i <= 15) and (check < 2) do
  begin
    check := check + count[i] mod 2;
    i := i + 1;
  end;
  writeln(ord(check < 2));
end.
```

Решение на языке C++:

```
#include <iostream>
using namespace std;
int main()
{
    int count[16]= {0}, num, check=-2, i;
    cin >>i;
    for (; i--;)
    {   cin>>num;
        while(num)
        {   count[num%16]++;
            num/=16;
        }
    }
    for (i=16; i--&&(check+=count[i]%2););
    cout<<!!check;
    return 0;
}
```

72) (**Д. Муфаззалов, Уфа**) Потребуется массив счетчиков появлений различных наибольших делителей. Количество таких делителей не превосходит 1000 (от 1 до 1000). Самый компактный способ найти наибольший общий делитель (НОД) – использование рекурентной зависимости (модифицированного алгоритма Евклида)

НОД(a,b)=НОД(b, a % b), НОД(a, 0)=a,

где **a mod b** – остаток от деления **a** на **b**. Проходя по полученному массиву, находим максимальное значение, предварительно обнулив его. Проходя повторно, обязательно в порядке убывания номеров элементов, выводим те, которые равны максимальному.

Решение на языке Паскаль:

```
var
  count: array[1..1000] of word;
  c, maximum, n, i, num1, num2: integer;
begin
  for i := 1 to 1000 do count[i] := 0;
  maximum := 0;
  readln(n);
  for i := 1 to n do begin
    readln(num1, num2);
    while num2 > 0 do
    begin
      c := num2;
      num2 := num1 mod num2;
      num1 := c;
    end;
    count[num1] := count[num1] + 1;
  end;
  for i := 1 to 1000 do
    if (count[i] > maximum) then maximum := count[i];
  for i := 1000 downto 1 do
    if (count[i] = maximum) then write(i, ' ');
end.
```

Решение на языке C++:

```
#include <iostream>
using namespace std;
int nod(int a, int b)
{
    if (b) return nod(b, a%b);
    else return a;
}
int main()
{
    int i,num1,num2,count[1001]= {0}, maximum=0;
    cin>>i;
    for (; i--;) {
      cin>>num1>>num2;
      count[nod(num1, num2)]++;
    }
    for(i=1001; i--;)
      if (count[i]>maximum) maximum=count[i];
    for(i=1001; i--;)
      if (count[i]==maximum) cout<<i<<' ';
    return 0;
}
```

73) (**Д. Муфаззалов, Уфа**) Если число кратно 4, то чтобы оно перестало быть кратным, из него нужно вычесть число, некратное 4. Будем выбирать из каждой тройки наибольшее значение и найдем их сумму. Вместе с этим по всем тройкам будем искать наименьшую

некратную 4 разницу между этим наибольшим значением и каждым из оставшихся чисел в тройке (*). Если полученная сумма некратна 4, она и есть ответ. Если полученная сумма кратна 4, то если существует разница (*), вычтем ее из суммы, и это будет ответ, иначе выведем 0.

Решение на Паскале:

```
Var
  a: array[0..2] of word;
  i, mind, j, s, dif, n,m: word;

begin
  mind := 20004;
  s := 0;
  readln(n);
  for i := 1 to n  do
  begin
    for j := 0 to 2 do read(a[j]);
    m:=a[0];
    for j :=1 to 2 do
      if (a[j] > m) then m:=a[j];
    s := s + m;
    for j := 0 to 2 do
    begin
      dif := m - a[j];
      if (dif<mind) and (dif mod 4>0) then
        mind := dif;
    end;
  end;
  s := s - mind * ord(s mod 4 = 0);
  writeln(s * ord( s mod 4 <> 0));
end.
```

Решение на C++:

```
#include <iostream>
using namespace std;
int main()
{
    int i,a[3],mind=20004,j,s=0,dif,m;
    cin>>i;
    for (; i--;)
    {
        for(j=3; j--;) cin>>a[j];
        m=a[2];
        for(j=3; j--;) m=max(m,a[j]);
        s+=m;
        for (j=3; j--;)
        {
            dif=m-a[j];
            if (dif%4) mind=min(mind,dif);
        }
    }
    s-=mind*!(s%4);
    cout<<s*!!(s%4);
    return 0;
}
```

Решение на Matlab:

```
s=0;
mind=20004;
for i=1: input('')
   a=sscanf(input('','s'),'%i')';
   m=max(a);
   s=s+m;
   dif=m-a;
   mind=min([mind, dif+mind*~rem(dif,4)]);
end
s=s-mind*~rem(s,4);
disp(s*~~(rem(s,4)))
```

Решение на Python:

```
mind,s=20004,0
for i in range(int(input())):
      a= [int(x) for x in input().split()]
      m=max(a)
      s+=m
      for num in a:
          mind=min([m-num , mind]) if (m-num)% 4 else mind
s-=mind if not s%4 else 0
print(s if s%4 else 0)
```

Решение на Ruby:

```
mind,s=20004,0
(1..gets.to_i).each do
   a = gets.split.map { |n| n.to_i }
   m=a.max
   s+=m
   a.each do |num| dif=m-num
   mind=dif if (dif)%4>0 and dif<mind
   end
end
s-=mind if s%4==0
s=0 if s%4==0
print s
```

74) (**Е. Жеглов**) Будем искать максимально возможные суммы, которые при делении на 4 дают остатки 0, 1, 2 и 3. Для хранения значений таких сумм удобно использовать таблицу, состоящую из 4 строк с номерами от 0 до 3 и N столбцов (позже мы увидим, что достаточно и двух столбцов). В **i**-м столбце будем хранить максимально возможные суммы после считывания первых **i** строчек.

При считывании первой тройки чисел (8, 3, 4) у нас могут получиться три различные суммы: 8+3=11, 3+4=7, 4+8=12. Причем, 11 и 7 при делении на 4 дают остаток 3, но нам нужно максимальное значение, следовательно в строку №3 запишем значение 11. 12 кратно 4, поэтому в строку №0 запишем значение 12. Так как у нас не было сумм, которые при делении на 4 дают остатки 1 и 2, поэтому строки №1 и №2 остались равными 0. Если бы у нас была только одна тройка чисел, наш ответ находился бы в первом столбце в строке №0, т.е. 12.

Номер строки						
0	**12**	0	0	0	0	0

1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	**11**	0	0	0	0	0

При считывании второй тройки чисел (4, 8, 12) у нас могут получиться три различные суммы: 4+8=12, 8+12=20, 12+4=16. Давайте найдем какие мы можем получить максимально возможные суммы, после считывания второй тройки чисел.

На предыдущем шаге мы выяснили, что у нас с остатком 0 при делении на 4 сумма равна 12. Попытаемся к этому значению поочередно прибавить возможные суммы из второй тройки и проанализировать остатки от деления на 4 полученных сумм:

12 + 12 = 24 – остаток = 0, следовательно, во второй столбец в строку №0 запишем значение 24.

12 + 20 = 32 – эта сумма тоже дает остаток 0 и при этом больше чем 24, значит запишем во второй столбец в строку №0 новое значение – 32.

12 + 16 = 28 – снова получили остаток 0, но на этот раз 28 < 32, значит не будем заменять значение.

Номер строки						
0	12	~~24~~,**32**,~~28~~	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	11	0	0	0	0	0

На предыдущем шаге мы не смогли получить суммы с остатком 1 и 2 (т.е. мы не взяли из предыдущей тройки ни одной пары чисел, чтобы общая сумма была с такими остатками), следовательно, к ним мы не можем ничего прибавить и пропускаем эти строки.

С остатком 3 при делении на 4 у нас записана сумма, равная 11. Тоже попытаемся поочередно к ней прибавить наши суммы из второй тройки (12, 20 и 16), получим значения 23, 31 и 27 – все они тоже дают остаток 3 при делении на 4. 31 – максимальная из них.

Номер строки						
0	12	32	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	11	~~23~~,**31**,~~27~~	0	0	0	0

После считывания второй тройки чисел максимально возможная сумма кратная 4 это 32.

При считывании третьей тройки чисел (9, 5,6) у нас могут получиться три различные суммы: 9+5=14, 5+6=11, 6+9=15. Давайте найдем какие мы можем получить максимально возможные суммы, после считывания третьей тройки чисел.

На предыдущем шаге мы выяснили, что у нас с остатком 0 при делении на 4 сумма равна 32. Попытаемся к этому значению поочередно прибавить возможные суммы из третьей тройки и проанализировать остатки от деления на 4 полученных сумм:

32 + 14 = 46 – остаток = 2, следовательно, в третий столбец в строку №2 запишем значение 46.

32 + 11 = 43 – остаток = 3, значит, запишем в третий столбец в строку №3 значение 43.

32 + 15 = 47 – снова получили остаток 3, но 47 > 43, значит, новое значение в строке №3 – 47.

Номер строки						
0	12	32	0	0	0	0
1	0	0	0	0	0	0
2	0	0	**46**	0	0	0
3	11	31	~~43~~,**47**	0	0	0

Следующее ненулевое значение суммы, которое мы смогли получить на предыдущем шаге имеет остаток 3 при делении на 4. Тоже попытаемся поочередно к ней прибавить наши суммы из третьей тройки (14, 11 и 15), получим значения 45, 42 и 46.

31 + 14 = 45 – остаток = 1, следовательно, в третий столбец в строку №1 запишем значение 45.

31 + 11 = 42 – остаток = 2, но 42 < 46, значит, в строке №2 третьего столбца менять значение не будем.

31 + 15 = 46 – снова получили остаток2, опять наше значение не больше того, которое там уже записано.

Номер строки						
0	12	32	0	0	0	0
1	0	0	**45**	0	0	0
2	0	0	**46**,~~42~~,~~46~~	0	0	0
3	11	31	~~43~~,**47**	0	0	0

После считывания второй тройки чисел максимально возможная сумма кратная 4 это 32.

Аналогичные рассуждения приведут нас к следующим значениям:

При считывании четвертой тройки чисел (2, 8,3) у нас получатся суммы: 10, 11, 5. Будем поочередно прибавлять эти суммы к значениям 45, 46, 47 и анализировать остатки.

Номер строки						
0	12	32	0	**56**,~~56~~,~~52~~	0	0
1	0	0	45	**57**,~~57~~	0	0
2	0	0	46	~~50~~,**58**	0	0
3	11	31	47	**55**,~~51~~	0	0

При считывании пятой тройки чисел (12, 3, 5) у нас получатся суммы: 15, 8, 17. Будем поочередно прибавлять эти суммы к значениям 56, 57, 58, 55 и анализировать остатки.

Номер строки						
0	12	32	0	56	~~64~~,**72**,~~72~~	0
1	0	0	45	57	**73**,~~65~~,~~73~~	0
2	0	0	46	58	**74**,~~66~~,~~70~~	0
3	11	31	47	55	~~71~~,**75**,~~63~~	0

При считывании шестой тройки чисел (1, 4, 12) у нас получатся суммы: 5, 16, 13. Будем поочередно прибавлять эти суммы к значениям 72, 73, 74, 75 и анализировать остатки.

Номер строки						

0	12	32	0	56	72	88,~~80~~,~~88~~
1	0	0	45	57	73	,~~85~~,89
2	0	0	46	58	74	~~78~~,~~86~~,90
3	11	31	47	55	75	~~79~~,~~87~~,91

В итоге получили, что после считывания всех шести троек чисел, максимально возможная сумма кратная 4 равна 88 – ответ всегда находится в строке №0 столбца N.

Обратите внимание, что мы параллельно решили задачи, когда остаток от деления на 4 равен 1, 2 и 3 – ответы находятся в соответствующих строках последнего столбца.

Обратите внимание, что для решения задачи нет необходимости хранить всю таблицу. Для расчета i-го столбца нам нужны значения (i-1)-го столбца. Поэтому можно использовать таблицу размером 4x2 или две таблицы размером 4x1. Следовательно, алгоритм будет эффективен по памяти. Эффективность по времени подтверждается тем, что все вычисления выполняются параллельно со считыванием данных и количество выполняемых операций кратно N.

Решение на Паскале

```
var i, j, k, a, N: integer;
  x, y: array[0..3] of integer;
  inp: array[0..2] of integer;
begin
  readln(N);
  for i:=0 to 3 do begin
    x[i] := 0; y[i] := 0;
  end;
  readln(inp[0], inp[1], inp[2]);
  for i:=0 to 2 do begin
    a := inp[i] + inp[(i+1) mod 3];
                    //inp0+inp1; inp1+inp2; inp2+inp0
    if y[a mod 4] < a then    //записываем максимальное значение
      y[a mod 4] := a;   //индекс = остатку от деления на 4
  end;
  for j:=0 to N-2 do begin
    for i:=0 to 3 do begin
      x[i] := y[i]; //копируем значения второй таблицы в первую,
      y[i] := 0;    //а значения второй обнулим
    end;
    readln(inp[0], inp[1], inp[2]);
    for i:=0 to 3 do
      if x[i] <> 0 then  // для каждой суммы,
                         // найденной на предыдущем шаге
        for k:=0 to 2 do begin  //найдем все три варианта
                                //значений новой суммы
                                //(с учетом последних чисел)
          a := x[i]+inp[k]+inp[(k+1)mod 3];
          if y[a mod 4] < a then //и запишем максимум в ячейку
            y[a mod 4] := a;     //с индексом a%4
        end;
  end;
  writeln(y[0])
end.
```

Решение на Python

```
n = int(input())
x = [0]*4            # две таблицы размером 4x1
y = [0]*4
inp = list(map(int, input().split()))  #считываем первую тройку
for i in range(3):     # три варианта различных сумм
  a = inp[i] + inp[(i+1)%3]  #inp0+inp1; inp1+inp2; inp2+inp0
  if y[a % 4] < a:    #записываем максимальное значение
    y[a % 4] = a      #индекс = остатку от деления на 4
for j in range(n-1):
  for i in range(4):
    x[i] = y[i] #скопируем значения второй таблицы в первую,
    y[i] = 0    #а значения второй обнулим
  inp = list(map(int,input().split())) #считаем очередную тройку
  for i in range (4):
    if x[i]!=0:  #для каждой суммы, найденной на предыдущем шаге
      for k in range(3): #найдем все три варианта значений
                         #новой суммы (с учетом последних чисел)
        a = x[i]+inp[k]+inp[(k+1)%3]
        if y[a % 4] < a: #и запишем максимум в ячейку
          y[a % 4] = a   #с индексом a%4
print(y[0])
```

Другой вариант решения на Python (П. Енин):

```
s = 0 # сумма выбранных чисел
delta = [10000]*3 # список попарных разностей
for i in range(int(input())):
      # список из очередной тройки чисел
    num_list = [int(x) for x in input().split()]
    mi = min(num_list)
      # добавлям пару с наибольшей суммой
    s += sum(num_list) - mi
# проверка разностей всех парных комбинаций в тройке
# на остаток от деления на 4
    for num in num_list:
# минимальная разность между 2-я парами чисел, остаток от
# деления которой на 4 равен 1
        if (num - mi) % 4 == 1:
            delta[0] = min(num - mi, delta[0])
# минимальная разность между 2-я парами чисел, остаток от
# деления которой на 4 равен 2
        elif (num - mi) % 4 == 2:
            delta[1] = min(num - mi, delta[1])
# минимальная разность между 2-я парами чисел, остаток от
# деления которой на 4 равен 3
        elif (num - mi) % 4 == 3:
            delta[2] = min(num - mi, delta[2])
if s % 4: # если итоговая сумма не делится на 4
# вычитаем минимальную дельту с таким же остатком
# или обнуляем сумму
    s -= delta[s % 4 - 1] if delta[s % 4 - 1] < 10000 else s
print(s)
```

75) (**Д.В. Богданов**)

Неэффективное решение (задача А). В этом решении, которое оценивается на 2 балла, все данные сначала записываются в массив, а потом используется полный перебор вариантов в двойном цикле. Решение неэффективно по времени, потому что используется вложенный цикл и общее количество операций пропорционально N^2. Решение неэффективно по памяти, потому что используется массив, размер которого линейно зависит от N.

```
var N, i, j: integer;
    count: longint;
    a: array[0..9999] of integer;
begin
  readln(N);
  for i:=0 to N-1 do
    readln(a[i]);
  count := 0;
  for i:=0 to N-2 do
    for j:=i+1 to N-1 do
      if a[i]*a[j] mod 6 = 0 then
        count := count + 1;
  writeln(count)
end.)
```

Эффективное решение (задача Б). Произведение двух чисел кратно 6, если:

- один из сомножителей кратен 6 (второй может быть любым);
- ни один из сомножителей не кратен 6, но один из сомножителей кратен 2, а другой кратен 3.

Поэтому программа, вычисляющая количество пар элементов (a_i, a_j), произведение которых кратно 6 и $1 \le i \le j \le N$, может работать следующим образом.

Программа читает все входные данные один раз, не запоминая данные в массиве.

Программа для прочитанного фрагмента входной последовательности хранит значения трёх величин:

- M_2 — количество чисел, кратных 2, но не кратных 6;
- M_3 — количество чисел, кратных 3, но не кратных 6;
- M_6 — количество чисел, кратных 6.

Первый из элементов, кратных 6, может быть умножен на любой из других $N - 1$ элементов (все элементы последовательности, не включая его самого). Следующий элемент, кратный 6, может быть умножен только на любой из $N - 2$ элементов (так как пара с первым элементом, кратным 6, уже была учтена). Рассуждая аналогично, получим, что количество пар, одним из элементов которой является элемент, кратный 6, равно $(N-1) + (N-2) + \ldots + (N - M_6) = M_6 \cdot N - (1 + 2 + \ldots + M_6)$.

Сумма членов арифметической прогрессии $(1 + 2 + \ldots + M_6)$ равна $M_6 \cdot (M_6 + 1) / 2$.

Множество пар, в которых один из элементов кратен 2, а второй кратен 3, и при этом оба элемента не кратны 6, не пересекается с рассмотренным выше множеством пар, в которых один из элементов кратен 6. Количество таких пар равно $M_2 \cdot M_3$.

После того как все данные прочитаны, требуемый в условии ответ вычисляется по формуле $M_6 \cdot N - M_6 \cdot (M_6 + 1) / 2 + M_2 \cdot M_3$.

Решение на Pascal:

```
var N, x, i, j, m2, m3, m6: integer;
begin
  m2 := 0; m3 := 0; m6 := 0;
  readln(N);
  for i := 1 to N do begin
    readln(x);
```

```
    if x mod 6 = 0 then
      m6 := m6 + 1
    else begin
      if x mod 2 = 0 then
        m2 := m2 + 1;
      if x mod 3 = 0 then
        m3 := m3 + 1
    end
  end;
  write(m6 * N - m6 * (m6 + 1) div 2 + m2 * m3)
end.
```

(**И. Самойлов**) Формулу для вычисления результата можно интерпретировать немного по-другому. Пусть у нас есть **m6** чисел, которые делятся на 6. Первое из них образует пару с о всеми, кроме самого себя (**N-1** пар) , второе – со всеми, кроме самого себя и уже учтённого первого числа (**N-2** пар), последнее – только с **N-m6** числами, которые не делятся на 6. Получаем сумму арифметической прогрессии, которую можно «свернуть» по формуле:

$$(N-1)+(N-2)+...+(N-m_6)=\left[\frac{(N-1)+(N-m_6)}{2}\right]m_6=\frac{2N-m_6-1}{2}m_6$$

Альтернативный вариант решения предложил **Е. Жеглов**.

Решение на Паскале:

```
Var i,n,k,k2,k3,k6: longint;
begin
  read(k);
  n:=0;     // - ответ
  k2:=0;    // - количество чисел кратных 2, но не кратных 6
  k3:=0;    // - количество чисел кратных 3, но не кратных 6
  k6:=0;    // - количество чисел кратных 6
  for i:=0 to k-1 do begin
    read(a);          // - очередное число
    if a mod 6 = 0    // если число кратно 6, то оно
    then begin        // составляет пару
        n:=n+i;       // со всеми введенными ранее числами
        k6:=k6+1
    end
    else if a mod 3 = 0 // иначе, если число кратно 3, то
    then begin          // оно составляет пару
        n:=n+k2+k6;     // со всеми ранее введенными числами
        k3:=k3+1        // кратными 2
    end
    else if a mod 2 = 0 // иначе, если число кратно 2, то
    then begin          // оно составляет пару
        n:=n+k3+k6;     // со всеми ранее введенными числами
        k2:=k2+1        // кратными 3
    end
    else                // иначе (если не кратны 2, 3, 6)
        n:=n+k6;        // составляет пару с числами кратными 6
  end;
  writeln(n);
end.
```

Решение на Python:

```
k = int(input())
```

```
n = 0      # - ответ
k2 = 0     # - количество чисел кратных 2, но не кратных 6
k3 = 0     # - количество чисел кратных 3, но не кратных 6
k6 = 0                  # - количество чисел кратных 6
for i in range(k):      # переберём все числа
    a = int(input())    # - очередное число
    if a % 6 == 0:      # если число кратно 6, то
        n += i          # оно составляет пару
        k6 += 1         # со всеми введенными ранее числами
    elif a % 3 == 0:    # иначе, если число кратно 3, то
        n = n + k2 + k6 # оно составляет пару
        k3 += 1         # со всеми ранее введенными числами
                        # кратными 2
    elif a % 2 == 0:    # иначе, если число кратно 2, то
        n = n + k3 + k6 # оно составляет пару
        k2 += 1         # со всеми ранее введенными числами
                        # кратными 3
    else:               # иначе (если не кратны 2, 3, 6)
        n += k6         # такое число составляет пару
                        # с числами кратными 6
print(n)
```

(Д. Ф. Муфаззалов) Попытаемся сделать это решение более компактным.

По мере поступления чисел будем накапливать количество таких, которые кратны 1, кратны 2, кратны 3 и кратны 6. Эти количества будем хранить в массиве, а накапливать в цикле.

Очередное число образует пары с предшествующими числами, при этом возможны следующие случаи:

- число кратно 6, образует пары со всеми числами;
- кратно 2, но не 3, образует пары с числами, кратными 3;
- кратно 3, но не 2, образует пары с числами, кратными 2;
- не кратно 2 и не кратно 3, образует пары с числами, кратными 6.

Определение, к какому случаю относится число, будем производиь в цикле.

Решение на Python:

```
c= [0]*4
answ,j = 0,3
d = [1,2,3,6]
for i in range(int(input())):
  x = int(input())
  while x % d[j]: j-=1
  answ += c[3-j]
  for j in range(4):
    c[j] += not(x%d[j]) # накопление количеств
print (answ)
```

Решение компактно и на C++:

```
#include <iostream>
using namespace std;
int main()
{   int N,c[4]= {0},x,answ=0,j=4,d[4]= {1,2,3,6};
    cin>>N;
    while(N--)
    {
      cin>>x;
```

```
      while(x%d[--j]);
      answ+=c[3-j];
      for (j=0; j<4; j++) c[j]+=!(x%d[j]);//накопление количеств
    }
    cout<<answ;
    return 0;
}
```

76) Все входные числа не превышают 10^9, поэтому из длина может быть от 1 до 10 включительно. Заводим 10 счётчиков и обнуляем их:

```
var count: array[1..10] of integer;
...
for i:=1 to 10 do count[i] := 0;
```

Простейший способ определить длину слова – читать данные как символные строки и использовать стандартную функцию Length для вычисления их длин:

```
for i:=1 to N do begin
  readln(s);
  len := Length(s);
  count[len] := count[len] + 1;
end;
```

После этого нужно найти минимальное значение в массиве count и запомнить номер этого минимального значения.

```
minCount := N+1;
  for i:=1 to 10 do
    if (count[i] > 0) and (count[i] < minCount) then begin
      minCount := count[i];
      minLen := i;
    end;
```

Обратите внимание, что

1) начальное значение переменной **minCount** принято равным **N+1**, это больше, чем любое возможное значение счётчика (не может быть больше **N** чисел какой-то длины);
2) в условии использовано строгое неравенство **count[i] < minCount** для того, чтобы была найдена именно минимальная из возможных длин, соответствущих одному и тому же значению счётчиков.

Остаётся вывести значения переменных **minLen** и **minCount**. Вот полная программа:

```
var count: array[1..10] of integer;
    i, N, len, minLen, minCount: integer;
    s: string;
begin
  for i:=1 to 10 do count[i] := 0;
  readln(N);
  for i:=1 to N do begin
    readln(s);
    len := Length(s);
    count[len] := count[len] + 1;
  end;
  minCount := N+1;
  for i:=1 to 10 do
    if (count[i] > 0) and (count[i] < minCount) then begin
      minCount := count[i];
      minLen := i;
    end;
```

```
  writeln(minLen, ' ', minCount);
end.
```

77) (**Д.В. Богданов**)

Задача А. Решение, не эффективное ни по времени, ни по памяти. Все данные записываются в массив, затем используется перебор всех пар элементов массива в двойном цикле.

```
var N, i, j : integer;
    count: longint;
   a: array[1..10000] of integer;
begin
  readln(N);
  for i:=1 to N do
    readln(a[i]);
  count:= 0;
  for i:=1 to N-1 do
    for j:=i+1 to N do
      if (a[i]+a[j]) mod 12 = 0 then
        count := count + 1;
  writeln(count)
end.
```

Задача Б. Идея решения основана на том, что если сумма двух чисел делится на 12, то сумма остатков от деления этих чисел на 12 также делится на 12, то есть равна 0 либо 12 (сумма двух цифр не может быт больше, чем 18). Поэтому заводим массив счётчиков чисел с одинаковыми остатками от деления на 12:

```
  var rem: array[0..11] of integer;
  ...
  for i:= 0 to 11 do
    rem[i]:= 0;
```

И при вводе очередного числа увеличиваем счётчик, соответствующий полученному остатку от деления на 12:

```
  for i := 1 to N do begin
    readln(x);
    inc( rem[x mod 12] ) {или rem[x mod 12]=rem[x mod 12]+1 }
  end;
```

Как теперь подсчитать общее количество пар? Если число делится на 12 (остаток 0), то оно образует пару со всеми числами, которые делятся на 12, кроме самого себя, таких сочетаний **rem[0]*(rem[0]-1)**, но при этом каждая пара подсчитана дважды, то есть это произведение нужно разделить на 2. Аналогичная ситуация с теми числами, которые делятся на 6, количество соответствующих пар находим как

rem[6]*(rem[6]-1) div 2.

Для остальных подходящих пар остатки для двух чисел, образующих пару, не равны, так что каждое число из первой группы (например, с остатком 1) сочетается с каждым числом из второй группы (с остатком 12-1 = 11). Чтобы не получить дублирование пар, будем в цикле перебирать только остатки, меньшие 6. Получается так:

```
  count := (rem[0]*(rem[0]-1) + rem[6]*(rem[6]-1)) div 2;
  for i := 1 to 5 do
    count := count + rem[i]*rem[12-i];
```

Приведём полную программу:

```
var rem: array[0..11] of integer;
    N, i, x: integer;
    count: longint;
```

```
begin
  for i:=0 to 11 do rem[i]:= 0;
  readln(N);
  for i:= 1 to N do begin
    readln(x);
    inc(rem[x mod 12])
  end;
  count:= (rem[0]*(rem[0]-1) +
           rem[6]*(rem[6]-1)) div 2;
  for i:= 1 to 5 do
    count:= count + rem[i]*rem[12-i];
  writeln(count)
end.
```

78) (**Д. Ф. Муфаззалов**)

Задача А. Решение, не эффективное ни по времени, ни по памяти. Все данные записываются в массив, затем используется перебор всех допустимых пар элементов массива в двойном цикле.

Решение на Python:

```
a=[0]*10000
answ=0
n=int(input())
for i in range(n):
    a[i]=int(input())
for i in range(n):
    for j in range(i+3,n):
        answ+=not((a[i]*a[j])%6)
print(answ)
```

Задача Б. Будем помещать поступающие числа в очередь длиной 3 (в решении на C++ использована замкнутая в кольцо очередь). По мере выхода чисел из очереди будем накапливать количество чисел, кратных 1, кратных 2, кратных 3 и кратных 6. Эти количества будем хранить в массиве. Число, поступающее в очередь, образует пары с числами, которые вышли из нее, при этом возможны следующие случаи:

- число кратно 6, образует пары со всеми числами;
- кратно 2, но не 3, образует пары с числами, кратными 3;
- кратно 3, но не 2, образует пары с числами, кратными 2;
- не кратно 2 и не кратно 3, образует пары с числами, кратными 6.

Решение на C++:

```
#include <iostream>
using namespace std;
int main()
{
    int N,i=0,a[3],c[4]= {0},x,answ=0,j,d[4]= {1,2,3,6};
    cin>>N;
    while(i<3) cin>>a[i++];
    while(i<N)
    {
        for (j=0; j<4; j++) c[j]+=!(a[i%3]%d[j]);
        cin>>x;
        while(x%d[--j]);//определение типа введенного числа
        answ+=c[3-j];//добавление количества пар с этим числом в
                     //результат
```

```
        a[i++%3]=x;// обработка очереди
    }
    cout<<answ;
    return 0;
}
```

Решение на Python:

```
from queue import Queue
c=[0]*4
a=Queue()
answ=0
d= [1,2,3,6]
N=int(input())
for i in range(3): a.put(int(input()))
for i in range(3,N):
        y=a.get()       #удаление из очереди
        x=int(input())
        a.put(x)        #добавление в очередь
        for j in range(4): c[j]+=not(y%d[j]) #подсчет количеств
        while(x%d[j]): j-=1 # опредедение типа числа
        answ+=c[3-j];   #добавление количества пар в результат
print(answ)
```

79) (**Д. В. Богданов**)

Задача А. Решение, не эффективное ни по времени, ни по памяти. Все данные записываются в массив, затем используется перебор всех допустимых троек элементов массива в тройном вложенном цикле. Мы ищем тройки *различных* элементов массива, поэтому в них номер первого элемента i изменяется от 1 до N-2, номер второго j – от i до N-1, а номер третьего, k, от j+1 до N.

Решение на Паскале:

```
var N, i, j, k: integer;
    count: longint;
   a: array[1..10000] of integer;
begin
  readln(N);
  for i:=1 to N do
    readln(a[i]);
  count:= 0;
  for i:=1 to N-2 do
    for j:=i+1 to N-1 do
      for k:=j+1 to N do
        if (a[i]+a[j]+a[k]) mod 12 = 0 then
          count := count + 1;
  writeln(count)
end.
```

Решение на Python:

```
N = int(input())
A = [0]*N
for i in range(N):
  A[i] = int(input())
count = 0
for i in range(N-2):
  for j in range(i+1,N-1):
    for k in range(j+1,N):
```

```
      if A[i] + A[j] + A[k]) % 12 == 0:
        count += 1
print(count)
```

Задача Б. Идея решения основана на том, что если сумма трёх чисел делится на 12, то сумма остатков от деления этих чисел на 12 также делится на 12, то есть равна одному из чисел: 0, 12, 24 (сумма трёх цифр не может быть больше, чем 27).

Поэтому заводим массив счётчиков чисел с одинаковыми остатками от деления на 12:

```
var rem: array[0..11] of integer;
...
for i:= 0 to 11 do
  rem[i]:= 0;
```

И при вводе очередного числа увеличиваем счётчик, соответствующий полученному остатку от деления на 12:

```
for i := 1 to N do begin
  readln(x);
  inc( rem[x mod 12] ) {или rem[x mod 12]=rem[x mod 12]+1 }
end;
```

Как теперь подсчитать общее количество пар?

Сумма трёх чисел делится на 12, если сумма остатков (здесь и далее везде имеются в виду остатки от деления на 12) делится на 12, то есть равна 0, 12 или 24.

Рассмотрим все числа, которые делятся на 12, их количество равно **rem[0]**. Они могут образовывать подходящие тройки сами с собой. Первое число в тройке мы можем выбрать **rem[0]** способами. Второе должно быть другое, поэтому есть **rem[0]-1** способов его выбора. Когда выбраны два числа из трёх, третье выбирается **rem[0]-2** способами. Таким образом, количество таких троек равно

rem[0]*(rem[0]-1)*(rem[0]-2)

Но так мы подсчитали каждую тройку 6 раз: abc, acb, bac, bca, cab, cba. Чтобы получить количество разных троек, нужно разделить полученное произведение на 6:

rem[0]*(rem[0]-1)*(rem[0]-2) div 6

Такая же ситуация будет всегда, когда мы связываем числа с одинаковыми остатками, например, с остатками, равными 4 (4 + 4 + 4 = 12).

Теперь пусть первое число в тройке делится на 12, а два остальных дают в остатке 6. Тогда первое число выбирается **rem[0]** способами. Количество пар чисел с остатком 6 равно **rem[6]*(rem[6]-1)**, но каждая из пар учитывается в этом произведении дважды, поэтому количество троек с остатками 0-6-6 равно

rem[0]*rem[6]*(rem[6]-1) div 2

Следующий вариант – остатки для всех чисел в тройке разные, например, 0, 5 и 7. При этом есть rem[0] способов выбрать первое число, rem[5] способов выбрать второе и rem[7] способов выбора третьего. Общее количество вариантов:

rem[0]*rem[5]*rem[7]

Наконец, рассмотрим задачу в общем случае. Объединим в тройку первое число с остатком i, второе – с остатком $j \geq i$, и третье – с остатком k. Все остатки должны дать в сумме 0, 12 или 24. Поэтому при выбранных i и j можно вычислить k по одной из трёх форму:

$k = 0 - i - j$

$k = 12 - i - j$

$k = 24 - i - j$

причём значение k должно получиться от 0 до 11. Эти формулы можно объединит в одну

$k = (24 - i - j) \bmod 12$

В двойном цикле перебираем все возможные остатки, соблюдая условие $j \geq i$, для каждой пары i-j вычисляем соответствующее k:

```
   count := 0;
   for i:=0 to 11 do
       for j:=i to 11 do begin
         k := (24 - i - j) mod 12;
         if k >= j then
           { увеличить count }
```

Обратите внимание, что для исключения повторяющихся троек мы рассматриваем только варианты, когда $i \le j \le k$. Увеличение счётчика count выполняется в соответствии с алгоритмом, который мы рассмотрели выше – в зависимости от того, сколько одинаковых чисел среди значений i, j, k.

Приведём полную программу:

```
var N, x, i, j, k: integer;
    count: longint;
    rem: array[0..11] of integer;
begin
  for i:= 0 to 11 do
    rem[i]:= 0;
  readln(N);
  for i := 1 to N do begin
    readln(x);
    inc( rem[x mod 12] )
  end;
  count:= 0;
  for i:=0 to 11 do
    for j:=i to 11 do begin
      k := (24 - i - j) mod 12;
      if k >= j then
        if (i = j) and (j = k) then
          count := count + rem[i]*(rem[i]-1)*(rem[i]-2) div 6
        else if i = j then
          count := count + rem[i]*(rem[i]-1)*rem[k] div 2
     {  else if i = k then
          count := count + rem[i]*rem[j]*(rem[i]-1) div 2 }
        else if j = k then
          count := count + rem[i]*rem[j]*(rem[j]-1) div 2
        else
          count := count + rem[i]*rem[j]*rem[k];
    end;
  writeln(count)
end.
```

Условный оператор **if i = k**, взятый в комментарий, тут, вообще говоря, лишний. Поскольку при переборе поддерживается выполнение условия $i \le j \le k$, условие $i = k$ может выполниться только при $i = j = k$, а этот случай рассмотрен отдельно.

Решение на Python:

```
N = int(input())

rem = [0]*12
for i in range(N):
  x = int(input())
  rem[x % 12] += 1

count = 0
```

```
for i in range(12):
  for j in range(i,12):
    k = (24 - i - j) % 12
    if k >= j:
      if i == j == k:
        count += rem[i]*(rem[i]-1)*(rem[i]-2)//6
      elif i == j:
        count += rem[i]*(rem[i]-1)*rem[k]//2
    # elif i == k:
    #   count += rem[i]*rem[j]*(rem[i]-1)//2
      elif j == k:
        count += rem[i]*rem[j]*(rem[j]-1)//2
      else:
        count += rem[i]*rem[j]*rem[k]
print(count)
```

Решение на Python (Г.М. Федченко):

Вместо вычисления k можно перебрать в цикле все возможные значения $k \geq j$ и увеличивать счётчик, когда сумма $(i + j + k)$ делится на 12:

```
N = int(input())

rem = [0]*12
for i in range(N):
  x = int(input())
  rem[x % 12] += 1

count = 0
for i in range(12):
  for j in range(i,12):
    for k in range(j,12):
      if (i+j+k)%12 == 0:
        if i == j == k:
          count += rem[i]*(rem[i]-1)*(rem[i]-2)//6
        elif i == j:
          count += rem[i]*(rem[i]-1)*rem[k]//2
     #  elif i == k:
     #    count += rem[i]*rem[j]*(rem[i]-1)//2
        elif j == k:
          count += rem[i]*rem[j]*(rem[j]-1)//2
        else:
          count += rem[i]*rem[j]*rem[k]
print(count)
```

Для решения этой задачи также можно использовать красивый и эффективный метод, который предложил Алексей Пискунов для задачи 86 (см. пример и объяснение там).

80) (**А. Жуков**)

Задача А. Полный перебор

Перебираем все пары таким образом, чтобы не учитывать одни и те же пары дважды. Например, для массива из N чисел будем рассматривать пары в следующем порядке:

(a[1],a[2]), (a[1],a[3]), (a[1],a[4]) … (a[1],a[N])

(a[2],a[3]), (a[2],a[4]) … (a[2],a[N])

пара (a[2],a[1]) не рассматривается, т.к. пара (a[1],a[2]) уже рассмотрена

…

(a[N-1], a[N])

Для организации такого перебора потребуется вложенный цикл.

Решение на **Паскале**:

```
const MAXN = 1000;
var N, i, j, k: integer;
    a: array[1..MAXN] of integer;
begin
  readln(N);
  for i:=1 to N do
    readln(a[i]);
  k := 0;
  for i:=1 to N-1 do
    for j:=i+1 to N do
      if (a[i] * a[j] mod 13 = 0) and
         ((a[i] + a[j]) mod 2 <> 0) then
        k := k + 1;
  writeln(k)
end.
```

Решение на **C++**:

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  int n,i,j,k;
  cin >> n;
  vector<int> a(n);
  for(i=0;i<n;++i)
    cin >> a[i];
  k = 0;
  for(i=0;i<n-1;++i)
    for(j=i+1;j<n;++j)
      if (a[i]*a[j] % 13 == 0 && (a[i] + a[j]) % 2 != 0)
        ++k;
  cout << k;
}
```

Решение на **Python**:

```
N = int(input())
a = [int(input()) for i in range(N)]
k = 0
for i in range(N-1):
  for j in range(i+1,N):
    if a[i]*a[j] % 13 == 0 and
      (a[i] + a[j]) % 2 != 0:
      k += 1
print(k)
```

Задача B. Счетчики чисел и формула

Эффективное решение не подразумевает сохранение чисел в массив, т.е. число считывается, каким-то образом обрабатывается и больше не участвует в алгоритме.

Для алгоритма важны следующие факты:

- произведение 2-х чисел делится на 13 тогда и только тогда, когда хотя бы одно из чисел делится на 13

- сумма 2-х чисел нечетная тогда и только тогда, когда одно из чисел четное, а другое нечетно

Значит мы можем составить пары из следующих чисел:

- четные кратные 13 и нечетные кратные 13
- четные кратные 13 и нечетные некратные 13
- нечетные кратные 13 и четные некратные 13

Обозначим:

k1 – нечетные некратные 13
k2 – четные некратные 13
k13 – нечетные кратные 13
k26 – четные кратные 13

Тогда, общее число пар = k26 * k13 + k26 * k1 + k13 * k2

Эту формулу можно упростить, выбросив один из счетчиков, например, следующими способами:

1. k26 * (k13 + k1) + k13 * (N – k1 – k13 – k26)
2. k26 * (N – k2 – k26) + k13 * k2, т.к. (k13 + k1) = (N – k2 – k26) – это все нечетные числа

Решение на **Паскале**:

```
var N, i, x: integer;
    k1, k2, k13, k26: integer;
begin
  readln(N);
  k1 := 0;
  k2 := 0;
  k13 := 0;
  k26 := 0;
  for i:=1 to N do begin
    readln(x);
    if x mod 13 = 0 then
      if x mod 2 = 0 then
        k26 += 1
      else
        k13 += 1
    else
      if x mod 2 = 0 then
        k2 += 1
      else
        k1 += 1
  end;
  writeln(k26 * k13 + k26 * k1 + k13 * k2);
end.
```

Решение на **C++**:

```
#include <iostream>
using namespace std;
int main() {
  int n,i,x,k26,k13,k1;
  cin >> n;
  k26 = k13 = k1 = 0;
  for(i=0; i<n; ++i) {
    cin >> x;
```

```
    if (x % 2 == 0) {
      if (x % 13 == 0)
        ++k26;
    } else {
      if (x % 13 == 0)
        ++k13;
      else
        ++k1;
    }
  }
  cout << k26 * (k13 + k1) + k13 * (n - k1 - k13 - k26);
}
```

Решение на **Python**:

```
n = int(input())
k26 = k13 = k2 = 0
for i in range(n):
  x = int(input())
  if x % 2 == 0:
    if x % 13 == 0:
      k26 += 1
    else:
      k2 += 1
  elif x % 13 == 0:
    k13 += 1
print(k26 * (n - k2 - k26) + k13 * k2)
```

Задача B. Счетчики числа пар

Идея ещё одного эффективного способа решения задачи заключается в подсчете количества пар при получении каждого числа. Для этого нужно рассмотреть задачу для некоторого элемента с номером **i**. Допустим к этому моменту мы вычислили среди чисел, которые уже были прочитаны следующие значения:

- **k1** - число **всех нечетных** чисел с номерами меньше **i**
- **k2** – число **всех четных** чисел с номерами меньше **i**
- **k13** – число нечетных чисел кратных 13 с номерами меньше **i**
- **k26** – число четных чисел кратных 13 с номерами меньше **i**

Тогда для текущего элемента (с номером **i**) можно определить какое количество пар с уже просмотренными элементами (с номерами меньше **i**) можно составить:

```
   если элемент кратен 3-м то
       если элемент четный то k1
       иначе k2
   иначе
       если элемент четный то k13
       иначе k26
```

Это число пар необходимо накапливать в переменной для подсчета количества пар, не забывая изменять счетчики. *Этот способ решения пригодится нам в следующих задачах*.

Решение на **Паскале**:

```
var N, i, x: integer;
    k, k1, k2, k13, k26: integer;
begin
  readln(N);
  k := 0;
  k1 := 0; // всего нечетных
```

```
  k2 := 0; // всего четных
  k13 := 0; // кратных 13, но не кратных 2
  k26 := 0; // кратных 26
  for i:=1 to N do begin
    readln(x);
    if x mod 13 = 0 then
      if x mod 2 = 0 then begin
        k += k1;
        k26 += 1;
        k2 += 1;
      end else begin
        k += k2;
        k1 += 1;
        k13 += 1;
      end
    else
      if x mod 2 = 0 then begin
        k += k13;
        k2 += 1;
      end else begin
        k += k26;
        k1 += 1
      end
  end;
  writeln(k);
end.
```

Решение на **C++**:

```
#include <iostream>
using namespace std;
int main() {
  int n,i,x,k26,k13,k2,k1,k;
  cin >> n;
  k = k26 = k13 = k2 = k1 = 0;
  for(i=0; i<n; ++i) {
    cin >> x;
    if (x % 13 == 0) {
      if (x % 2 == 0) {
        k += k1;
        ++k26;
        ++k2;
      } else {
        k += k2;
        ++k1;
        ++k13;
      }
    } else {
      if (x % 2 == 0) {
        k += k13;
        ++k2;
      } else {
        k += k26;
        ++k1;
      }
```

```
    }
  }
  cout << k;
}
```

Решение на **Python**:

```
n = int(input())
k = k26 = k13 = k2 = k1 = 0
for i in range(n):
  x = int(input())
  if x % 13 == 0:
    if x % 2 == 0:
      k, k26, k2 = k+k1, k26+1, k2+1
    else:
      k, k13, k1 = k+k2, k13+1, k1+1
  elif x % 2 == 0:
    k, k2 = k+k13, k2+1
  else:
    k, k1 = k+k26, k1+1
print(k)
```

81) (**А. Жуков**)

Задача А. Полный перебор

Перебираем все пары таким образом, чтобы не учитывать одни и те же пары дважды и при этом (**j** – **i**) >= 5. Например, для массива из N чисел будем рассматривать пары в следующем порядке:

(a[1],a[6]), (a[1],a[7]), (a[1],a[8]) … (a[1],a[N])

(a[2],a[7]), (a[2],a[8]) … (a[1],a[N])

…

(a[N-5], a[N])

Для организации такого перебора потребуется вложенный цикл.

Решение на **Паскале**:

```
const MAXN = 1000;
var N, i, j, k: integer;
    a: array[1..MAXN] of integer;
begin
  readln(N);
  for i:=1 to N do
    readln(a[i]);
  k := 0;
  for i:=1 to N-5 do
    for j:=i+5 to N do
      if (a[i] * a[j] mod 13 = 0) and ((a[i] + a[j]) mod 2 <> 0)
then
        k := k + 1;
  writeln(k)
end.
```

Решение на **C++**:

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  int n,i,j,k;
```

```
  cin >> n;
  vector<int> a(n);
  for(i=0;i<n;++i)
    cin >> a[i];
  k = 0;
  for(i=0;i<n-5;++i)
    for(j=i+5;j<n;++j)
      if (a[i]*a[j] % 13 == 0 && (a[i] + a[j]) % 2 != 0)
        ++k;
  cout << k;
}
```

Решение на **Python**:

```
N = int(input())
a = [int(input()) for i in range(N)]
k = 0
for i in range(N-5):
  for j in range(i+5,N):
    if a[i]*a[j]%13 == 0 and (a[i] + a[j]) % 2 != 0:
      k += 1
print(k)
```

Задача B. Счетчики числа пар и очередь

Идея эффективного решения задачи заключается в подсчете количества пар при получении каждого очередного числа, за исключением последних 4-х элементов (их мы сохраняем в очереди).

Для этого нужно рассмотреть задачу для некоторого элемента с номером i. Допустим к этому моменту мы вычислили среди чисел, которые уже были прочитаны следующие значения:

- k1 - число **всех нечетных** чисел с номерами меньше i-4
- k2 – число **всех четных** чисел с номерами меньше i-4
- k13 – число нечетных чисел кратных 13 с номерами меньше i-4
- k26 – число четных чисел кратных 13 с номерами меньше i-4

Тогда для текущего элемента (с номером i) можно определить какое количество пар с уже просмотренными элементами (с номерами меньше i) можно составить:

```
   если элемент кратен 3-м то
       если элемент четный то k1
       иначе k2
   иначе
       если элемент четный то k13
       иначе k26
```

Это число пар необходимо накапливать в переменной для подсчета количества пар. Основная проблема в этом подходе в том, что изменение счетчиков нужно производить не сразу, как только пришел очередной элемент, а только, когда элемент отстоит от текущего на 5 – для этого используется очередь (см. задачи 54, 55, 56, 78).

Решение на **Паскале**:

```
const
  D = 5;// минимальное расстояние между элементами
var
  queue: array[0..D] of integer;
  N, i, x: integer;
  k, k1, k2, k13, k26: integer;
```

```
begin
  readln(N);
  for i := 1 to D do
    readln(queue[i mod D]); // считываем первые D элементов в
очередь
  k := 0;  // подходящих пар
  k1 := 0; // всего нечетных
  k2 := 0; // всего четных
  k13 := 0; // кратных 13, но не кратных 2
  k26 := 0; // кратных 26
  for i := D + 1 to N do
  begin
    // очередной элемент (queue[i mod D]) выходит
    // из очереди - модифицируем счетчики
    if queue[i mod D] mod 13 = 0 then
      if queue[i mod D] mod 2 = 0 then begin
        k26 += 1;
        k2 += 1;
      end else begin
        k1 += 1;
        k13 += 1;
      end
    else
    if x mod 2 = 0 then
      k2 += 1
    else
      k1 += 1;
    // считываем очередной элемент и подсчитываем
    // количество пар
    readln(x);
    if x mod 13 = 0 then
      if x mod 2 = 0 then
        k += k1
      else
        k += k2
    else
      if x mod 2 = 0 then
        k += k13
      else
        k += k26;
    queue[i mod D] := x; // записываем элемент в очередь
  end;
  writeln(k);
end.
```

Решение на **C++**:

```
#include <iostream>
using namespace std;
int main() {
  const int D = 5;
  int n,i,x,k26,k13,k2,k1,k;
  int queue[D] = {0};
  cin >> n;
  for (i=0;i<D;++i)
```

```
    cin >> queue[i];
  k = k26 = k13 = k2 = k1 = 0;
  for(i=D; i<n; ++i) {
    if (queue[i % D] % 13 == 0) {
      if (queue[i % D] % 2 == 0)
        ++k26, ++k2;
      else
        ++k1, ++k13;
    } else {
      if (queue[i % D] % 2 == 0)
        ++k2;
      else
        ++k1;
    }
    cin >> x;
    if (x % 13 == 0) {
      if (x % 2 == 0)
        k += k1;
      else
        k += k2;
    } else {
      if (x % 2 == 0)
        k += k13;
      else
        k += k26;
    }
    queue[i % D] = x;
  }
  cout << k;
}
```

Решение на **Python**:

```
D = 5
n = int(input())
queue = [int(input()) for i in range(D)]
k = k26 = k13 = k2 = k1 = 0
for i in range(D,n):
  # элемент из очереди
  if queue[i % D] % 13 == 0:
    if queue[i % D] % 2 == 0:
      k26, k2 = k26+1, k2+1
    else:
      k13, k1 = k13+1, k1+1
  elif queue[i % D] % 2 == 0:
    k2 += 1
  else:
    k1 += 1
  # вводим элемент
  x = int(input())
  if x % 13 == 0:
    if x % 2 == 0:
      k += k1
    else:
      k += k2
```

```
  elif x % 2 == 0:
    k += k13
  else:
    k += k26
  queue[i % D] = x
print(k)
```

82) (**А. Жуков**)

Задача А. Полный перебор

Перебираем все пары таким образом, чтобы не учитывать одни и те же пары дважды и при этом (j – i) <= 5. Например, для массива из N чисел будем рассматривать пары в следующем порядке:

(a[1],a[2]), (a[1],a[3]), (a[1],a[4]), (a[1],a[5])
(a[2],a[3]), (a[2],a[4]), (a[2],a[5]), (a[2],a[6])
…
(a[N-5],a[N-4]), (a[N-5],a[N-3]), (a[N-5],a[N-2]), (a[N-5],a[N-1])
(a[N-4],a[N-3]), (a[N-4],a[N-2]), (a[N-4],a[N-1]), (a[N-4],a[N])
(a[N-3],a[N-2]), (a[N-3],a[N-1]), (a[N-3],a[N])
(a[N-2],a[N-1]), (a[N-2],a[N])
(a[N-1],a[N])

Для организации такого перебора потребуется вложенный цикл. Так как во вложенном цикле может быть разное количество повторений, то верхняя граница (на Паскале) выбирается как минимальное из двух чисел (i+4, N) – чтобы не выйти за границы массива.

Решение на **Паскале**:

```
const MAXN = 1000;
var N, i, j, k: integer;
    a: array[1..MAXN] of integer;
begin
  readln(N);
  for i:=1 to N do
    readln(a[i]);
  k := 0;
  for i:=1 to N-1 do
    for j:=i+1 to min(i+4,N) do
      if (a[i] * a[j] mod 13 = 0) and
         ((a[i] + a[j]) mod 2 <> 0) then
        k := k + 1;
  writeln(k)
end.
```

Решение на **C++**:

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  int n,i,j,k;
  cin >> n;
  vector<int> a(n);
  for(i=0;i<n;++i)
    cin >> a[i];
  k = 0;
  for(i=0;i<n-1;++i)
    for(j=i+1;j<min(i+5,N);++j)
```

```
      if (a[i]*a[j] % 13 == 0 && (a[i] + a[j]) % 2 != 0)
        ++k;
  cout << k;
}
```

Решение на **Python**:

```
N = int(input())
a = [int(input()) for i in range(N)]
k = 0
for i in range(N-1):
  for j in range(i+1, min(i+5,N)):
    if a[i]*a[j]%13 == 0 and (a[i] + a[j]) % 2 != 0:
      k += 1
print(k)
```

Задача B. Счетчики числа пар и очередь

Здесь можно применить достаточно простой подход – считаем первые 5 чисел и для них вычисляем все интересующие нас счетчики:

- k1 - число **всех нечетных** чисел с номерами больше i-5 и меньше i
- k2 – число **всех четных** чисел с номерами больше i-5 и меньше i
- k13 – число нечетных чисел кратных 13 с номерами больше i-5 и меньше i
- k26 – число четных чисел кратных 13 с номерами больше i-5 и меньше i
- k – количество различных пар, удовлетворяющих условию для первых 5-ти элементов (можно сделать любым способом)

Каждый новый элемент обрабатываем как раньше, но перед этим удаляем элемент из очереди. В тот момент, когда элемент удаляется из очереди некоторые из счетчиков необходимо изменить (уменьшить). Таким образом в рассмотрении у нас всегда только 5 элементов и значения счетчиков только для этих 5-ти элементов (это так называемое скользящее окно).

Рассмотрим пример – пользователь ввел 8, а затем вводит следующие числа:

13	26	2	4	7	8	39	2

Сначала считываем первые 5 чисел в очередь и подсчитываем значения счетчиков

13	26	2	4	7

k1 = 2, k2 = 3, k13 = 1, k26 = 1, k = 4 (пары: 13 и 2, 13 и 4, 13 и 26, 26 и 7)

На очередном шаге элемент 13 должен выйти из рассмотрения, но вместе с этим должны измениться и значения счетчиков:

k1 = 1, k2 = 3, **k13 = 0**, k26 = 1

Теперь читаем новое число – это 8. Для него мы можем использовать только нечетные числа кратные 13 (т.е. k13), но таких чисел нет, поэтому k не меняется.

После изменения (или нет) количества пар нужно изменить счетчики. Т.к. 8 четное число, то счетчики и очередь примут следующие значения:

26	2	4	7	8

k1 = 1, **k2 = 4**, k13 = 0, k26 = 1, k = 4

Следующий шаг – элемент 26 выходит из рассмотрения: k1 = 1, **k2 = 3**, k13 = 0, **k26 = 0**

Новый элемент – число 39. Для него мы можем составить пары с любым четным, поэтому увеличиваем счетчик числа пар (т.е. k) на 3, а затем меняем остальные счетчики и добавляем новый элемент в очередь

2	4	7	8	39

k1 = 2, k2 = 3, **k13 = 1**, k26 = 0, **k = 7**

На последнем шаге очередь покидает элемент 2: k1 = 2, **k2 = 2**, k13 = 1, k26 = 0
Новый элемент – число 2. Пары можно составить только с нечетными числами кратными 13, а такое число одно. В завершении алгоритма значение элементов очереди и счетчиков становятся следующими:

4	7	8	39	2

k1 = 2, **k2 = 3**, k13 = 1, k26 = 0, **k = 8**

Программа выводит – 8 (пары: 13 и 2, 13 и 4, 13 и 26, 26 и 7, 2 и 39, 4 и 39, 8 и 39, 39 и 2).

Решение на **Python**:

```
D = 5
n = int(input())
queue = []
k = k26 = k13 = k2 = k1 = 0
for i in range(min(D,n)):
  x = int(input())
  queue.append(x)
  if x % 13 == 0:
    if x % 2 == 0:
      k, k26, k2 = k+k1, k26+1, k2+1
    else:
      k, k13, k1 = k+k2, k13+1, k1+1
  elif x % 2 == 0:
    k, k2 = k+k13, k2+1
  else:
    k, k1 = k+k26, k1+1
for i in range(D,n):
  # элемент удаляется из очереди
  if queue[i % D] % 13 == 0:
    if queue[i % D] % 2 == 0:
      k26, k2 = k26-1, k2-1
    else:
      k13, k1 = k13-1, k1-1
  elif queue[i % D] % 2 == 0:
    k2 -= 1
  else:
    k1 -= 1
  # вводим элемент
  x = int(input())
  if x % 13 == 0:
    if x % 2 == 0:
      k += k1
      k26, k2 = k26+1, k2+1
    else:
      k += k2
      k13, k1 = k13+1, k1+1
  elif x % 2 == 0:
    k, k2 = k+k13, k2+1
  else:
    k, k1 = k+k26, k1+1
  queue[i % D] = x
print(k)
```

83) **Задача A.** Полный перебор

Перебираем все пары (a_i, a_j), для которых $i < j$. Считаем все подходящие пары с помощью счётчика **count**.

Решение на **Паскале**:

```
const MAXN = 1000;
var N, i, j, count: integer;
    a: array[1..MAXN] of integer;
begin
  readln(N);
  for i:=1 to N do
    readln(a[i]);
  count := 0;
  for i:=1 to N-1 do
    for j:=i+1 to N do
      if a[i] * a[j] mod 14 <> 0 then
        count := count + 1;
  writeln(count)
end.
```

Задача B (эффективное решение).

Идея состоит в том, чтобы сначала найти количество пар, произведение которых кратно 14 (так же, как в задаче 75), и вычесть это количество из общего количества всех возможных пар. При выполнении условия $i < j$ количество пар с $i = 1$ равно $N - 1$, количество пар с $i = 2$ равно $N - 2$ и т.д., так что общее количество пар равно

$$(N-1) + (N-2) + \ldots + 2 + 1,$$

что по формуле суммы первый членов арифметической прогрессии равно

$$N \cdot (N-1) / 2$$

Произведение двух чисел кратно 14, если:

- один из сомножителей кратен 14 (второй может быть любым);
- ни один из сомножителей не кратен 14, но один из сомножителей кратен 2, а другой кратен 7.

Поэтому программа, вычисляющая количество пар элементов (a_i, a_j), произведение которых кратно 14 и $1 \le i \le j \le N$, может работать следующим образом.

Программа читает все входные данные один раз, не запоминая данные в массиве.

Программа для прочитанного фрагмента входной последовательности хранит значения трёх величин:

- M_2 — количество чисел, кратных 2, но не кратных 14;
- M_7 — количество чисел, кратных 7, но не кратных 14;
- M_{14} — количество чисел, кратных 14.

Первый из элементов, кратных 14, может быть умножен на любой из других $N - 1$ элементов (все элементы последовательности, не включая его самого). Следующий элемент, кратный 14, может быть умножен только на любой из $N - 2$ элементов (так как пара с первым элементом, кратным 14, уже была учтена). Рассуждая аналогично, получим, что количество пар, одним из элементов которой является элемент, кратный 14, равно $(N-1) + (N-2) + \ldots + (N - M_{14}) = M_{14} \cdot N - (1 + 2 + \ldots + M_{14})$.

Сумма членов арифметической прогрессии $(1 + 2 + \ldots + M_{14})$ равна $M_{14} \cdot (M_{14} + 1) / 2$.

Множество пар, в которых один из элементов кратен 2, а второй кратен 7, и при этом оба элемента не кратны14, не пересекается с рассмотренным выше множеством пар, в которых один из элементов кратен 14. Количество таких пар равно $M_2 \cdot M_7$.

После того как все данные прочитаны, требуемый в условии ответ вычисляется по формуле $M_{14} \cdot N - M_{14} \cdot (M_{14} + 1) / 2 + M_2 \cdot M_7$.

Решение на Pascal:

```
var N, x, i, j, m2, m7, m14, count14, total: integer;
begin
  m2 := 0; m7 := 0; m14 := 0;
  readln(N);
  for i := 1 to N do begin
    readln(x);
    if x mod 14 = 0 then
      m14 := m14 + 1
    else begin
      if x mod 2 = 0 then
        m2 := m2 + 1;
      if x mod 7 = 0 then
        m7 := m7 + 1
    end
  end;
  total := N*(N-1) div 2;
  count14 := m14 * N - m14 * (m14 + 1) div 2 + m2 * m7;
  writeln(total - count14)
end.
```

84) **Задача A.** Полный перебор

Для хранения двух элементов найденной пары введём переменные $r1$ и $r2$, а в переменной $pMax$ будем хранить максимальное произведение.

Перебираем все пары (a_i, a_j), для которых $i < j$. Проверяем для каждой пары три условия:

- сумма значений a_i и a_j нечётная
- произведение значений a_i и a_j делится на 5
- произведение значений a_i и a_j больше, чем $pMax$

Если все они выполняются, запоминаем значения a_i и a_j в переменных $r1$ и $r2$, а их произведение – в переменной $pMax$.

Решение на **Паскале**:

```
const MAXN = 1000;
var N, i, j, r1, r2, pMax: integer;
    a: array[1..MAXN] of integer;
begin
  readln(N);
  for i:=1 to N do
    readln(a[i]);
  pMax := 0;
  for i:=1 to N-1 do
    for j:=i+1 to N do
      if ((a[i] + a[j]) mod 2 <> 0) and
         (a[i]*a[j] mod 5 = 0) and (a[i]*a[j] > pMax) then begin
        r1 := a[i];
        r2 := a[j];
        pMax := r1*r2;
      end;
  if pMax > 0 then
    writeln(r1, ' ', r2)
  else
    writeln(0)
end.
```

Можно обойтись и без переменной *pMax*, так как её значение всегда равно произведению *r1* и *r2*. Но при этом переменным *r1* и *r2* нужно до цикла задать начальные нулевые значения.

Задача B (эффективное решение).

Выделим 4 группы чисел:

- *Rxx* – не делятся ни на 2, ни на 5
- *R2x* – делятся на 2, не делятся на 5
- *Rx5* – не делятся на 2, делятся на 5
- *R10* – делятся и на 2, и на 5 (то есть, делятся на 10)

Чтобы сумма получилась нечётной, одно число должно быть чётным, второе – нечётным, то есть можно «компоновать» только пары из групп *Rxx-R2x*, *Rxx-R10*, *Rx5-R2x* или *Rx5-R10*. Но произведение должно делиться на 5, поэтому пары использовать *Rxx-R2x* нельзя: остаются только *Rxx-R10*, *Rx5-R2x* и *Rx5-R10*. Нас интересует только максимальное значение произведения, поэтому для каждой их 4-х групп достаточно хранить значение максимального элемента, принадлежащего этой группе.

Решение на Pascal:

```
var i, N, x: integer;
    max_xx, max_2x, max_x5, max_10: integer;
    r1, r2: integer;
begin
  max_xx := 0; max_2x := 0;
  max_x5 := 0; max_10 := 0;
  readln(N);
  for i:=1 to N do begin
    readln(x);
    if (x mod 2 = 0) and (x mod 5 = 0) then
      max_10 := max(max_10, x)
    else if x mod 2 = 0 then
      max_2x := max(max_2x, x)
    else if x mod 5 = 0 then
      max_x5 := max(max_x5, x)
    else
      max_xx := max(max_xx, x);
  end;
  max_xx := max(max_xx, max_x5);
  if max_xx*max_10 > max_2x*max_x5 then begin
    r1 := max_xx;
    r2 := max_10;
  end
  else begin
    r1 := max_2x;
    r2 := max_x5;
  end;
  if r1 = 0 then
    writeln(0)
  else
    writeln(r1, ' ', r2);
end.
```

85) **Задача A.** Полный перебор

Перебираем все пары (a_i, a_j), для которых $i < j$. Проверяем каждую пару: если сумма значений a_i и a_j делится на 12, увеличиваем счётчик пар.

Решение на **Паскале**:

```
const MAXN = 1000;
var N, i, j, count: integer;
    a: array[1..MAXN] of integer;
begin
  readln(N);
  for i:=1 to N do
    readln(a[i]);
  count := 0;
  for i:=1 to N-1 do
    for j:=i+1 to N do
      if (a[i] + a[j]) mod 12 = 0 then
        count := count + 1;
  writeln(count);
end.
```

Задача B (эффективное решение).

Чтобы сумма двух чисел делилась на 12, нужно, чтобы сумма остатков этих чисел от деления на 12 делилась на 12. Например, это могут быть два числа, которые делятся на 12 (дают в остатке 0) или числа, одно из которых даёт остаток 5, а второе – 7.

Подсчитаем в массиве **r** количество чисел, имеющих определённый остаток: в **r[i]** будем хранить количество чисел, которые при делении на 12 дают остаток **i**. Получив на вход очередное значение **x**, увеличим соответствующий ему счётчик:

```
  for i:=1 to N do begin
    readln(x);
    r[x mod 12] := r[x mod 12] + 1;
  end;
```

Теперь нужно понять, что с этими счётчиками делать. Числа, которые дали остаток 0, компонуются только с числами из этой же группы, причём для каждого из них найдется **r[0]-1** парных (число не может образовать пару само с собой). При этом все такие пары будет посчитаны дважды (*ab* и *ba*), поэтому это количество нужно поделить на 2. Аналогичная ситуация с числами, которые дают в остатке 6:

```
  count := (r[0]*(r[0]-1) + r[6]*(r[6]-1)) div 2;
```

В остальных парах остатки разные. Например, каждое число, дающее в остатке 1, сочетается с каждым числом, дающим в остатке 11, число таких пар – **r[1]*r[11]**. В итоге получается цикл, где первый остаток изменяется от 1 до 5 (так чтобы второй остаток был больше него, иначе будут повторения пар):

```
  for i:=1 to 5 do
    count := count + r[i]*r[12-i];
```

Решение на Pascal:

```
var i, N, x, count: integer;
    r: array[0..11] of integer;
begin
  readln(N);
  for i:=0 to 11 do r[i] := 0;
  for i:=1 to N do begin
    readln(x);
    r[x mod 12] := r[x mod 12] + 1;
  end;
  count := (r[0]*(r[0]-1) + r[6]*(r[6]-1)) div 2;
  for i:=1 to 5 do
    count := count + r[i]*r[12-i];
  writeln(count);
end.
```

86) **(О.Л. Дуркин)**

Задача А. Полный перебор

Перебираем все тройки (a_i, a_j, a_k), для которых $i < j < k$. Проверяем каждую тройку: если сумма значений делится на 7 и не делится на 2 (нечётна), увеличиваем счётчик **m**.

Решение на **Паскале**:

```
var a:array[1..10000] of integer;
    n,i,j,k,m:integer;
begin
  m:=0;
  read(n);
  for i:=1 to n do
    read(a[i]);
  for i:=1 to n-2 do
    for j:=i+1 to n-1 do
      for k:=j+1 to n do
        if((a[i]+a[j]+a[k]) mod 7 = 0) and
          ((a[i]+a[j]+a[k]) mod 2 <> 0) then begin
          inc(m);
        end;
  writeln(m);
end.
```

Решение на C++:

```
#include <iostream>
using namespace std;
int main()
{
  int a[1000],n,i,j,k,m=0;
  cin >> n;
  for(i=0;i<n;i++)
    cin >> a[i];
  for(i=0;i<n-2;i++)
    for(j=i+1;j<n-1;j++)
      for(k=j+1;k<n;k++)
        if((a[i]+a[j]+a[k])%7==0 &&
           (a[i]+a[j]+a[k])%2!=0 )
          m++;
  cout << m;
  return 0;
}
```

Задача В.

Эффективное решение во многом аналогично решению задачи 79.

Решение на Паскале:

```
var
  a:array[0..1,0..6] of integer;
  n,i,j,l,k,x:integer;
begin
  for i:=0 to 6 do begin
    a[0,i]:=0;
    a[1,i]:=0;
  end;
```

```
  read(n);
  for i:=1 to n do begin
    read(x);
    inc(a[x mod 2, x mod 7]);
  end;
  k:=0;
  for i:=0 to 4 do
    for j:=i+1 to 5 do
      for l:=j+1 to 6 do
        if(i+j+l) mod 7 = 0 then begin
          k:=k+a[1,i]*a[0,j]*a[0,l];
          k:=k+a[0,i]*a[1,j]*a[0,l];
          k:=k+a[0,i]*a[0,j]*a[1,l];
          k:=k+a[1,i]*a[1,j]*a[1,l];
        end;

  for i:=1 to 6 do begin
    k:=k+a[1,(14-i-i) mod 7]*(a[0,i]*(a[0,i]-1) div 2);
    k:=k+a[1,(14-i-i) mod 7]*(a[1,i]*(a[1,i]-1) div 2);
    k:=k+a[0,(14-i-i) mod 7]*a[1,i]*a[0,i];
  end;
  k:=k+a[1,0]*(a[0,0]*(a[0,0]-1) div 2);
  k:=k+a[1,0]*(a[1,0]-1)*(a[1,0]-2) div 6;
  writeln(k);
end.
```

Решение на C++:

```
#include <iostream>
using namespace std;
int main(){
    int a[2][7];
    for(int i=0;i<7;i++){
        a[0][i]=0;
        a[1][i]=0;
    }
    int n,x,k=0;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>x;
        a[x%2][x%7]++;
    }
    for (int i=0;i<=4;i++)
        for (int j=i+1;j<=5;j++)
            for (int l=j+1;l<=6;l++)
                if((i+j+l)%7 == 0){
                    k=k+a[1][i]*a[0][j]*a[0][l];
                    k=k+a[0][i]*a[1][j]*a[0][l];
                    k=k+a[0][i]*a[0][j]*a[1][l];
                    k=k+a[1][i]*a[1][j]*a[1][l];
                }
    for (int i=1;i<=6;i++){
        k=k+a[1][(14-i-i)%7]*(a[0][i]*(a[0][i]-1)/2);
        k=k+a[1][(14-i-i)%7]*(a[1][i]*(a[1][i]-1)/2);
        k=k+a[0][(14-i-i)%7]*a[1][i]*a[0][i];
```

```
    }
    k=k+a[1][0]*(a[0][0]*(a[0][0]-1)/2);
    k=k+a[1][0]*(a[1][0]-1)*(a[1][0]-2)/6;
    cout<<k;

    return 0;
}
```

Ещё одно решение на C++ (М. Коротков):

Из троек «кратных» 7 вычитаем тройки «кратные» 14 (т.е. вычли все «чётные»). Остаются «нечётные» тройки «кратные» 7.

```
#include <iostream>

using namespace std;

int count_triples(int a[], const int size) {
    int triples = 0;
    for (int i = 0; i < size; i++)
        for (int j = i; j < size; j++)
            for (int k = j; k < size; k++)
                if ((i + j + k) % size == 0)
                    if (i == j && j == k)
          triples += a[i] * (a[i]-1) * (a[i]-2) / 6;
                    else if (i == j)
          triples += a[i] * (a[i]-1) * (a[k]) / 2;
                    else if (j == k)
          triples += a[i] * (a[j]) * (a[j]-1) / 2;
                    else
          triples += a[i] * a[j] * a[k];
    return triples;
}

int main() {
    int N;
    cin >> N;

    int a[7] = {0};
    int b[14] = {0};

    for (int i = 0; i < N; i++) {
        int num;
        cin >> num;
        a[num % 7]++;
        b[num % 14]++;
    }

    int triples7 = count_triples(a, 7);
    int triples14 = count_triples(b, 14);

    cout << triples7 - triples14;
    return 0;
}
```

Решение на Python (Valia Sergeev):

Сначала мы заполняем массивы chet и nechet. Каждый из них содержит 7 элементов, где **i**-й элемент – это количество чисел, которые при делении на 7 в остатке дают **i**. **Chet** - для подсчёта остатков чётных чисел, **nechet** – для нечётных.

Далее во вложенных циклах мы рассматриваем все возможные комбинации отстатков, которые бы обеспечивали делимость на 7. (В частности: сумма отстатков должна делиться на 7). Для предотвращения дублирования выставляем нижние границы внутренних циклов на значения 'наиболее близких' к ним внешних переменых (для второго цикла – граница **i**, для третьего – **j**). И последним шагом мы считаем количество пар для каждого полученного варианта, исходя из того, что сумма чисел должна быть нечётна. (далее н – нечётное, ч – чётное)

н + н + н = н, н + ч + ч = н

Нужно заметить, что при соблюдении равенства любых двух из **i**, **j**, **k** попарно, необходимо учесть, что в случае перемножения количества остатков из одного массива (либо оба из **chet**, либо оба из **nechet**), необходимо уменьшить один из множителей на единицу, так как берём *из одного множества*. Поскольку в этом случае каждая пара из одного множества считается дважды, полученное количество нужно разделить на 2.

В случае взятие всех трёх чисел из массива **nechet** при **i = j = k** необходимо уменьшить один множитель на 1 а другой на 2 по той же причине. Кроме того, это количество вариантов нужно разделить на 6, чтобы исключить повторояющиеся тройки.

```
N = int(input())

nechet = [0] * 7
chet = [0] * 7
res = 0

def cnt(i, j, k):
    global res
    if i == j == k:
        res += nechet[i]*(nechet[i] - 1)*(nechet[i] - 2) // 6 \
               + nechet[i] * chet[i] * (chet[i] - 1) // 2
    elif i == j:
        res += nechet[i] * (nechet[i] - 1) * nechet[k] // 2 \
               + chet[i] * (chet[i] - 1) * nechet[k] // 2 \
               + chet[i] * nechet[i] * chet[k]
    elif j == k:
        res += nechet[j] * (nechet[j] - 1) * nechet[i] // 2\
               + chet[j] * (chet[j] - 1) * nechet[i] // 2 \
               + chet[j] * nechet[j] * chet[i]
    elif i == k:
        res += nechet[i] * (nechet[i] - 1) * nechet[j] // 2\
               + chet[i] * (chet[i] - 1) * nechet[j] // 2\
               + chet[i] * nechet[i] * chet[j]
    else:
        res += nechet[i] * nechet[j] * nechet[k]\
               + chet[i] * chet[j] * nechet[k]\
               + chet[i] * nechet[j] * chet[k]\
               + nechet[i] * chet[j] * chet[k]

for i in range(N):
    num = int(input())
    if num % 2 == 0:
```

```
        chet[num % 7] += 1
    else:
        nechet[num % 7] += 1

for i in range(0, 7):
    for j in range(i, 7):
        for k in range(j, 7):
            if (i + j + k) % 7 == 0:
                cnt(i, j, k)
print(res)
```

Красивое и эффективное решение этой задачи предложил **Алексей Пискунов**.

Для начала переформулируем условие. Фактически требуется найти **тройки чисел, сумма при делении на 14 которых даёт остаток 7**. Введём два массива из 14 элементов (это не влияет на эффективность по памяти, потому что их размеры не зависят от N):

rem14[i] – количество чисел, которые при делении на 14 дают остаток **i**;

pairs14[i] – количество пар чисел, сумма которых при делении на 14 даёт остаток **i**.

Переменная **count** – это искомое количество троек.

В цикле, прочитав новый элемент последовательности **x**, обновляем массивы **rem14**, **pairs14** и значение переменной **count**.

Сначала считаем количество подходящих троек с участием нового числа **x**. Два оставшихся числа (скажем, **a** и **b**) в такой тройке должны быть такими, чтобы

```
(a + b + x) % 14 = 7
```

то есть

```
a + b + x = 14*k + 7,
```

где **k** – некоторое неотрицательное целое число. Поэтому сумма остатков

```
(a + b) % 14 + x % 14
```

может быть равна 7 или 21. Тогда сумма чисел **a** и **b** должна быть равна **7-(x%14)** или **21-(x%14)**. Эти два варианта можно объединить формулой

```
(21 - (x%14)) % 14
```

Таким образом, количество походящих троек, которые образует **x** с предыдущими числами, равно количеству пар с остатком **(21-(x%14))%14**. Поэтому счётчик будет изменяться так:

```
count += pairs14 [(21-(x%14))%14];
```

При обновлении массива pairs14 нужен цикл по всем возможным остаткам:

```
for(j = 0; j < 14; j++)
  pairs14[(j+x)%14] += rem14[j];
```

Этот цикл означает следующее: если взять любое число, которое даёт в остатке **j**, и сложить с **x**, то сумма это пары будет иметь остаток **(j+x)%14**, и мы увеличиваем соответствующий счётчик пар в массиве **pairs**.

Обновление массива **pairs14** делаем после изменения счётчика **count**, потому что нас интересуют только тройки, который **x** образует с **предыдущими** парами.

Наконец, обновляем массив **rem14**, учитывая **x**:

```
rem14[x%14]++;
```

Это нужно сделать после того, как мы изменим переменную **count** и массив **pairs14**, иначе мы учтём тройки, в которые **x** входит дважды.

Вот полная программа:

```
#include <iostream>
using namespace std;
int main(){
  int n, x, count=0;
  int rem14[14] = {0},
```

```
      pairs14[14] = {0};
  cin >> n;
  for(int i = 0; i < n; i++) {
    cin >> x;
    count += pairs14[(21-(x%14))%14];
    for(int j = 0; j < 14; j++)
      pairs14[(j+x)%14] += rem14[j];
    rem14[x%14]++;
    }
  cout << count;
}
```

Таким же способом можно решить и задачу 79.

Решение на языке Паскаль (Н.А. Голяков)

```
var n,i,j,kol,ch: integer;
   k: array[0..1,0..6] of integer;
    // k[четность] [остаток] – массив количества
    // чисел с указанной четностью и остатком
   k2: array[0..1,0..6] of integer;
    // k2[четность] [остаток] – массив количества пар
    // суммы чисел с указанной четностью и остатком
begin
  readln(n); kol:=0;
  for i:= 0 to 6 do begin
    k[0][i]:=0; k[1][i]:=0;
    k2[0][i]:=0; k2[1][i]:=0;
  end;
  for i:= 1 to n do begin
    read(ch);
    kol:=kol+k2[1-ch mod 2][(7-ch mod 7) mod 7];
      // Увеличиваем количество групп троек на значение
      // такой пары, которая в сумме с новым числом
      // удовлетворяет условию нечетности  и кратности 7
      // если число четное  (ch mod 2=0), пара должна
      // быть нечетной (1-ch mod 2=1-0=1)
      // если число нечетное  (ch mod 2=1), пара должна
      //быть четной (1-ch mod 2=1-1=0)

  for j:=0 to 6 do begin
    k2[0][(j+ch) mod 7]:=k2[0][(j+ ch) mod 7]+k[ch mod 2][j];
    k2[1][(j+ch) mod 7]:=k2[1][(j+ch) mod 7]+k[1-ch mod 2][j];
      // или
      // k2[0][j]:=k2[0][j]+k[ch mod 2][(7+j-ch mod 7) mod 7];
      // k2[1][j]:=k2[1][j]+k[1-ch mod 2][(7+j-ch mod 7) mod 7];
  end;
    // увеличиваем значения 14 пар с учетом нового числа
  inc(k[ch mod 2][ch mod 7]);
    // увеличиваем массив количества чисел с указанной
    // четностью и остатком для нового числа на 1
  end;
  writeln(kol);
end.
```

Решение на языке C++ (П. Дьяченко)

```
#include <iostream>
```

```
#include <vector>
using namespace std;
int main() {
int m; cin » m;
vector<int> ch(7, 0);
vector<int> n(7, 0);
long long k = 0;
for (int i = 0; i < m; i ++) {
    int a; cin » a;
    int s = (7-a%7); // сумма, которую нужно получить
                     // в остатках двух других чисел..
    if (a%2 == 0) {  // Если число четное, то два других четное
                     // с нечетным (7 групп)
        for (int i = 0; i < 7; i ++) {
        k += ch[i]*n[(s-i+7)%7];
    }
    ch[a%7]++;
    } else {   //Если число нечетное - то два других
               // оба нечетны или оба четны
    for (int i = 0; i < 7; i ++) {
        if (i<=(s-i+7)%7) {  //Убираем половину повторений
            int pl = ch[i]*ch[(s-i+7)%7];
            int pl2 = n[i]*n[(s-i+7)%7];
            if (i == (s-i+7)%7) { //Если из одного отбора
                pl = ch[i] * (ch[i]-1) / 2;
                pl2 = n[i] * (n[i]-1) / 2;
            }
            k += pl + pl2;
        }
    }
    n[a%7]++;
}
}
cout « k « endl;
return 0;
}
```

87) **(А. Жуков)**

Задача А. Полный перебор

Идея переборного решения – проверить все возможные суммы. Для этого нужно последовательно вычислять сначала все возможные суммы последовательностей начиная с первого элемента, затем начиная со второго элемента и т.д.

Решение на Паскале ABC.NET:

```
const MAXN = 1000;
var a:array [1..MAXN] of integer;
    i,j,n,s,mxs: integer;
begin
  readln(n);
  for i:=1 to n do
    readln(a[i]);

  mxs := a[1];
  for i:=1 to n do begin
    s := 0;
```

```
    for j:=i to n do begin
      s += a[j];
      if s > mxs then
        mxs := s;
    end;
  end;
  writeln(mxs);
end.
```

Решение на GCC 4.6.3:

```
#include <iostream>
using namespace std;
int main() {
  const int MAXN = 1000;
  int a[MAXN];
  int i,j,n,s,mxs;
  cin >> n;
  for(i=0;i<n;++i) cin>>a[i];
  mxs = a[0];
  for(i=0;i<n;++i) {
    s = 0;
    for(j=i;j<n;++j) {
      s += a[j];
      mxs = max(mxs, s);
    }
  }
  cout << mxs;
  return 0;
}
```

Решение на Python:

```
n = int(input())
a = []
for i in range(n):
  a.append(int(input()))
mxs = a[0]
for i in range(n):
  s = 0
  for j in range(i,n):
    s += a[j]
    mxs = max(s, mxs)
print(mxs)
```

На Python 3 переборное решение можно написать и так ☺:

```
a = [int(input()) for i in range(int(input()))]
print(max([sum(a[i:j+1]) for i in range(len(a))
                         for j in range(i,len(a))]))
```

Задача Б. Идея эффективного решения опирается на методологию динамического программирования. Рассмотрим вначале программу неэффективную по памяти, т.е. использующую массив с размером, зависящим от объема входных данных.

Элемент массива **DP[i]** - максимальная сумма, которая может быть получена для последовательности элементов массива, последний из которых имеет номер **i**.

Рассмотрим это на примере массива из 3х элементов:

-2	1	-3

Значение DP[1] = -2, т.к. нет элементов предшествующих 1-му
Значение DP[2] = 1, последовательность 1 (нет смысла включать в последовательность -2)
Значение DP[3] = -2, последовательность 1-3
Таким образом, для каждого следующего элемента последовательности имеет смысл продолжать последовательность только, если сумма элементов последовательности положительна, иначе нужно начать новую последовательность. **DP[1]** = первому элементу последовательности, т.к. нет предшествующих элементов. Рекуррентная формула для вычисления элементов массива DP следующая:

DP[i] = max(DP[i-1], 0) + a[i]

где **a[i]** – элемент последовательности.
Ответ на задачу находим, как максимальный элемент из массива **DP**.

Решение на Паскале ABC.NET:

```
const MAXN = 1000;
var a:array [1..MAXN] of integer;
    dp:array [1..MAXN] of integer;
    i,j,n,s,mxs,x: integer;
begin
  readln(n);
  for i:=1 to n do
    readln(a[i]);
  dp[1] := a[1];
  mxs := dp[1];
  for i:=2 to n do begin
    dp[i] := max(dp[i-1], 0) + a[i];
    mxs := max(dp[i], mxs);
  end;
  writeln(mxs);
end.
```

Для того, чтобы сделать решение эффективным по памяти, нужно отказаться от массивов. Сделать это достаточно просто с учетом того, что на каждом шаге цикла мы используем только текущее значение из последовательности и только предыдущее значение вспомогательного массива DP.

Решение на Паскале ABC.NET:

```
var i,dp,n,s,mxs,x: integer;
begin
  readln(n);
  readln(dp);
  mxs := dp;
  for i:=2 to n do begin
    readln(x);
    dp := max(dp, 0) + x;
    mxs := max(dp, mxs);
  end;
  writeln(mxs);
end.
```

Решение на GCC 4.6.3:

```
#include <iostream>
using namespace std;
int main() {
  int i,dp,n,s,mxs,x;
  cin >> n;
  cin >> dp;
```

```
  mxs = dp;
  for(i=1;i<n;++i) {
    cin >> x;
    dp = max(dp, 0) + x;
    mxs = max(mxs, dp);
  }
  cout << mxs;
  return 0;
}
```

Решение на Python:

```
n = int(input())
mxs = dp = int(input())
for i in range(1,n):
  x = int(input())
  dp = max(dp, 0) + x
  mxs = max(dp, mxs)
print(mxs)
```

88) **(А. Жуков)**

Задача А. Полный перебор

Неэффективное решение заключается в рассмотрении всех вариантов продажи и покупки. Доход – это разность между стоимостью продажи и стоимостью покупки.

Решение на Паскале ABC.NET:

```
const MAXN = 1000;
var a:array [1..MAXN] of integer;
    i,j,mx,n: integer;
begin
  readln(n);
  for i:=1 to n do
    readln(a[i]);
  mx := 0;
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if a[j] - a[i] > mx then
        mx := a[j] - a[i];
  writeln(mx)
end.
```

Задача Б.

Идея эффективного решения – использование динамического программирования. Для этого немного переформулируем задачу: вместо исходной последовательности стоимостей акций возьмем разности в стоимостях между текущим и предыдущим днями. Тогда для массива из примера получаем следующий массив **b**:

a	10	2	5	4	8	7	1	6	4
b	0	-8	3	-1	4	-1	-6	5	-2

`b[1] = 0`, так как нам не известно об изменении курса.

Теперь для того, чтобы найти максимальную выгоду **нужно найти максимальное изменение курса**, т.е. максимальную сумму последовательности в массиве **b**, т.е. задача сводится к предыдущей. Ниже представлено решение эффективное по времени, но **не эффективное по памяти**:

Решение на Паскале ABC.NET:

```
const MAXN = 1000;
```

```
var a,b,dp: array[1..MAXN] of integer;
    i,j,mx,n: integer;
begin
  readln(n);
  readln(a[1]);
  b[1] := 0; // первое значение изменения - 0, т.е. оно не
определено
  for i:=2 to n do begin
    readln(a[i]);
    b[i] := a[i] - a[i-1]; // изменение курса в соседние дни
  end;
  mx := 0;
  // dp - массив для подсчета максимальной суммы
  dp[1] := b[1];
  for i := 2 to n do begin
    dp[i] := max(dp[i-1], 0) + b[i];
    mx := max(mx, dp[i]);
  end;
  writeln(mx)
end.
```

Эффективное решение без массивов: так как в каждый день нам нужно знать только стоимость в этот день и в предыдущий, то в решении задачи можно обойтись без массивов – потоковая обработка поступающих данных. Это решение эффективно и по времени и по памяти.

Решение на Паскале ABC.NET:

```
var dp, cur, prev: integer;
    i, mx, n: integer;
begin
  readln(n);
  readln(prev);
  mx := 0;
  dp := 0;
  for i:=2 to n do begin
    readln(cur);
    dp := max(dp, 0) + cur - prev;
    mx := max(mx, dp);
    prev := cur;
  end;
  writeln(mx)
end.
```

Ещё одно решение на Паскале ABC.NET (С. Попкович):

```
var N, i, x, min, rmax: integer;
begin
  readln(N, x);
  min:= x; rmax:= -10001;
  for i:= 2 to N do begin
    readln(x);
    if x - min > rmax then
      rmax:= x - min;
    if x < min then
      min:= x;
  end;
  writeln(rmax);
```

```
end.
```

Решение на GCC 4.6.3:

```
#include <iostream>
using namespace std;
int main() {
  int dp, cur, prev, mx, i, n;
  cin >> n >> prev;
  mx = dp = 0;
  for(i=1; i<n; ++i) {
    cin >> cur;
    dp = max(dp, 0) + cur - prev;
    mx = max(mx, dp);
    prev = cur;
  }
  cout << mx;
  return 0;
}
```

Решение на Python:

```
n = int(input())
prev = int(input())
mx = dp = 0
for i in range(n-1):
  cur = int(input())
  dp = max(dp, 0) + cur - prev
  mx = max(mx, dp)
  prev = cur
print(mx)
```

Другое решение (**Муфаззалов Д.Ф., Уфа**) состоит в вычислении разности между текущей стоимостью акций и минимальной стоимостью акций в предшествующие дни с сохранением максимальной из этих разностей.

Решение на GCC 4.6.3:

```
#include <iostream>
using namespace std;
int main()
{
    int mn = 10000, cur, mx=0, n;
    for(cin >> n; n--; ) {
     cin>>cur;
     mn=min(mn,cur);//минимальная стоимость в предшествующие дни
     mx=max(cur-mn,mx);//максмальная прибыль
    }
    cout<<mx;
    return 0;
}
```

Решение на Паскале (В. Бабий):

```
var i, n, x, mp, mc: integer;
begin
  readln(n);
  readln(mc);
  mp := -10000;
  for i:=2 to n do begin
    readln(x);
```

```
    mp := max(mp, x-mc);
    mc := min(mc, x);
  end;
  writeln(mp);
end.
```

89) **(А. Жуков)**

Задача А. Полный перебор

Неэффективное решение заключается в переборе всех пар и выборе максимальной разности.

Решение на Паскале ABC.NET:

```
const MAXN = 1000;
var a:array [1..MAXN] of integer;
    i,j,mx,n,si,sj: integer;
begin
  readln(n);
  for i:=1 to n do
    readln(a[i]);
  mx := 0;
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if a[j] - a[i] > mx then begin
        mx := a[j] - a[i];
        si := i;
        sj := j;
      end;
  if mx > 0 then
    writeln(mx, ' ', si, ' ', sj)
  else
    writeln(0);
end.
```

Задача Б. Рассмотрим эффективное решение на 4 балла: оно практически не отличается от решения, которое мы рассматривали в предыдущей задаче, но добавляются еще переменные для хранения начала и конца подпоследовательности.

Решение на Паскале ABC.NET:

```
var i, mx, dpi, n, x, prev: integer;
    st, fn, t: integer;
begin
  readln(n);
  readln(prev);
  dpi := 0;
  mx := 0;
  for i:=2 to n do begin
    readln(x);
    if dpi <= 0 then begin // начинаем новую последовательность
      dpi := x - prev;
      t := i - 1; // день, когда сумма была меньше
    end else
      dpi += x - prev; // продолжаем старую последовательность
    if dpi >= mx then begin // новый максимум
      mx := dpi;
      st := t;
      fn := i;
```

```
    end;
    prev := x;
  end;
  write(mx);
  if mx > 0 then
    writeln(' ', st, ' ', fn)
end.
```

Решение на GCC 4.6.3:

```
#include <iostream>
using namespace std;
int main() {
  int i, mx, dpi, n, x, prev, st, fn, t;
  cin >> n >> prev;
  dpi = 0;
  mx = 0;
  for(i=1; i<n; ++i) {
    cin >> x;
    if (dpi <= 0) {
      dpi = x - prev;
      t = i;
    } else
      dpi += x - prev;
    if (dpi >= mx) {
      mx = dpi;
       st = t;
       fn = i + 1;
    }
    prev = x;
  }
  cout << mx;
  if (mx > 0) cout << " " << st << " " << fn;
  return 0;
}
```

Решение на Python:

```
n = int(input())
prev = int(input())
dp = mx = 0
for i in range(1,n):
  x = int(input())
  if dp <= 0:
    dp, t = x - prev, i
  else:
    dp += x - prev
  if dp >= mx:
    mx, st, fn = dp, t, i+1
  prev = x
print(mx, end=' ')
if mx > 0:
  print(st, fn)
```

Ещё одно эффективное решение прислал **Станислав Мильке** (г. Анапа). Идея состоит в том, что мы запоминаем текущий минимум и ищём максимальное значение курса после этого минимума. На каждом шаге проверяем разницу максимума и минимума (доход), есть доход больше, чем запомненный предыдущий результат, обновляем результат. Если

встретился новый минимум, начинаем искать наибольший доход, который может принести покупка в этот день (если покупать раньше, доход будет меньше).

```
var N, profit, i, dayBuy, daySell,
    bit, min, max, dayMin, dayMax:integer;
begin
  readln(N);
  min := -1;
  profit := 0;
  for i:=1 to N do begin
    readln(bit); // считываем стоимость биткойна
    if (min = -1) or (bit < min) then begin
      // переприсвоим минимум и максимум в буфере
      // при достижении нового минимума
      min := bit; dayMin := i;
      max := bit; dayMax := i;
    end
    else if bit > max then begin
      // переприсвоим максимум в буфере при достижении
      // нового максимума
      max := bit; dayMax := i;
    end;
    if max - min > profit then begin
      //запишем значение из буфера если выгода в нем больше
      profit := max - min;
      dayBuy := dayMin;
      daySell := dayMax;
    end;
  end;
  write( profit ); //выведем максимальную выгоду
  //выведем дни когда нужно купить и продать (если есть выгода)
  if profit > 0 then
     writeln(' ', dayBuy, ' ', daySell);
end.
```

Решение **Е. Джобса** (язык Python).

Описание переменных:

payDay – номер дня, когда выгодно купить биткоин

soldDay – номер дня, когда выгодно продать биткоин

minDay – номер дня с наименьшей ценой ДО текущего считанного значения

cash – максимальная прибыль

minPrice – минимальная цена ДО обрабатываемого дня

Изначально считаем, что выгодно продать и купить в первый день с нулевой прибылью, соответственно день с минимальной ценой –первый, минимальная цена по итогу – первое считанное значение курса биткоина.

Обрабатываем значения со второго дня (переменная **i** – номер обрабатываемого дня). **x** – курс в **i**-тый день. Если разница между минимальным значением цены ДО **i**-ого дня и курса в **i**-тый день больше текущей максимальной прибыли, то

- перезаписываем значение максимальной прибыли
- день покупки – день с минимальной ценой до **i**-того дня
- день продажи – текущий день (**i**)

Если курс в **i**-ый день меньше, чем предыдущее значение минимума, сохраняем **i**-тый день, как день с минимальным значением курса. Изменяем значение минимальной цены закупки.

```
n = int(input())

payDay, soldDay, minDay, cash = 1, 1, 1, 0
minPrice = int(input())

for i in range(2, n+1):
  x = int(input())
  if x - minPrice > cash:
    cash = x - minPrice
    payDay, soldDay = minDay, i
  if x < minPrice:
    minPrice, minDay = x, i

print(cash, end=' ')
if cash != 0:
  print(payDay, soldDay)
```

90) **(А. Жуков)**

Задача Б. Сразу отметим, что жадная стратегия основанная на подсчете количества отрицательных и отбрасывании элементов слева или справа (если отрицательных нечетное количество) не проходит, т.к. в последовательности могут встречаться нули.

В этой задаче, как и в предыдущих можно использовать схожую идею, слегка дополнив ее: для **i**-го элемента можно получить наибольшее произведение либо из наибольшего положительного на предыдущем шаге (если **i**-е число положительное), либо из наименьшего (если **i**-е число отрицательное).

Решение на Паскале ABC.NET:

```
var dp_min, dp_max, mx, i, n, x, t: integer;
begin
  readln(n);
  readln(x);
  dp_min := x; dp_max := x; mx := x;
  for i:=2 to n do begin
    readln(x);
    // сохраняем новое значение минимума
    t := min(dp_min*x, min(dp_max*x, x));
    // считаем максимальное
    dp_max := max(dp_min*x, max(dp_max*x, x));
    // восстанавливаем минимальное
    dp_min := t;
    // ищем максимальное
    mx := max(mx, dp_max);
  end;
  writeln(mx);
end.
```

Решение на GCC 4.6.3:

```
#include <iostream>
using namespace std;
int main() {
  int dp_min, dp_max, mx, i, n, x;
```

```
  cin >> n >> x;
  dp_min = dp_max = mx = x;
  for(i=1; i<n; ++i) {
    cin >> x;
    int t = min(dp_min*x, min(dp_max*x, x));
    dp_max = max(dp_min*x, max(dp_max*x, x));
    dp_min = t;
    mx = max(mx, dp_max);
  }
  cout << mx;
  return 0;
}
```

Решение на Python:

```
n = int(input())
dp_min = dp_max = mx = x = int(input())
for i in range(n-1):
  x = int(input())
  t = min(dp_min*x, dp_max*x, x)
  dp_max = max(dp_min*x, dp_max*x, x)
  dp_min = t
  mx = max(mx, dp_max)
print(mx)
```

91) **(Муфаззалов Д. Ф., г. Уфа)**

Задача A. Сохраним количество коробок в каждом контейнере в массив. Будем распределять каждую из новых коробок в тот контейнер, в котором в данный момент находится наименьшее их количество. Тогда ответом будет количество коробок в самом заполненном контейнере после того, как все новые коробки будут распределены.

Решение на Паскале ABC.NET:

```
var
  n, j, pos, m, i, ans: longint;
  a: array[1..100000] of longint;
begin
  readln(n, m);
  for i := 1 to n do readln(a[i]);
  for i := 1 to m do begin
    pos := 1;
    for j := 2 to n do
      if a[pos] > a[j] then
         pos := j;
    a[pos] := a[pos] + 1;
  end;
  ans := 0;
  for i := 1 to n do
    ans := max(ans, a[i]);
  write(ans);
end.
```

Решение на GCC 4.6.3:

```
#include <iostream>
using namespace std;
int main()
{
```

```
    int n, j, pos, m, i=0, a[100000], ans=0;
    cin >> n >> m;
    for (i = 0; i < n; i++)
      cin >> a[i];
    for(i = 0; i < m; i++) {
      pos = 0;
      for(j = 1; j < n; j++)
        if(a[pos] > a[j])
          pos = j;
      a[pos]++;
      }
    for (i = 0; i < n; i++)
      ans = max(ans, a[i]);
    cout << ans;
    return 0;
}
```

Можно изменить выбор контейнера для заполнения. Будем распределять каждую из новых коробок в контейнеры последовательно и циклично, начиная с первого. При этом нужно учесть, что нельзя добавлять коробку в наиболее загруженный контейнер, исключая случай, когда все контейнеры заполнены одинаково.

Так как проход по всем элементам массива для проверки того, заполнены ли контейнеры одинаково, приведет к дополнительному вложенному циклу, что нежелательно, будем осуществлять такую проверку с помощью суммы коробок во всех контейнерах. Если эта сумма равна произведению количества контейнеров на количество коробок в наиболее загруженном из них, то добавить коробку в текущий контейнер можно.

Решение на Паскале ABC.NET:

```
var
  n, j, pos, m, i, ans, s: longint;
  a: array[1..100000] of longint;
begin
  ans := 0;
  readln(n, m);
  s := 0;
  for i := 1 to n do begin
    readln(a[i]);
    s := s + a[i];
    ans := max(ans, a[i]);
  end;
  i := 1;
  while m > 0 do begin
    if (a[i] <> ans) or (s = n * ans) then begin
      a[i] := a[i] + 1;
      m := m - 1;
      s := s + 1;
      ans := max(a[i], ans);
    end;
    i := i mod n + 1; // циклическое перемещение по массиву
  end;
  write(ans);
end.
```

Решение на GCC 4.6.3:

```
#include <iostream>
using namespace std;
int main()
{
  int n, aMax = 0, i = 0, a[100000], ans = 0, s = 0;
  cin >> n >> m;
  for (i = 0; i < n; i++ ) {
    cin >> a[i];
    aMax = max(a[i], ans);
    s += a[i];
    }
  for(i = 0; m > 0; i = (i+1)%n) // циклическое перемещение
                                 // по массиву
   if( a[i] != aMax ||  s == n*aMax) {
    a[i]++;
    m--;
    s++;
    aMax = max(a[i], aMax);
    }
  cout << aMax;
  return 0;
}
```

Сложность обоих этих решений по времени в худшем случае составляет $O(N \cdot M)$, кроме того, они требуют дополнительно $O(N)$ памяти, так как все входные данные сохраняются.
Оптимизируем решение.
В первую очередь определим значение *amax* – количество коробок в самом заполненном контейнере до распределения новых грузов. Доведем количество коробок во всех контейнерах до *amax*. Для этого в контейнеры нужно будет добавить количество коробок, равное $amax - a_i$, i=1, 2, …, N. Общее количество добавляемых коробок вычислим по формуле

$$(amax\text{-}a_1) + (amax\text{-}a_2) + \ldots + (amax\text{-}a_N) = amax \cdot N - S,$$

где S – количество коробок во всех контейнерах до распределения. После этого количество нераспределенных коробок будет равно значению

$$K = M - (amax \cdot N - S).$$

Заметим, что если количество коробок, требуемых для равномерного заполнения контейнеров, больше M, то значение K будет отрицательным, поэтому количество нераспределенных коробок, верное для всех вариантов входных данных найдем по формуле

$$L = \max(0, M - amax \cdot N + S).$$

Далее возможны два случая:

1) количество коробок L кратно количеству контейнеров, тогда распределим оставшиеся коробки равномерно; ответ найдем по формуле $amax + L/N$;
2) количество коробок L не кратно количеству контейнеров, тогда распределим коробки в количестве $[L/N]$, где $[x]$ – целая часть от числа x, равномерно по всем контейнерам, а оставшиеся после этого коробки распределим по одной среди любых $Z = \text{mod}(L, N)$ контейнеров, где $\text{mod}(L, N)$ – остаток от деления числа L на число N. Во всех этих Z контейнерах количество коробок будет одинаковым и максимальным, поэтому ответ найдем по формуле $amax + L/N + 1$.

В данном решении для вычисления ответа требуются лишь суммарное количество коробок во всех контейнерах до распределения и максимальное количество коробок по всем

контейнерам. Эти значения можно определить во время считывания данных, и их запоминание не требуется. Поэтому сложность по времени составляет $O(N)$, а количество памяти не зависит от входных данных.

Решение на Паскале ABC.NET:

```
var
  aMax, n, m, i, ans, s, a: longint;
begin
  aMax := 0;
  readln(n, m);
  s := 0;
  for i := 1 to n do begin
    readln(a);
    s := s + a;
    aMax := max(aMax, a);
  end;
  m := max(0, m - aMax * n + s);
  ans := aMax + (m div n);
  if m mod n > 0 then
     ans := ans + 1;
  write(ans);
end.
```

Решение на GCC 4.6.3:

```
#include <iostream>
using namespace std;
int main()
{
  int n, m, s = 0, i = 0, a, aMax = 0;
  cin >> n >> m;
  for(i = 0; i < n; i++){
    cin >> a;
    s += a;
    aMax = max(aMax, a);
    }
  m = max(0, m-aMax*n+s);
  aMax += m/n;
  if( m % n > 0 ) aMax++;
  cout << aMax;
  return 0;
}
```

Решения на языке Python (Е. Джобс):

Описание переменных:

sum – общее количество всех коробок (доставленных и находившихся на складе);

maxBox – наибольшее количество коробок в контейнерах до распределения прибывшего груза;

mink – минимальная заполненность максимально заполненного контейнера при равномерном распределении **sum** коробок по **n** контейнерам

Алгоритм.

1) Считаем сумму коробок в изначальной партии и привезенных контейнерах:

   ```
   sum += x
   ```

   и находим максимальную заполненность контейнеров

   ```
   if maxBox < x: maxBox = x
   ```

2) Определяем сколько будет коробок в каждом контейнере, если мы равномерно распределим коробки:
   mink = sum // n
3) Если после равномерного распределения остались нераспределенные коробки
   if sum % n > 0:
   то минимальное допустимое количество будет на 1 больше, чем в каждом контейнере после распределения: **mink += 1**, так как нужно будет доложить по одной коробке в **sum % n** контейнеров (**sum % n < n**).
4) Если после равномерного распределения оказалось, что в каждом контейнере должно быть меньше коробок, чем количество коробок в максимально заполненном прибывшем контейнере
   if maxBox > mink:
   тогда это количество и является ответом (**answer = maxBox**), иначе ответом является максимальное количество коробок при равномерном распределении (**answer = mink**)

```
n, m = map(int, input().split())
sum, maxBox = m, 0
for i in range(n):
  x = int(input())
  # maxBox = max(x, maxBox)
  if maxBox < x:
    maxBox = x
  sum = sum + x

mink = sum // n
if sum % n > 0:
  answer = mink + 1

# answer = max(maxBox, mink)
if maxBox > mink:
  answer = maxBox
else:
  answer = mink
print(answer)
```

92) **(Ю. А. Кондрашин)**

Для решения задачи А необходимо выполнить следующие действия:

1) Установить количество пропускаемых значений $K = 6$.
2) Считать N – общее количество показаний прибора.
3) Считать N строк – показание прибора в массив $Buf[]$.
4) Задать минимальное значение МР равным -1. (Это значение также будет использовано как признак отсутствия результата)
5) **Для** всех значений I от 1 до $N - K$ (первый множитель)
6) **Для** всех значений J от $I + K$ до N (второй множитель)
7) **Если** произведение $Buf[I] * Buf[J]$ чётное **И** (произведение $Buf[I] * Buf[J] < MP$ минимального **ИЛИ** минимальное значение отсутствует $MP < 0$)
8) **То** минимальное значение MP = произведение $Buf[I] * Buf[J]$
9) Показать результат – минимальное значение MP.

Примечания.

1) На 4 шаге в качестве минимального значения можно взять максимально возможное значение + 1. Тогда на 7 шаге можно убрать часть условия (после ИЛИ), а перед

выводом результата на 9 шаге проверить его наличие и вывести -1 в случае отсутствия.

2) На 5 шаге исключаем из просмотра последние K значений, т.к. у них нет множителя.
3) На 6 шаге исключаем из просмотра элементы, меньшие I или стоящие недалеко от I. Это позволит перебирать только пары в которых $I < J + K$, что сократит количество операций вдвое (даже чуть больше).
4) Для проверки чётности проверяем остаток от деления на 2.

Текст программы на языке Python:

```
buf = []  # место для хранения значений
k = 6  # отступ от текущего значения
mp = -1  # минимальное значение в последовательности
num = int(input())  # считать количество значений
for i in range(num):
    buf.append(int(input()))
for i in range(num - k):
    for j in range(i + k, num):
        if ((buf[i] * buf[j] % 2 == 0) and
           ((buf[i] * buf[j] < mp) or (mp < 0))):
            # если произведение чётное и первое
            # или меньше минимального
            mp = buf[i] * buf[j]
print(mp)
```

Для решения второй задачи проанализируем представленный выше алгоритм.

Для минимального чётного произведения надо чтобы из пары чисел одно было чётным, а второе минимальным или одно – минимальное чётное, а второе любое. Для каждого элемента с индексом I существует один элемент с минимальным значением, расположенный не ближе чем за K позиций от этого индекса.

Предположим, что нам известно значение минимального элемента, расположенного не ранее чем на K позиций левее просматриваемого элемента с индексом I. При переходе к позиции $I + 1$ значением минимального элемента будет выбираться из пары: минимальный элемент для позиции I и элемент исходного массива с индексом $I + 1$ - K. Эту операцию можно выполнить за 1 действие, что позволит убрать вложенный цикл.

Для сокращения используемой памяти можно заметить, что в каждый момент времени нам необходимо только $K + 1$ элементов, расположенных левее просматриваемого элемента. Если использовать кольцевую очередь, то можно сократить и используемую для хранения элементов память до K вместо N.

Текст программы на языке Python:

```
buf = []  # место для хранения значений
k = 6  # отступ от текущего значения
mp = 1002 * 1002  # минимальное значение в последовательности
cur = 0  # граница кольцевой очереди
min_a = mp  # для минимальных значений
min_e = mp  # для минимальных чётных значений
num = int(input())  # считать количество значений
for i in range(k):  # заполняем очередь
    buf.append(int(input()))
for i in range(k, num):  # просматриваем оставшиеся элементы
    temp = int(input())
    if buf[cur] < min_a:
        min_a = buf[cur]  # обновляем минимальный элемент
    if (buf[cur] % 2 == 0) and (buf[cur] < min_e):
```

```
            min_e = buf[cur]  # обновляем минимальный чётный элемент
        if temp % 2 == 0:  # чётный элемент
            if temp * min_a < mp:  # умножаем на минимальный
                mp = temp * min_a
        else:  # temp % 2 == 1:  # нечётный элемент
            if temp * min_e < mp:  # умножаем на минимальный чётный
                mp = temp * min_e
        buf[cur] = temp  # обновляем первый элемент очереди и
        cur = (cur + 1) % k  # переходим к следующему элементу
if mp == 1002 * 1002:  # не нашли :-(
    mp = -1
print(mp)
```

93) **(В.Н. Бабий, Челябинск)**

Пример решения на 2 балла.

Введём все элементы в массив, затем перебором проверим каждую пару элементов в этом массиве.

Рассмотрим пример. Введем в массив 10 элементов последовательности:

1	2	3	4	5	6	7	8	9	10
17	11	15	24	36	47	13	9	44	53

Номера элементов должны отличаться не менее чем на 6, то есть 1-й элемент массива мы можем умножать только на 7, 8, 9 и 10 элементы:

1	2	3	4	5	6	7	8	9	10
17	11	15	24	36	47	13	9	44	53

2-й элемент на 8, 9, и 10:

1	2	3	4	5	6	7	8	9	10
17	11	15	24	36	47	13	9	44	53

3-й элемент на 9 и 10:

1	2	3	4	5	6	7	8	9	10
17	11	15	24	36	47	13	9	44	53

4-й элемент на 10:

1	2	3	4	5	6	7	8	9	10
17	11	15	24	36	47	13	9	44	53

Для организации перебора потребуется вложенный цикл. Внешний цикл будет брать первый элемент пары с индексом i, вложенный цикл будет перебирать оставшиеся возможные элементы пары с индексом от $i+1$ до последнего (N).

```
var
   a: array[1..10000] of integer;
   i, j, n, R: integer;
begin
   readln(n);
   for i:=1 to n do
       readln(a[i]);
   R := 0;
   for i:=1 to n-6 do
      for j:=i+6 to n do
          if ((a[i]*a[j]) mod 10 = 3) then
              R := R + 1;
   writeln(R);
end.
```

Пример решения №1 на 4 балла (динамическое программирование):

Для начала нужно понять, произведение каких чисел будет заканчиваться на 3. Произведение чисел заканчивается на 3 в том случае, если произведение последних цифр этих чисел заканчивается на 3. К примеру, 27*89 = 2403, т.к. произведение последних цифр этих чисел заканчивается на 3 (7*9=63).

В десятичной системе счисления существует всего два случая, когда произведение цифр заканчивается на 3:

1*3 = 3

7*9 = 3

То есть произведение чисел будет заканчиваться на 3, если последняя цифра одного множителя равна 1, а другого – 3, либо если последняя цифра одного из множителя равна 7, а другая – 9.

Теперь рассмотрим пример последовательности:

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

Для решения задачи с помощью динамического программирования нужно понять, какие данные нам нужны с предыдущих шагов для текущего шага. Рассмотрим шаг 10 (ввод числа 59):

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

Так как разница в номерах элементов должна быть не менее 6, то 10-й элемент последовательности мы можем умножать на первые четыре элемента:

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

Так как число 59 заканчивается на 9, то для него существует два множителя на предыдущих шагах (17 и 27), и, соответственно, на данном шаге количество пар для этого числа будет равно количеству чисел, заканчивающихся на 7 на предыдущих шагах, с разницей в номерах элементов не менее, чем в 6.

Если элемент последовательности заканчивается на 7, то для него количество пар будет равно количеству элементов на предыдущих шагах, которые заканчиваются на 9. Такая же ситуация с элементами, заканчивающимися на 1 или 3.

Для решения задачи на каждом шаге будем хранить количество чисел, заканчивающихся на 1, 3, 7, 9 на предыдущих шагах, а затем, в случае, если текущее введенное число заканчивается на 1, 3, 7 или 9, будем прибавлять к общему количеству пар подходящее количество чисел с предыдущих шагов. К примеру, если на текущем шаге мы ввели число 67, то к общему количеству прибавляем количество чисел на предыдущих шагах, которые заканчиваются на 9.

Так как разница в номерах элементов последовательности должна быть не менее 6, используем массив из 7-ми элементов. Между первым и седьмым элементов разница между номерами ровно 6. Сначала введём в массив первые шесть чисел последовательности (на данном этапе мы еще ничего не можем проверять, т.к. разница в номерах не достигла 6). После этого на каждом шаге будем вводить 7-й элемент массива, проверять, на какую цифру заканчивается 1-й элемент (и если, к примеру, он заканчивается на 3, то количество элементов на предыдущих шагах, заканчивающихся на 3, увеличиваем на 1), проверять, на какую цифру заканчивается ввёденный нами 7-й элемент, и если эта цифра – 1, 3, 7 или 9, прибавлять к общему количеству пар соответствующее число элементов с предыдущих шагов. После этого сдвигаем массив, и на следующем шаге вводим новый элемент последовательности.

```
var
   a: array[1..7] of integer;
   i, j, R, n, k1, k3, k7, k9: integer;
begin
   readln(n);
   for i:=1 to 6 do
      readln(a[i]);
   k1 := 0; k3 := 0; k7 := 0; k9 := 0; R := 0;
   for i:=7 to n do
   begin
      readln(a[7]);
      if (a[1] mod 10 = 1) then k1 := k1 + 1;
      if (a[1] mod 10 = 3) then k3 := k3 + 1;
      if (a[1] mod 10 = 7) then k7 := k7 + 1;
      if (a[1] mod 10 = 9) then k9 := k9 + 1;
```

```
      if (a[7] mod 10 = 1) then R := R + k3;
      if (a[7] mod 10 = 3) then R := R + k1;
      if (a[7] mod 10 = 7) then R := R + k9;
      if (a[7] mod 10 = 9) then R := R + k7;
      for j:=1 to 6 do
         a[j] := a[j+1];
   end;
   writeln(R);
end.
```

Пример решения №2 на 4 балла (комбинаторика):

Для решения определим общее количество пар, в которых произведение элементов заканчивается на 3, и количество пар, в которых произведение элементов заканчивается на 3, а разница между номерами меньше 6. Из общего количества пар вычтем то, что нам не нужно (количество пар с разницей в номерах менее 6), и получим количество пар, с разницей в номерах не менее 6.

Для начала разберемся, как найти общее количество пар, в которых произведение элементов заканчивается на 3.

Произведение чисел заканчивается на 3, если одно из чисел заканчивается на 1, а другое на 3, либо если одно заканчивается на 7, а другое на 9.

Рассмотрим пример. Предположим, в программу были введены пять чисел: три из них заканчиваются на 7, а два – на 9:

27 59
47 29
17

У каждого из трех чисел, заканчивающихся на 7, есть две пары чисел, заканчивающихся на 9. То есть количество пар будет равно 3*2=6, где 3 – количество чисел, заканчивающихся на 7, а 2 – количество чисел, заканчивающихся на 9. Такая же ситуация с парами, в которых одно число заканчивается на 1, а другое – на 3.

Решение данной задачи выглядит так:

```
var
   i, R, n, k1, k3, k7, k9, x: integer;
begin
   readln(n);
   k1 := 0; k3 := 0; k7 := 0; k9 := 0;
   for i:=1 to n do
   begin
      readln(x);
      if (x mod 10 = 1) then k1 := k1 + 1;
      if (x mod 10 = 3) then k3 := k3 + 1;
      if (x mod 10 = 7) then k7 := k7 + 1;
      if (x mod 10 = 9) then k9 := k9 + 1;
   end;
   R := k1*k3 + k7*k9;
   writeln(R);
end.
```

Теперь определим количество пар элементов последовательности, в которых произведение элементов оканчивается на 3, но разница между номерами элементов составляет менее 6.

Рассмотрим пример последовательности:

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

Одновременно в программе будут храниться только шесть элементов последовательности. Для этого будем использовать массив из шести элементов. В таком массиве разница между номерами элементов менее 6.

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

После ввода первых шести элементов каждый раз будем сдвигать массив и вводить следующий элемент последовательности, пока не дойдем до конца:

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

Теперь можно предложить с виду правильное, но для данной задачи неверное решение: каждый раз перебирать все возможные пары внутри массива и считать количество пар, в которых произведение оканчивается на 3:

```
readln(n);
for i:=1 to 5 do
   readln(a[i]);
R := 0;
for i:=6 to n do
begin
   readln(a[6]);
   for j:=1 to 5 do
      for k:=j+1 to 6 do
         if ((a[j]*a[k]) mod 10 = 3) then
            R := R + 1;
   for j:=1 to 5 do
      a[j] := a[j+1];
end;
```

Однако такое решение не верно, потому что в этом случае мы будем проверять одни и те же пары несколько раз. Для наглядности, в данном примере

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

Сначала будут проверены пары (17, 11), (17, 27), (17, 24), (17, 36), (17,47), (11,27), (11, 24), (11, 36), (11, 47), (27, 24), (27, 36), (27, 47), (24, 36), (24, 47), (36,47), а на следующем шаге

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

Будут проверены пары (11,27), (11, 24), (11, 36), (11, 47), (11, 13), (27, 24), (27, 36), (27, 47), (27, 13), (24, 36), (24, 47), (24, 13), (36,47), (36, 13), (47, 13).

Получается, что пары (11,27), (11, 24), (11, 36), (11, 47), (27, 24), (27, 36), (27, 47), (24, 36), (24, 47), (36,47) будут проверены дважды, что недопустимо, т.к. мы находим количество. При этом данное решение будет работать при поиске максимума/минимума (прим.: максимальное произведение двух элементов с разницей во времени менее 6), правда из-за перебора эффективность решения снижается.

Для решения задачи исключим все повторения пар. Для этого сначала проверим пары для первых пяти элементов массива:

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

(17, 11), (17, 27), (17, 24), (17, 36), (11,27), (11, 24), (11, 36), (27, 24), (27, 36), (24, 36)

А затем будем проверять пары, в которых первый элемент – один из первых пяти элементов массива, а второй – последний элемент массива:

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

(17,47), (11, 47), (27, 47), (24, 47), (36,47)

То же самое делаем на следующем шаге:

1	2	3	4	5	6	7	8	9	10
17	11	27	24	36	47	13	9	44	59

(11, 13), (27, 13), (24, 13), (36, 13), (47, 13).

Таким образом, мы проверяем все возможные пары, при этом каждую пару проверяем ровно один раз. Решение:

```
var
   a: array[1..6] of integer;
   i, j, R, n: integer;
begin
   readln(n);
   for i:=1 to 5 do
      readln(a[i]);
   R := 0;
   for i:=1 to 4 do
      for j:=i+1 to 5 do
          if ((a[i]*a[j]) mod 10 = 3) then
             R := R + 1;
   for i:=6 to n do
   begin
      readln(a[6]);
      for j:=1 to 5 do
         if ((a[j]*a[6]) mod 10 = 3) then
             R := R + 1;
      for j:=1 to 5 do
         a[j] := a[j+1];
   end;
   writeln(R);
end.
```

Теперь объединим две программы в одну:

```
var
   a: array[1..6] of integer;
   i, j, R, n, k1, k3, k7, k9: integer;
begin
   readln(n);
   k1 := 0; k3 := 0; k7 := 0; k9 := 0; R := 0;
   for i:=1 to 5 do
   begin
      readln(a[i]);
      if (a[i] mod 10 = 1) then k1 := k1 + 1;
      if (a[i] mod 10 = 3) then k3 := k3 + 1;
      if (a[i] mod 10 = 7) then k7 := k7 + 1;
      if (a[i] mod 10 = 9) then k9 := k9 + 1;
   end;
   for i:=1 to 4 do
      for j:=i+1 to 5 do
          if ((a[i]*a[j]) mod 10 = 3) then
             R := R + 1;
```

```
   for i:=6 to n do
   begin
      readln(a[6]);
      if (a[6] mod 10 = 1) then k1 := k1 + 1;
      if (a[6] mod 10 = 3) then k3 := k3 + 1;
      if (a[6] mod 10 = 7) then k7 := k7 + 1;
      if (a[6] mod 10 = 9) then k9 := k9 + 1;
      for j:=1 to 5 do
         if ((a[j]*a[6]) mod 10 = 3) then
             R := R + 1;
      for j:=1 to 5 do
         a[j] := a[j+1];
   end;
   R := (k1*k3 + k7*k9) - R;
   writeln(R);
end.
```

94) (**К.М. Багдасарян, Ковров**)

Задача A.

Решение (Python).

```
n = int(input())
d = [ int(input()) for i in range(n)]
mx = -1001
for i in range(n-3):
  for j in range(i+1,n-2):
    for k in range(j+1,n-1):
      for m in range(k+1,n):
        s = d[i] + d[j] + d[k] + d[m]
        if( s > mx and s % 2 == 0 ):
          mx = s;
if( mx > -1001 ):
  print(mx)
else:
  print('Не найдено')
```

Задача B.

Рассмотрим следующие ситуации

1) Сумма всех положительных чисел чётная. В этом случае данная сумма является искомой.
2) Сумма всех положительных чисел нечётная. В этом случае необходимо вычесть минимальное нечетное положительное число или прибавить максимальное отрицательное нечётное число (если они имеются).
3) Имеется единственное положительное число, оно нечётно и при этом нет других нечётных чисел. Переходим к п. 4.
4) Сравниваем максимальное отрицательное чётное число с суммой 2-х максимальных отрицательных нечётных чисел. Большее из них и будет искомой суммой.

Решение (Python).

```
n=int(input())
max_otr1 = -201       # Максимальное отрицательное нечетное число
max_otr2 = -201       # Второе максимальное отрицательное
                      # нечетное число
max_otr_ch = -201     # Максимальное отрицательное четное число.
min_pol_nch = 201     # Минимальное положительное нечетное число


summ = 0              # Сумма положительных значений
```

```
for i in range(n):
    m=int(input())
    if m>0:
        if (m % 2 == 1 and m < min_pol_nch):
            min_pol_nch = m
        summ += m
    else:
        if m % 2 == 0:
            if m > max_otr_ch:
                max_otr_ch = m
        elif m > max_otr1:
            max_otr2 =  max_otr1
            max_otr1 = m
        elif m > max_otr2:
                max_otr2 = m

if summ > 0:
    if summ % 2 == 0:
        s = summ           # Искомая сумма
    elif summ == min_pol_nch:
        s = max(max_otr_ch, max_otr1 + min_pol_nch)
    elif min_pol_nch < -(max_otr1):
        s = summ - min_pol_nch
    else:
        if max_otr1 > -101:
            s = summ + max_otr1
elif max_otr1 > -101 and max_otr2 > -101 \
     and max_otr1 + max_otr2 > max_otr_ch:
    s = max_otr1 + max_otr2
else:
    s = max_otr_ch

print(s)
```

Решение (C++).

```
#include <iostream>
using namespace std;
int main(){
int n,m;
int max_otr1 = -201;     // Максимальное отрицательное нечетное
                         // число
int max_otr2 = -201;     // Второе максимальное отрицательное
                         // нечетное число
int max_otr_ch = -201;   // Максимальное отрицательное четное
                         // число.
int min_pol_nch = 201;   // Минимальное положительное нечетное
                         // число
int summ = 0;            // Сумма положительных значений
int s;                   // Искомая сумма
cin >> n;
for( int i = 0; i < n; i++ ) {
 cin >> m;
 if( m > 0 ){
```

```
   if( m % 2 != 0 and m < min_pol_nch )
     min_pol_nch = m;
   summ += m;
   }
 else {
   if( m % 2 == 0 ) {
     if( m > max_otr_ch )
       max_otr_ch = m;
     }
   else {
    if( m > max_otr1 ) {
      max_otr2 =  max_otr1;
      max_otr1 = m;
      }
    else
     if( m > max_otr2 ) max_otr2 = m;
    }
  }
}

if( summ > 0 ) {
  if( summ % 2 == 0 ) s = summ;
  else
   if (summ == min_pol_nch)
    s = max(max_otr_ch, max_otr1 + min_pol_nch);
   else
    if( min_pol_nch < -(max_otr1) )
      s = summ - min_pol_nch;
    else
      if( max_otr1 > -101 )
        s = summ + max_otr1;
  }

else
  if( max_otr1 > -101 and max_otr2 > -101 and
      max_otr1 + max_otr2 > max_otr_ch)
    s = max_otr1 + max_otr2;
  else s = max_otr_ch;

cout << s;
}
```

Решение (Паскаль).

```
var n, i, m, s, max_otr1, max_otr2,
    max_otr_ch, min_pol_nch, summ: integer;
begin
 readln(n);
 max_otr1:= -201;       // Максимальное отрицательное нечетное
                        // число
 max_otr2:= -201;       // Второе максимальное отрицательное
                        // нечетное число.
 max_otr_ch:= -201;     // Максимальное отрицательное четное
                        // число.
```

```
 min_pol_nch:= 201;     // Минимальное положительное нечетное
                        // число
 summ:= 0;              // Сумма положительных значений

 for i:=1 to n do begin
   readln( m );
   if( m > 0 ) then begin
     if( m mod 2 <> 0 ) and ( m < min_pol_nch ) then
       min_pol_nch:= m;
     summ:= summ + m;
   end
   else begin
     if( m mod 2 = 0 ) then begin
       if m > max_otr_ch then
         max_otr_ch:= m;
     end
     else if( m > max_otr1 ) then begin
       max_otr2:= max_otr1;
       max_otr1:= m;
     end
     else
       if( m > max_otr2 ) then max_otr2:= m;
    end;
  end;

  if( summ > 0 ) then begin
   if( summ mod 2 = 0 ) then
     s:= summ                  // Искомая сумма
   else if (summ = min_pol_nch) then
     s:= max(max_otr_ch, max_otr1 + min_pol_nch)
   else if( min_pol_nch < -(max_otr1) )then
     s:= summ - min_pol_nch
   else if( max_otr1 > -101 ) then s:= summ + max_otr1;
 end

 else
   if( max_otr1 > -101 ) and ( max_otr2 > -101 ) and
     ( max_otr1 + max_otr2 > max_otr_ch ) then
     s:= max_otr1 + max_otr2
   else s:= max_otr_ch;

 writeln(s);
end.
```

Решение (Н.И. Герасименко, C++, динамическое программирование).

```
#include <iostream>
#include <bits/stdc++.h>

using namespace std;

/**
 *  max3 ищет максимум из трех аргументов
 */
int max3(int a,int b,int c)
```

```
{
    int t;
    t = a > b ? a: b;
    return t > c ? t: c;
}

int main()
{
    int x,
        s,      //искомая сумма
        sd,     //динамическая сумма
        sr,     //рабочая ячейка(для удобства отладки)
        n,      //кол-во чисел
        i;

    cin >> n;

    // инициализирую обе суммы начальным значением
    cin >> x;
    if( !(x & 1) )
        s = x;
    else
        s = INT_MIN;
    sd = x;

    // в цикле реализую динамический алгоритм поиска
    for( i=1; i < n; ++i )
      {
      cin >> x;
      sr = sd + x;           //динамическая сумма -
      sd = max3(sd, sr, x); //максимум из прежней, новой и x

        // новое s ищем как наибольшее
        // из ЧЕТНЫХ sd, sr, x)

      if( !(sd & 1) && sd > s )
          s = sd;
      if( !(sr & 1) && sr > s )
          s = sr;
      if( !(x & 1) && x > s )
          s = x;
      }
    cout << s;
}
```

95) (**О.Л. Дуркин, Сыктывкар**)

Задача A.

Заметим, что если между числами, которые можно поставить в пару, должны находиться еще как минимум 4 числа, то разница между номерами чисел в последовательности должна быть 5 и более. Для организации перебора потребуется вложенный цикл. Внешний цикл будет брать первый элемент пары с индексом i (до N–5), вложенный цикл будет перебирать оставшиеся возможные элементы пары с индексом от i + 5 до последнего (N).

Решение (C++).

```
#include <iostream.h>
using namespace std;
int main()
{
  int a[1000];
  int n, k = 0, i, j;
  cin >> n;
  for(i=0;i<n;i++)
    cin >> a[i];
  for( i = 0; i < n-5; i++)
    for( j = i + 5; j < n; j++ )
      if( a[i]*a[j] % 10 == 1 )
        k++;
  cout << k;
}
```

Задача B.

Произведение двух чисел оканчивается на 1 только в трех случаях:

- первый и второй множители одновременно оканчиваются на 1;
- первый и второй множители одновременно оканчиваются на 9;
- один множитель оканчивается на 3, а второй на 7.

Для решения задачи будем считать количество чисел, которые оканчиваются на 1, 3, 7 и 9. При этом необходимо учитывать расстояние между этими числами. Для этого создадим массив из 6 элементов.

Решение (C++).

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  int a[6];
  int n, n1 = 0, n3 = 0, n7 = 0, n9 = 0,
      k = 0, i, j;
  cin >> n;
  for( i = 0; i < 5; i++ )
    cin >> a[i];
  for( i = 5; i < n; i++ )
    {
    cin >> a[5];
    if( a[0]%10 == 1) n1++;
    if( a[0]%10 == 3) n3++;
    if( a[0]%10 == 7) n7++;
    if( a[0]%10 == 9) n9++;
    if( a[5]%10 == 1) k += n1;
    if( a[5]%10 == 3) k += n7;
    if( a[5]%10 == 7) k += n3;
    if( a[5]%10 == 9) k += n9;
    for( j = 0; j < 5; j++ )
      a[j] = a[j+1];
    }
  cout << k;
  return 0;
}
```

(**Д.Ф. Муфаззалов, Уфа**)

Как было сказано выше, для нахождения искомого количества имеют значение лишь младшие цифры вводимых чисел, поэтому будем запоминать только их. В массиве на десять элементов будем запоминать, сколько раз встретилась каждая цифра. Начиная с первого введенного числа, для которого можно составить пару (оно поступает шестым), будем увеличивать переменную результата на количество раз, которое встретилось парное ему число к этому моменту. Зависимость парного числа от поступившего числа покажем в таблице:

Поступившее число	Парное число
1	9
3	7
7	3
9	1

Определим зависимость чисел из второго столбца от чисел из первого столбца. Так как строк в таблице четыре, предположим, что зависимость описывается кубическим выражением

$$ax^3+bx^2+cx+d \quad (1)$$

и найдем его коэффициенты. Для этого подставим каждое значение из первого столбца таблицы в выражение (1) и запишем соответствующее уравнение, приравняв это выражение к значению из второго столбца на этой же строчке:

$$\begin{aligned} a+b+c+d&=9,\\ 27a+9b+3c+d&=7,\\ 343a+49b+7c+d&=3,\\ 729a+81b+9c+d&=1. \end{aligned}$$

Решая эту систему, получим a=1/6, b=-15/6, c=65/6, d=-45/6. Поэтому искомое выражение имеет вид

$$(x^3-15x^2+65x-45)/6 \quad (2)$$

В первом столце таблицы нет числа 5, поэтому при поступлении этого числа результат не должен увеличиваться. Заметим, что значение выражения (1) от числа 5 равно пяти. Чтобы результат не изменялся, будем поддерживать значение элемента массива с индексом 5 равным нулю.

В первом столце таблицы нет и четных чисел, поэтому и в этом случае результат не должен увеличиваться. Чтобы результат не изменялся, при увеличении результата будем умножать увеличивающее слагаемое на 0, для четных чисел это остаток от деления на 2. Отметим, что для четных x, меньших 10, значение выражения (2) больше 0 и меньше 10, то есть выхода за пределы массива не произойдет.

Вместо сдивига элементов массива будем использовать очередь в виде кольца. Голова очереди перемещается.

```
#include <iostream>
using namespace std;
int main()
{
    int a[5];
    int n, d[10] = {0}, m, k = 0, i, j;
    cin >> n;
    for( i = 0; i < 5; i++ ){
      cin >> m;
      a[i] = m % 10; //запоминаем только младшую цифру
      }
    for(; i < n; i++) {
      j = i%5; //смещаем голову очереди
      d[a[j]] += a[j]!=5; // увеличиваем счетчик
      cin >> m;        //вводим очередное число
      m %= 10;    //выделяем младшую цифру
                  //формируем результат
      k += d[(m*m*m - 15*m*m + 65*m - 45)/6]*(m%2);
      a[j] = m;    //помещаем элемент в очередь
      }
    cout << k;
```

```
    return 0;
}
```

Заводить массив на 10 чисел не обязательно, ведь нужна информация только о четырех из них. Будем помещать количество чисел, оканчивающихся на цифры 1, 3, 7 и 9, в элементы массива с индексами 0, 1, 2 и 3 соответственно. Для определения такого индекса надо взять целую часть от частного от деления младшей цифры на 3.

Зависимость индекса элемента массива, который будет использоваться для увеличения результата, от поступившего числа покажем в таблице:

Поступившее число	Индекс элемента массива
1	0
3	2
7	1
9	3

По методике, описанной выше, найдем эту зависимость: $x*(2*x*x -9*x +11)/2$, где x - целая часть от деления младшей цифры поступившего числа на число 3. Заметим без доказательства, что можно использовать и формулу $((3- x)\%3+ x/3*3)$, где $a\%b$ – остаток от деления числа a на число b.

Для организации очереди удобно использовать STL контейнер queue.

```
#include <iostream>
#include <queue>
using namespace std;
int main()
{
    queue <int> a;
    int n, d[4] = {0}, m, k = 0, i, p;
    cin >> n;
    for(i = 0; i < 5; i++ )
    {
        cin >> m;
        a.push(m%10); // выделяем младшую цифру
    }
    for(; i < n; i++)
    {
        p=a.front();// число в голове очереди
        if (p%2 && p!=5) d[p/3]++;//увеличиваем счетчик
        a.pop();// удаляем число из очереди
        cin >> m; // вводим очередное число
        m%=10; // выделяем младшую цифру
        if( m%2 && m!=5 ) // только если поступило нужное число
            {
             p=m/3;
             k += d[p*(2*p*p - 9*p + 11)/2]; // увеличиваем результат
        //   k += d[(3-p)%3 + p/3*3]; // или такая формула
            }
        a.push(m); // помещаем число в очередь
    }
    cout << k;
    return 0;
}
```

(**Е. Джобс**)

Общее пояснение решения. Произведение двух чисел оканчивается на 1 в четырех случаях:

- первое и второе числа оканчиваются на 1;
- первое число оканчивается на 3, второе – на 7;

- первое число оканчивается на 7, второе - на 3;
- первое и второе числа оканчиваются на 9.

Описание структур для хранения данных.

a – массив с очередью из элементов, которые не могут быть обработаны с вводимым «хвостовым» значением. Так как в задании необходимо, чтобы между элементами, составляющими произведение, расстояние было не меньше 4, следовательно порядковый номер вновь вводимого элемента и первого в очереди должны отличаться на 5. Поэтому длина массива **a** равна 5 (от 0 до 4).

r – количество чисел оканчивающиеся на **i**, которые могут быть перемножены с вновь введенным. Индекс **i** указывает на последнюю цифру таких чисел.

k – индекс начала очереди для динамической очереди.

Описание алгоритма.

В массиве с очередью будем сохранять только последнюю цифру введенных чисел, то есть остаток от деления на 10. При вводе нового числа необходимо обработать первый (головной) элемент в очереди. Увеличив соответствующий значению этого элемента элемент массива **r**. Например, если значение первого элемента очереди – 5, увеличиваем **r[5]** на единицу.

У вновь веденного числа нас также интересует только последний разряд. Если введено число **x** оканчивающееся на 1 или 9, необходимо увеличить количество найденных пар на число, записанное в **r[1]** или **r[9]** соответственно (или **r[x]**). Если последняя цифра введённого числа 3 или 7, необходимо увеличить количество найденных пар на значение **r[7]** и **r[3]** соответственно (или **r[10-x]**).

Решение (Паскаль).

```
var k, i, N, x, count: integer;
a: array[0..4] of integer;
r: array[0..9] of integer;
begin
  readln(N);
  count := 0;
  for i:=0 to 4 do readln(a[i]);
  for i:=0 to 4 do a[i] := a[i] mod 10;
  for i:=0 to 9 do r[i] := 0;
  for i:=5 to N-1 do begin
    k := i mod 5;
    r[ a[k] ] := r[ a[k] ] + 1;
    readln(x);
    x := x mod 10;
    if(x=1)or(x=9) then count:=count + r[x]
    else if(x=3)or(x=7) then count:=count + r[10-x];
    a[k] := x;
  end;
  writeln(count);
end.
```

Решение (Python).

```
a = []
r = [0]*10
count = 0
N = int(input())
for i in range(5):
  a.append(int(input()) % 10)
for i in range(5, N):
```

```
  k = i % 5
  r[ a[k] ] += 1
  x = int(input()) % 10
  if x == 1 or x == 9:
    count += r[x]
  elif x == 3 or x == 7:
    count += r[10-x]
  a[k] = x
print(count)
```

(**Е. Джобс**)

Нам интересны пары чисел, у которых последняя цифра 1, 3, 7 или 9. Заметим, что остатки от деления чисел 1, 3, 7 и 9 на 5 равны 1, 3, 2 и 4 соответственно.

Также отметим, что 1, 3, 7 и 9, которые нас интересуют – нечетные числа, а чётные не влияют на результат. Поэтому в массиве **r** будем хранить количество нечётных чисел с остатками от 1 до 4.

Если введенное число оканчивается на 3 ил 7, необходимо добавить к количеству пар количество противоположных чисел (**r[5 - x % 5]** в Python), если на 1 или 9, то количество чисел с таким же остатком (**r[x % 5]** в Python).

Решение (Паскаль).

```
var k, i, N, x, count: integer;
    a: array[0..4] of integer;
    r: array[0..4] of integer;
begin
  readln(N);
  count := 0;
  for i:=0 to 4 do readln(a[i]);
  for i:=0 to 4 do a[i] := a[i] mod 10;
  for i:=0 to 4 do r[i] := 0;
  for i:=5 to N-1 do begin
    k := i mod 5;
    if a[k] mod 2 = 1 then
      r[ a[k] mod 5 ] := r[ a[k] mod 5 ] + 1;
    readln(x);
    x := x mod 10;
    if (x = 3)or(x = 7)    then
      count := count + r[5 - (x mod 5)]
    else if (x = 1) or (x = 9) then
    count := count + r[x mod 5];
    a[k] := x;
  end;
  writeln(count);
end.
```

Решение (Python).

```
a = []
r = [0]*5
N = int(input())
count = 0
for i in range(5):
    a.append( int(input()) % 10)
for i in range(5, N):
    k = i % 5
```

```
    if a[k] % 2 == 1:
        r[ a[k] % 5] += 1
    x = int(input()) % 10
    if x == 3 or x == 7:
        count += r[5 - x % 5]
    elif x == 1 or x == 9:
        count += r[x % 5]
    a[k] = x
print(count)
```

96) (**Д.Ф. Муфаззалов, Уфа**)

Задача А.

Заметим, что если между числами, которых можно поставить в пару, должны находиться как минимум 4 числа, то разница между номерами чисел в последовательности должна быть 5 и более. Для организации перебора потребуется вложенный цикл. Так как порядок чисел в паре значения не имеет, будем формировать только такие пары, в которых номер второго элемента больше номера первого элемента. Внешний цикл будет брать первый элемент пары с индексом i от первого до последнего, вложенный цикл будет перебирать оставшиеся возможные элементы пары с индексом от $i + 5$ до последнего (N). Перед перебором обнулим счетчик искомых пар. Если во время перебора сумма элементов пары кратна числу 3, будем увеличивать количество пар на число 1.

Решение (Паскаль).

```
var
  a: array[1..1000] of integer;
  n, i, j, ans: integer;
begin
  readln(n);
  ans := 0;
  for i := 1 to n do readln(a[i]);
  for i := 1 to n do
    for j := i + 5 to n do
      if (a[i] + a[j]) mod 3 = 0 then ans := ans + 1;
  writeln(ans);
end.
```

Задача Б.

Сумма двух натуральных чисел может быть кратной числу 3 только в следующих случаях:

1) остатки от деления обоих числе на число 3 равны нулю;
2) остаток от деления одного числа на три равен одному, остаток от деления другого числа на три равен двум.

Первое число, для которого возможно сформировать пару, имеет номер 5, поэтому все предыдущие числа будем сохранять в буфер (очередь) на пять элементов. Так как для определения ответа нужны только остатки, будем сохранять только их. В массиве на три элемента будем подсчитывать количество чисел, вышедших из очереди и имеющих соответствующий остаток. В момент поступления нового числа (его остатка), будем определеять, сколько пар может быть с ним составлено, и удалять один элемент из очереди.

Очередь оранизуем в виде кольца, голова очереди перемещается на один элемент в сторону увеличения номера элемента при вводе очередного числа. В начале программы массив с количествами остатков и переменную, в которой будет аккумулируется результат, обнулим.

Количество памяти, требуемое для работы программы, не завивсит от количества входных данных, а время ее работы зависит линейно от этого количества. Достигнута максимальная эффективность как по времени, так и по памяти.

Решение (C++).

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int ns[5], r[3]= {0}, i, n, ans=0, k;
    for(i=0; i<5; i++) // формируем очередь
    {
        cin>>n;
        ns[i]=n%3;   // сохраняем только остатки
    }
    for(cin>>n; n; i++)// ввод пока не встретили ноль
    {
        i%=5; // находим положение головы очереди
        n%=3; // работаем с остатками, а не с самими числами
        r[ns[i]]++; // подсчитываем количество остатков, вышедших из очереди
        if (n==0) k=r[0]; // находим пару
           else if (n==1) k=r[2];
                     else k=r[1];
        ans+=k; // наращиваем результат
        ns[i]=n; // помещаем остаток в очередь
        cin>>n; // вводим новое число
    }
    cout<<ans;
    return 0;
}
```

Для упрощения кода избавимся от условий. Составим таблицу зависимости остатков от деления на три чисел пары:

Остаток от деления первого числа пары на три	Остаток от деления второго числа пары на три
0	0
1	2
2	1

Видно, что на второй и третьей строках при возрастании числа в первом столбце, число во втором столбце убывает. Формула, которая реализует эту зависимость 3-*x*, где *x* – число в первом столбце. Значения, получаемые по этой формуле, представлены в таблице:

X	3- *x*
0	3
1	2
2	1

В первой строке вместо числа 3 нужно получить число 0, а все остальные числа в таблице меньше числа 3, поэтому окончательная формула имеет вид: (3 - *x)* mod 3, где mod – операция нахождения остатка от деления.

Запишем итоговую компактную программу:

```
#include <iostream>
using namespace std;
int main()
{
    int ns[5], r[3]= {0}, i, n, ans=0, k;
    for(i=0; i<5; i++) // формируем очередь
    {
        cin>>n;
        ns[i]=n%3;   // сохраняем только остатки введенных чисел
    }
```

```
    for(cin>>n; n; i++)// ввод пока не встретили ноль
    {
        i%=5; // находим положение головы очереди
        n%=3; // работаем с остатками, а не с самими числами
        r[ns[i]]++; // подсчитываем количество остатков, вышедших из очереди
        ans+=r[(3-n)%3]; // находим пару и наращиваем результат
        ns[i]=n; // помещаем остаток в очередь
        cin>>n; // вводим новое число
    }
    cout<<ans;
    return 0;
}
```

97) (**А.А. Богданов - Danov1802 №27-4**)

Задача A.

Решение на 2 балла выполняется в стандартном шаблоне, перебирающим все пары чисел из массива. Перебор осуществляется циклом в цикле. Сложность алгоритма $O(N^2)$. Используется массив для хранения ВСЕХ элементов последовательности.

Решение на PascalABC на 2 балла:

```
var n,i,j,k:integer;
    a:array[1..10000] of integer; //массив для всех элементов
begin
  read(n); // длина последовательности
  for i:=1 to n do
    read(a[i]); // считываем очередной элемент в массив
  k := 0; // счетчик
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if (a[i]*a[j] mod 5=0)and((a[i]+a[j])mod 3=0) then
        k:=k+1;
  writeln(k); // количество пар
end.
```

Задача B.

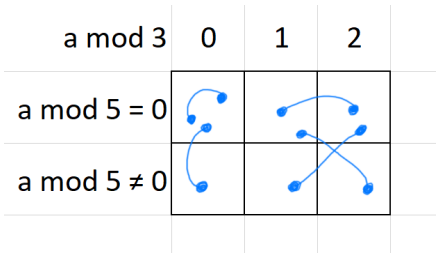
Решение на 4 балла обрабатывает последовательность за один проход и вычисляет значение нескольких счетчиков. Чтобы результат произведения был кратен 5 один из элементов пары должен быть кратен 5. А чтобы результат сложения был кратен 3 сумма остатков элементов пары должна быть кратна 3. Потребуется 6 счетчиков.

После основного цикла из значений счетчиков собирается общее количество пар. Будем считать количество элементов кратных и не кратных 5. А каждый из кратных и некратных 5 еще необходимо разбить по остатку от деления на 3. Всего 6 счетчиков, которые можно представить в виде двумерного массива, размером 2 на 3. Первая строка счетчиков содержит количество элементов кратных 5, вторая некратных. Столбцы соответствуют количеству элементов с остатками от деления на 3. Решение без массива будет более громоздким.

Инициализацию счетчиков можно сделать сразу при объявлении массива. А для расчета индекса строк будет полезна функция sign (знак), которая возвращает -1/0/+1 для отрицательных/нулевых/положительных аргументов. Пример:

sign(10 mod 5) = sign(0) = 0
sign(14 mod 5) = sign(4) = 1

Произведение счетчиков дает количество пар. На рисунке синим показаны пары счетчиков, произведение которых даст часть от общего количества пар с требуемыми условиями. Исключение составляет лишь пара (0,0)x(0,0). Это одно множество элементов и потому необходимо использовать формулу C(2,n)=n*(n-1)/2 (каждый с любым другим, без повторов).



Решение на PascalABC на 4 балла:

```
var n,a,i,k:integer;
    s:array[0..1,0..2] of integer=((0,0,0),(0,0,0));
begin
  read(n);
  for i:=1 to n do begin
    read(a);
    inc(s[sign(a mod 5),a mod 3]);
  end;
  k:=s[0,0]*(s[0,0]-1) div 2 + s[0,0]*s[1,0]
    +s[0,1]*(s[0,2]+s[1,2])+s[0,2]*s[1,1];
  writeln(k);
end.
```

98) (**А.А. Богданов - Danov1802 №27-5**)

Задача А.

Решение на 2 балла выполняется в стандартном шаблоне, перебирающим все пары чисел из массива. Перебор осуществляется циклом в цикле. Сложность алгоритма $O(N^2)$. Используется массив для хранения ВСЕХ элементов последовательности. В теле вложенного цикла ищется минимальная сумма пары и количество повторений этой суммы.

Решение на PascalABC на 2 балла:

```
var n,i,j,m,k:integer;
    a:array[1..10000] of integer; //массив для всех элементов
begin
  read(n); // длина последовательности
  for i:=1 to n do
    read(a[i]); // считываем очередной элемент в массив
  m := 20001; // инициализируем значением «больше большего»
  k := 0;     // количество пар с суммой равной минимальной
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if a[i]+a[j]<m then begin
        m := a[i]+a[j];
        k := 1;
      end else if a[i]+a[j]=m then
        k:=k+1;
  writeln(m,' ',k);
end.
```

Решение на 4 балла обрабатывает последовательность за один проход и вычисляет значение двух первых минимумов и количество их повторений в последовательности. Стандартный алгоритм поиска двух минимумов дополняется парой счетчиков. По окончанию прохода остается обработать две ситуации:

- минимумы равны и тогда количество пар рассчитывается по формуле C(2,n)=n*(n-1)/2, где n- количество повторений первого минимума;
- минимумы различны, тогда первый минимум встретился только один раз и количество пар определяется количеством повторений второго минимума.

В выходных данных минимальная сумма складывается из найденных двух минимумов.

Решение на PascalABC на 4 балла:

```
var n,a,i,m1,m2,k1,k2,k:integer;
begin
  read(n); // длина последовательности
  m1 := 10001; m2 := 10001;
  k1 := 0;
  for i:=1 to n do begin
    read(a);
    if a<m1 then begin
      m2 := m1; k2 := k1;
      m1 := a; k1 := 0;
    end else if a<m2 then begin
      m2 := a; k2 := 0;
    end;
    if a=m1 then k1:=k1+1;
    if a=m2 then k2:=k2+1;
  end;
  if m1=m2 then k := k1*(k1-1) div 2
  else k := k2;
  writeln(m1+m2,' ',k);
end.
```

Решение на Python на 4 балла (Е. Джобс):

Описание переменных:

sum – значение минимальной суммы

count – количество минимальных элементов впоследовательности до обрабатываемого значения

min – значение минимального элемента до текущего обработанного

k – количество минимальных сумм

Первичная инициализация:

Значение минимума – первое введенное значение

Минимальная суммма – невозможное значение больше максимального (20001)

Количество найденных минимальных сумм – 0

Количество минимумов – 1 (первое введенное значение).

1) Если нашли новое значение минимальной суммы (**if x + min < sum**:):

– обновляем значение минимальной суммы (**sum = x + min**)

– количество минимальных сумм равно количеству найденных минимумов (**k = count**)

Если нашли пару с суммой равной минимальной (**elif x + min == sum**)

– добавляем к количеству минимальных сумм количество минимумов, так как в сумме с любым из них получится минимальная сумма (**k = k + count**)

2) Если обрабатываемое значение меньше текущего минимума (**if min > x:**):

– обновляем минимум (**min = x**)
– сбрасываем счетчик минимумов (**count = 1**)
Если обрабатываемое значение равно минимуму (**elif x == min**):
– увеличиваем количество минимумов (**count = count + 1**)

```
n = int(input())
min = int(input())
sum, k, count = 20001, 0, 1

for i in range(n-1):
  x = int(input())
  if x + min < sum:
    sum = x + min
    k = count
  elif x + min == sum:
    k += count

  if min > x:
    min = x
    count = 1
  elif x == min:
    count += 1

print(sum, k)
```

99) (**А.А. Богданов - Danov1900 №27-1**)

Решение на 2 балла выполняется в стандартном шаблоне, перебирающим все пары чисел из массива. Перебор осуществляется циклом в цикле. Сложность алгоритма $O(N^2)$. Используется массив для хранения ВСЕХ элементов последовательности. В теле вложенного цикла ведется подсчет пар с суммой чисел и разностью индексов кратных трем.

Решение на PascalABC на 2 балла:

```
var n,i,j,k:integer;
  a:array[1..10000] of integer;
begin
  read(n);
  for i:=1 to n do
    read(a[i]);
  k:=0;
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if ((a[i]+a[j]) mod 3 = 0) and ((j-i) mod 3 = 0) then
        k := k+1;
  writeln(k);
end.
```

В языках C++/Python/Basic можно задать шаг цикла кратный 3. Тогда из условия можно убрать проверку кратности разности индексов. Можно совсем избавиться от проверки условия, если воспользоваться тем, что FALSE конвертируется к 0, а TRUE к 1.

Решение на C++ на 2 балла:

```
#include <iostream>
using namespace std;
int main()
{
```

```
  int n, i, j, k=0;
  int a[10000];
  cin >> n;
  for (i = 0; i < n; i++)
       cin >> a[i];

  for (i = 0; i < n - 1; i++)
       for (j = i + 3; j < n; j+=3)
            k += ((a[i] + a[j]) % 3 == 0);
  cout << k;
}
```

На Pascal для шага цикла на 3 можно использовать цикл while. Потребуется ручная инициализация и инкремент j. И тогда в коде Pascal из условия можно убрать проверку кратности разности индексов. Стандартная функция SIGN возвращает -1/0/+1 для отрицательных/нулевых/положительных аргументов. Таким образом, SIGN((a[i]+a[j]) mod 3) будет равно 0 при кратности суммы чисел и 1 в противном случае. Чтобы прибавлять 1 при sign(s)=0 и ничего не делать при sign(s)=1 будем увеличивать k на (1-sign((a[i]+a[j])mod 3)).

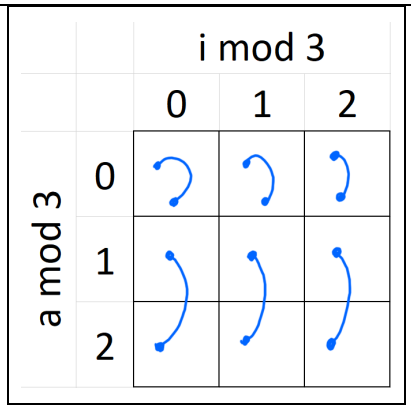
Решение на PascalABC на 2 балла:

```
var n,i,j,k:integer;
  a:array[1..10000] of integer;
begin
  read(n);
  for i:=1 to n do
    read(a[i]);
  k:=0;
  for i:=1 to n-3 do begin
    j:=i+3;
    while j<=n do begin
      k:=k+1-sign((a[i]+a[j]) mod 3);
      j:=j+3;
    end;
  end;
  writeln(k);
end.
```

Заметим, что код на Pascal стал заметно больше, чем на C++. Подобная оптимизация никак не повлияет на оценку экспертов.

Решение на 4 балла обрабатывает последовательность за один проход и для каждого элемента увеличивает соответствующий его свойствам счетчик. Всего нужно 9 счетчиков, которые удобно представить в виде матрицы 3x3. Номер строки (первого индекса) определяется остатком от деления значения элемента последовательности на 3. Номер столбца (второго индекса) определяется остатком от деления индекса элемента на 3.

После цикла объединим счетчики в пары, чтобы вычислить общее количество пар элементов с заданными условиями. Элементы множества (0,0) можно комбинировать с другими элементами этого же множества, т.е. количество пар получится C(2,n)=n*(n-1)/2, где n – мощность множества, т.е. значение счетчика (0,0). Множество (1,0) можно комбинировать с (2,0), тогда сумма элементов и разность индексов будут кратны 3. Между столбцами пар нет, т.к. тогда для пары не будет выполняться условие кратности 3 для разницы индексов. А в столбцах схема будет одинаковой, поэтому применим цикл по **J**. Приведем диаграмму Данова для обозначения пар.



Решение на PascalABC на 4 балла:

```
var n,i,j,a,s:integer;
  k:array[0..2,0..2] of integer=((0,0,0),(0,0,0),(0,0,0));
begin
  read(n);
  for i:=1 to n do begin
    read(a);
    k[a mod 3, i mod 3]+=1; // инкремент соответствующего счетчика
  end;
  s := 0;
  for j:=0 to 2 do
    s:=s+k[0,j]*(k[0,j]-1)div 2 + k[1,j]*k[2,j];
  writeln(s);
end.
```

100) (**А.А. Богданов - Danov1900 №27-1**)

Решение на 2 балла выполняется в стандартном шаблоне, перебирающим все пары чисел из массива. Перебор осуществляется циклом в цикле. Сложность алгоритма $O(N^2)$. Используется массив для хранения ВСЕХ элементов последовательности. В теле вложенного цикла ведется подсчет пар с произведением чисел и разностью индексов кратных трем. Т.к. разность индексов не может быть менее трех, можно добавить небольшую оптимизацию: заканчивать внешний цикл по **i** на n-3, а внутренний по **j** начинать с i+3.

Решение на PascalABC на 2 балла:

```
var n,i,j,k:integer;
  a:array[1..10000] of integer;
begin
  read(n);
  for i:=1 to n do
    read(a[i]);
  k:=0;
  for i:=1 to n-3 do
    for j:=i+3 to n do
      if ((a[i]*a[j]) mod 3 = 0) and ((j-i) mod 3 = 0) then
        k := k+1;
  writeln(k);
end.
```

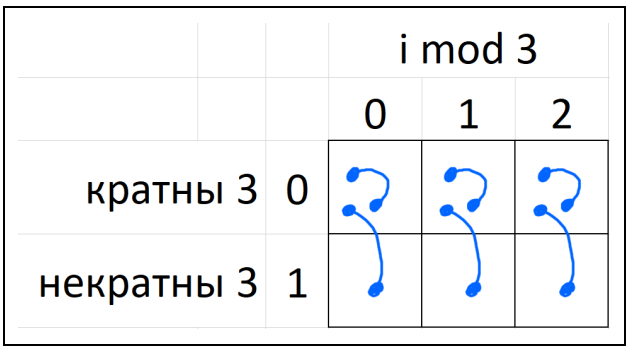
В языках C++/Python/Basic можно задать шаг цикла кратный 3. Тогда из условия можно убрать проверку кратности разности индексов. Можно совсем избавиться от проверки условия, если воспользоваться тем, что FALSE конвертируется к 0, а TRUE к 1.

Решение на C++ на 2 балла:

```
#include <iostream>
using namespace std;
int main()
{
  int n, i, j, k=0;
  int a[10000];
  cin >> n;
  for (i = 0; i < n; i++)
       cin >> a[i];

  for (i = 0; i < n - 3; i++)
       for (j = i + 3; j < n; j+=3)
            k += ((a[i] * a[j]) % 3 == 0);
  cout << k;
}
```

Решение на 4 балла обрабатывает последовательность за один проход и для каждого элемента увеличивает соответствующий его свойствам счетчик. Всего нужно 6 счетчиков, которые удобно представить в виде матрицы 2x3. Номер строки (первого индекса) определяется делимостью значения элемента последовательности на 3. В первой строке матрицы (строка 0) будем считать элементы кратные 3м, а во второй (строка 1) некратные. Номер столбца (второго индекса) определяется остатком от деления индекса элемента на 3.

После цикла объединим счетчики в пары, чтобы вычислить общее количество пар элементов с заданными условиями. Элементы множества (0,0) можно комбинировать с другими элементами этого же множества, т.к. они кратны 3м и результат их произведения тоже будет кратен 3, т.е. количество пар получится C(2,n)=n*(n-1)/2, где n – мощность множества, т.е. значение счетчика (0,0). Множество (0,0) можно комбинировать с (1,0), тогда произведение элементов и разность индексов будут кратны 3. Между столбцами пар нет, т.к. тогда для пары не будет выполняться условие кратности 3 для разницы индексов. А в столбцах схема будет одинаковой, поэтому применим цикл по **J** для столбцов. Приведем диаграмму Данова для обозначения пар.



Решение на PascalABC на 4 балла:

```
var n,i,j,a,s:integer;
    k:array[0..1,0..2] of integer=((0,0,0),(0,0,0));
begin
  read(n);
  for i:=1 to n do begin
    read(a);
    k[sign(a mod 3),i mod 3]+=1;
  end;
```

```
  s := 0;
  for j:=0 to 2 do
    s := s + k[0,j]*(k[0,j]-1) div 2 + k[0,j]*k[1,j];
  writeln(s);
end.
```

101) (**А.А. Богданов - Danov1901 №27**)

Решение на 2 балла выполняется в стандартном шаблоне, перебирающим все пары чисел из массива. Перебор осуществляется циклом в цикле. Сложность алгоритма $O(N^2)$. Используется массив для хранения ВСЕХ элементов последовательности. В теле вложенного цикла ведется поиск максимума с заданными ограничениями. Инициализируем переменную максимума значением -1. Если условие ни разу не сработает, тогда будет выведено -1, в соответствии с условием задачи.

Решение на PascalABC на 2 балла:

```
var n,i,j,m:integer;
  a:array[1..10000] of integer;
begin
  read(n);
  for i:=1 to n do
    read(a[i]);
  m:=-1;
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if ((a[i]+a[j]) mod 3 = 0) and ((i+j) mod 3 = 0) and
((a[i]+a[j])>m) then
        m := a[i]+a[j];
  writeln(m);
end.
```

В C++ нумерация массива начинается с 0. Однако, согласно условию, первый элемент имеет индекс 1. Поэтому в условии проверки кратности 3 суммы индексов нужно добавить 1+1=2.

```
(i + j + 2) % 3 == 0
```

или можно вынести 2 за знак равенства и условие примет вид:

```
(i + j) % 3 == 1
```

Решение на C++ на 2 балла:

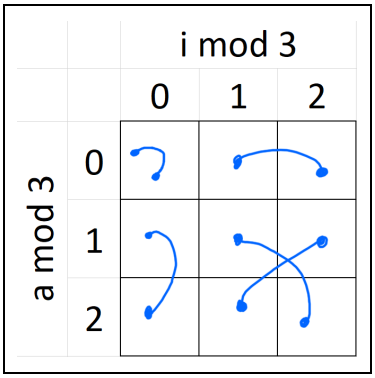
```
#include <iostream>
using namespace std;
int main()
{
  int n, i, j, m=-1, a[10000];
  cin >> n;
  for (i = 0; i < n; i++)
       cin >> a[i];

  for (i = 0; i < n - 1; i++)
    for (j = i + 1; j < n; j += 1)
      if ((a[i] + a[j]) % 3 == 0 &&
          (i + j) % 3 == 1 && a[i] + a[j] > m)
        m = a[i] + a[j];
  cout << m;
}
```

Решение на 4 балла использует модификацию алгоритма поиска первого и второго максимума. Алгоритм расширен с учетом заданных ограничений. Для каждого ограничения ищем первый максимум. А для чисел кратных 3 с индексом кратным 3 ищем и второй максимум, т.к. можно складывать 6+9=15, которые попадают в строку/столбик с индексом 0. Например, 7+8=15 попадают в разные строки/столбцы и нет необходимости искать второй максимум (со вторым максимумом пара не будет кратна 3).

Всего нужно найти 9+1 максимумов, которые удобно представить в виде матрицы 3x3 и еще одной отдельной переменной (второй максимум для ячейки (0,0). Номер строки (первого индекса) определяется остатком от деления значения элемента последовательности на 3. Номер столбца (второго индекса) определяется остатком от деления индекса элемента на 3.

Некоторые максимумы могут быть не найдены и это нужно учитывать при переборе возможных сочетаний пар. Так как мы будем выбирать из нескольких сумм найденных максимумов, можно проинициализировать переменную максимума таким отрицательным значением, чтобы даже в сумме с самым большим положительным элементом последовательности получалось отрицательное число, которое будет признаком, что пара не полная или не найдена. Максимальное число в последовательности 10000. Поэтому инициализировать будет значением -10001.

После прохода по всей последовательности объединим найденные максимумы в пары, чтобы найти максимальную сумму среди этих пар. Если найденная сумма максимумов окажется отрицательной, значит ничего не найдено и выведем -1. Найденные максимумы кратные трем можно комбинировать с другими максимумами кратными трем. Максимумы с остатком 1 и 2 дадут сумму кратную 3. Приведем диаграмму Данова, отображающую сочетание возможных пар.



Решение на PascalABC на 4 балла:

```
const z=-10001;
var n,i,a,m2,r,s,t:integer;
    m:array[0..2,0..2] of integer=((z,z,z),(z,z,z),(z,z,z));
begin
  read(n); m2 := z;
  for i:=1 to n do begin
    read(a); s := a mod 3; t := i mod 3;
    if (s=0) and (t=0) then
      if a>m[0,0] then m2 := m[0,0] // сохраняем второй максимум для
(0;0);
      else if a>m2 then m2 := a;
    if m[s,t]<a then m[s,t] := a;
  end;
  r := max(m[0,0]+m2,m[0,1]+m[0,2]);
  r := max(r,m[1,0]+m[2,0]);
  r := max(r,m[1,2]+m[2,1]);
```

```
  r := max(r,m[1,1]+m[2,2]);
  if r>0 then writeln(r)
  else writeln(-1);
end.
```

Решение на C++ на 4 балла:

```
#include <iostream>
using namespace std;
int max(int a, int b) { if (a > b) return a; else return b; }
int main()
{
  int n, a, r, s, t, m2 = -10001;
  int m[3][3];
  for (int i = 0; i < 3; i++)
       for (int j = 0; j < 3; j++)
            m[i][j] = -10001;
  cin >> n;
  for (int i = 1; i <= n; i++) {
       cin >> a;
       s = a % 3; t = i % 3;
       if (s == 0 && t == 0)
            if (a > m[0][0]) m2 = m[0][0]; // сохраняем второй
максимум для (0;0);
            else if (a > m2) m2 = a;
       if (m[s][t] < a) m[s][t] = a;
  }
  r = max(m[0][0] + m2, m[0][1] + m[0][2]);
  r = max(r, m[1][0] + m[2][0]);
  r = max(r, m[1][2] + m[2][1]);
  r = max(r, m[1][1] + m[2][2]);
  if (r > 0) cout << r;
  else cout << -1;
}
```

102) (**А.Г. Минак**)

Решение на Python на 4 балла:

```
N = int(input())
buf = [0] * 8
rem_div = [0] * 8
for i in range(8):
  buf[i] = int(input())
  rem_div[buf[i]%8] += 1

cnt = (rem_div[0] * (rem_div[0] - 1)) // 2
cnt += (rem_div[4] * (rem_div[4] - 1)) // 2

for i in range(1, 4):
  cnt += rem_div[i] * rem_div[8-i]

for _ in range(8, N):
  rem_div[buf[0]%8] -= 1
  buf.pop(0)
```

```
  x = int(input())
  d = x % 8
  if d == 0:
    cnt += rem_div[0]
  elif d == 4:
    cnt += rem_div[4]
  else:
    cnt += rem_div[8 - d]
  buf.append(x)
  rem_div[d] += 1
all_ = 7 * (N - 8) + (8 * (8 - 1)) // 2
print(all_ - cnt)
```

Решение на PascalABC на 4 балла (Д. Ф. Муфаззалов) :

Каждое вновь поступившее число может быть поставлено в пару с каждым числом, растояние до которого не более 7. Таких чисел может быть не более 7, поэтому организуем очередь с таким количеством элементов и при вводе нового числа будем увеличивать количество пар на количество чисел в очереди.

Если вновь поступившее число имеет остаток отделения на 8, равный 0 или 4, то оно не может быть в паре с числами, имеющими такой же остаток от деления на 8, так как сумма таких чисел будет кратна 8. Если поступило такое число, количество пар нужно уменьшить на количество чисел в очереди, имеющих соответствующий остаток.

Числа, имеющие при делении на 8 другие остатки (обозначим их за *r*), не могут быть в паре с числами, имеющими остаток от деления на 8, равный (8-*r*), по той же причине. Если поступило такое число, количество пар уменьшим на количество чисел в очереди, имеющих соответствующий остаток.

Заметим, что числа, имеющие остаток от деления 8, равный 0 и 4, имеют остаток от деления на 4, равный 0, а других чисел, имеющих при делении на 4 такой же остаток, нет. Это сократит программу.

Таким образом, поступающие числа нужно помещать в очередь размером 7, и хранить количество чисел в этой очереди, имеющих соответствующий остаток от деления на 8. Количество будем хранить в массиве размером 8. В очереди будем хранить не сами числа, так как они не нужны, а их остатки от деления на 8.

Очередь организуем в виде кольца. Голова очереди перемещается на 1 позицию с вводом очередного числа.

```
var
  A: array[0..7] of byte;
  B: array[0..7] of longint;
  ans, n, i, num: longint;
begin
  readln(n);
  ans := 0;
  for i := 0 to n - 1 do
  begin
    if (i > 7) then begin
      num := A[i mod 7];
```

```
      dec(B[num]);
    end;
    readln(num);
    num := num mod 8;
    ans := ans + min(i, 7);
    if num mod 4 = 0 then ans := ans - B[num] else ans := ans - B[8 -
num];
    A[i mod 7] := num;
    inc(B[num]);
  end;
  writeln(ans);
end.
```

Сумма чисел, равных количеству элементов в очереди при вводе каждого числа, не зависит от вводимых чисел. Действительно: при вводе первого числа очередь пуста, при вводе следующих 6 чисел, количество чисел в очереди увеличивается на 1, а затем остается постоянным. Поэтому сумма таких чисел равна 1+2+3+4+5 +6 +7*(n-7)=7*3+7n-49=7n-28, где n – количество вводимых чисел. Вычисление этой величины можно вынести из цикла. Заметим, что это число равно количеству всех пар чисел, расстояние между которыми не более 7.

Очередь организуем стандартными средствами языка программирования.

Решение на C++ на 4 балла (Д. Ф. Муфаззалов) :

```
#include <iostream>
#include <queue>
using namespace std;
int main()
{
    queue <int> A;
    int B[8]={0},ans=0,n,i,num;
    cin>>n;
    for (i=0; i<n; i++)
    {
        if (i>7)
        {
            B[A.front()]--;
            A.pop();
        }
        cin>>num;
        num%=8;
        if (num%4) ans-=B[8-num]; else ans-=B[num];
        A.push(num);
        B[num]++;
    }
    cout<<ans-28+7*n;
    return 0;
}
```

Программу можно еще сократить, унифицировав формулу для удаления ненужных пар.

Решение на Python на 4 балла (Д. Ф. Муфаззалов):

```
buf = []
rem_div = [0] * 8
N = int(input())
```

```
ans=7*N-28;
for i in range(0, N):
    num = int(input())%8
    ans-=rem_div[(8-num)%8]
    if i>6:
            rem_div[buf[0]]-=1 #//удаляем число из очереди
            buf.pop(0)  # // вместе с его остатком
    buf.append(num); #// добавление числа в очередь
    rem_div[num]+=1
print(ans)
```

103) (**О.Л. Дуркин**)

Решение задачи A.

Для организации перебора потребуется вложенный цикл. Внешний цикл будет брать первый элемент пары с индексом **i** (до **n**), вложенный цикл будет перебирать оставшиеся возможные элементы пары с индексом от 1 до последнего (**n**). Полный перебор необходим потому, что мы ищем разность (**a-b** и **b-a** дают разный результат). Для того чтобы учесть расстояние между элементами, нужно использовать следующее условие **|i-j|>=5**.

Программа на языке C++:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  int a[10000];
  int n, min_sub = 1001, i, j;
  cin >> n;
  for( i=0; i < n; i++ )
    cin >> a[i];
  for( i=0; i < n; i++ )
    for( j=0; j < n; j++ )
      if( (a[i]-a[j]) % 3 == 0 and
          a[i]-a[j] < min_sub and abs(i-j) >= 5 )
        min_sub = a[i]-a[j];
  if( min_sub == 1001 )
    cout << "NO";
  else
    cout << min_sub;
  return 0;
}
```

Решение задачи B. Разность двух чисел кратна трем, если:
- из числа кратного 3 вычитаем число кратное 3;
- из числа, которое при целочисленном делении на 3 даёт остаток 1, вычитаем число при целочисленном делении на 3 даёт остаток 1;
- из числа, которое при целочисленном делении на 3 даёт остаток 2, вычитаем число при целочисленном делении на 3 даёт остаток 2;

Для нахождения минимальной разности, нужно из всех чисел вычитать максимальное и сравнивать с **min_sub** или из минимального числа вычитать все числа и сравнивать разность с **min_sub**. При этом необходимо учитывать расстояние между этими числами. Для этого создадим массив из 6 элементов.

Программа на языке C++:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  int a[6];
  int min[3] = {1001, 1001, 1001};
  int max[3] = {0, 0, 0};
  int n,i,j;
  int min_sub = 1001;
  cin >> n;
  for( i=0; i < 5; i++ )
    cin >> a[i];
  for( i=5; i < n; i++ ) {
    cin >> a[5];
    if( a[0] > max[a[0]%3] )
      max[a[0]%3] = a[0];
    if( a[0] < min[a[0]%3] )
      min[a[0]%3] = a[0];
    if( (a[5]-max[a[5]%3]) < min_sub and
        max[a[5]%3] != 0)
      min_sub = a[5] - max[a[5]%3];
    if( (min[a[5]%3]-a[5]) < min_sub and min[a[5]%3] != 1001)
      min_sub = min[a[5]%3] - a[5];
    for( j=0; j < 5; j++ ) a[j] = a[j+1];
    }
  if (min_sub ==1001)
    cout<<"NO";
  else
    cout<<min_sub;
  return 0;
}
```

Решение на языке PascalABC.NET (Н. Чурсин)

```
var
  a: array[1..5] of integer;
  min, max: array[0..2] of integer;
  x, p, n, i, r, j, p2: integer;
begin
  readln(n);
  for i := 0 to 2 do
  begin
    min[i] := 1001;
    max[i] := -1;
  end;
  r := 1001;
  for i := 1 to 5 do
    readln(a[i]);
  for i := 6 to n do begin
    readln(x);
    p := a[1] mod 3;
    p2 := x mod 3;
    if a[1] < min[p] then min[p] := a[1];
```

```
    if a[1] > max[p] then max[p] := a[1];
    if (-1 * abs(x - max[p2]) < r) and (max[p2] > 0) then
      r := -1 * abs(x - max[p2]);
    if (-1 * abs(x - min[p2]) < r) and (min[p2] < 1001) then
      r := -1 * abs(x - min[p2]);
    for j := 1 to 4 do
      a[j] := a[j + 1];
    a[5] := x;
  end;
  if r < 1001 then writeln(r) else writeln('NO');
end.
```

104) (**О.Л. Дуркин**)

Решение задачи A. Программа на языке C++:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  int a[10000];
  int n, max_sub = -1001, i, j;
  cin >> n;
  for( i=0; i < n; i++ )
    cin >> a[i];
  for( i=0; i < n; i++ )
    for( j=0; j < n; j++ )
      if( (a[i]-a[j]) %3 == 0 and a[i]-a[j] > max_sub
          and abs(i-j) >= 5 )
        max_sub = a[i]-a[j];
  if( max_sub == -1001 )
    cout << "NO";
  else
    cout << max_sub;
  return 0;
}
```

Решение задачи B. Программа на языке C++:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  int a[6];
  int min[3] = {1001, 1001, 1001};
  int max[3] = {0, 0, 0};
  int n,i,j;
  int max_sub = -1001;
  cin >> n;
  for( i=0; i < 5; i++ )
    cin >> a[i];
  for( i=5; i < n; i++ ) {
    cin >> a[5];
    if( a[0] > max[a[0]%3] )
      max[a[0]%3] = a[0];
    if( a[0] < min[a[0]%3] )
```

```
      min[a[0]%3] = a[0];
    if( (max[a[5]%3] - a[5]) > max_sub and
        max[a[5]%3] !=0 )
       max_sub = max[a[5]%3]-a[5];
    if( (a[5] - min[a[5]%3]) > max_sub and
        min[a[5]%3] != 1001 )
      max_sub = a[5] - min[a[5]%3];
    for( j=0; j < 5; j++ ) a[j] = a[j+1];
    }
  if( max_sub ==-1001 )
    cout << "NO";
  else
    cout << max_sub;
  return 0;
}
```

Решение на языке PascalABC.NET (Н. Чурсин)

```
var
  a: array[1..5] of integer;
  min, max: array[0..2] of integer;
  x, p, n, i, r, j, p2: integer;

begin
  readln(n);
  for i := 0 to 2 do
  begin
    min[i] := 1001;
    max[i] := -1;
  end;
  r := -1;
  for i := 1 to 5 do
    readln(a[i]);
  for i := 6 to n do
  begin
    readln(x);
    p := a[1] mod 3;
    p2 := x mod 3;
    if a[1] < min[p] then min[p] := a[1];
    if a[1] > max[p] then max[p] := a[1];
    if (abs(x - max[p2]) > r) and (max[p2] > 0) then
      r := abs(x - max[p2]);
    if (abs(x - min[p2]) > r) and (min[p2] < 1001) then
      r := abs(x - min[p2]);
    for j := 1 to 4 do
      a[j] := a[j + 1];
    a[5] := x;
  end;
  if r > -1 then writeln(r) else writeln('NO');
end.
```

105) (**О.Л. Дуркин**)

Решение задачи A. Программа на языке C++:

```
#include <bits/stdc++.h>
```

```
using namespace std;
int main()
{
  int a[10000];
  int n, k = 0, i, j;
  cin >> n;
  for( i=0; i < n; i++ )
    cin >> a[i];
  for( i=0; i < n-5; i++ )
    for( j=i+5; j < n; j++ )
      if( a[i] - a[j] % 5 ==0 )
        k++;
  cout << k;
  return 0;
}
```

Решение задачи B. Программа на языке C++:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  int a[6];
  int count[5] = {0,0,0,0,0};
  int n, k = 0, i, j;
  cin >> n;
  for( i=0; i < 5; i++ )
    cin >> a[i];
  for( i=5; i < n; i++ ) {
    cin >> a[5];
    count[a[0]%5]++;
    k += count[a[5]%5];
    for( j=0; j < 5; j++ )
      a[j] = a[j+1];
    }
  cout << k;
  return 0;
}
```

Решение на языке PascalABC.NET (Н. Чурсин)

```
var
  a: array[1..5] of integer;
  k: array[0..4] of integer;
  i, n, x, j, p, p2, count: integer;
begin
  for i := 0 to 4 do
    k[i] := 0;
  readln(n);
  count := 0;
  for i := 1 to 5 do
    readln(a[i]);
  for i := 6 to n do begin
    readln(x);
    p := x mod 5;
    p2 := a[1] mod 5;
```

```
    k[p2] := k[p2] + 1;
    if k[p] > 0 then count := count + k[p];
    for j := 1 to 4 do
      a[j] := a[j + 1];
    a[5] := x;
  end;
  if count > 0 then writeln(count) else writeln('NO');
end.
```

106) (**Е.А. Мирончик**)

По условию задачи известно, что никакая пара не повторяется. Из этого следует, что если имеется две костяшки, соединенные в цепочку, то одновременный переворот этих двух костяшек не является цепочкой. То есть переворот костяшки разрывает имеющуюся цепочку (если костяшка не является парой двух одинаковых чисел).

Минимально возможная цепочка состоит из одной фишки. Каждую следующую фишку следует приставлять таким образом, чтобы по возможности продлить последнюю сформированную цепочку. Если новую костяшку не удается поставить так, чтобы она стала продолжением, то следует перевернуть последнюю костяшку и проверить может ли быть сформирована цепочка после переворота. При этом длина последней цепочки будет равна двум, так как переворот последней фишки наверняка разорвал связку между двумя последними костяшками. Если приставить последнюю не удается к последней костяшке, то длина последней цепочки снова становится равной одному. В процессе добавления новых костяшек будем следить за максимальным значением очередной цепочки.

```
program pr1;

type
  Tkost = record a, b: integer  end;
var
  i, n, max, t: integer;
  L: Tkost; // последняя обработанная костяшка
  R: Tkost; // новая костяшка
procedure p180(var k: Tkost); // переворот одной костяшки
var h: integer;
begin
  h := k.a; k.a := k.b; k.b := h;
end;

begin
  max := 1; // длина максимальной цепочки
  t := 1;   // длина последней цепочки
  readln(n);
  readln(L.a, L.b);
  for i := 2 to n do begin
    readln(R.a, R.b);
    if L.b = R.a then // костяшка продолжает цепочку
      t := t + 1
    else if L.b = R.b then begin // продолжает, но после поворота
      p180(R);
      t := t + 1;
    end
    else begin
```

```
      p180(L); // переворот последней костяшки построенной цепочки
      if L.b = R.a then // продолжение возможно
        t := 2
      else if L.b = R.b then begin // продолжение после поворота
        p180(R);
        t := 2;
      end
      else t := 1; // костяшка начинает новую цепочку
    end;
    if t > max then max := t;
    l := r;
  end;
  writeln(max);
end.
```

107) (**А.А. Богданов – Danov1902 №27-1**)

Задача А. Решение на 2 балла

Решение на 2 балла выполняется в стандартном шаблоне, перебирающим все пары чисел из массива. Перебор осуществляется циклом в цикле – $O(N^2)$. Используется массив для хранения ВСЕХ элементов последовательности. В теле вложенного цикла в **a[j]** ищется ноль и после ноля все ненулевые числа образуют пары с **a[j]**, которые считаются в **k**.

Решение на PascalABC.NET на 2 балла:

```
var n,i,j,k,f:integer;
    a:array[1..10000] of integer;
begin
  read(n); k:=0;
  for i:=1 to n do read(a[i]);
  for i:=1 to n-2 do if a[i]<>0 then begin
    f:=0;
    for j:=i+1 to n do
      if a[j]=0 then f:=1
      else if f=1 then k+=1;
  end;
  writeln(k);
end.
```

Задача Б. Решение на 4 балла

Последовательность можно разбить на несколько фрагментов, разделенных нулями. В каждом фрагменте подсчитываем количество элементов (**L, R**). Все элементы текущего фрагмента (**R**) образуют пары со всеми элементами предыдущих фрагментов (**L**). Пары считаются в момент окончания фрагмента, т.е. нахождения нулевого элемента. В конце нулевого элемента нет и поэтому после основного цикла необходимо добавить в результат пары, образуемые последним фрагментом.

Решение на PascalABC.NET на 4 балла:

```
var n,i,k,x:integer;
begin
  read(n); L:=0;R:=0;k:=0;
  for i:=1 to n do begin
    read(x);
    if x>0 then R+=1
    else begin
      k+=L*R;
      L:=L+R;
      R:=0;
```

```
    end;
  end;
  k+=L*R;
  writeln(k);
end.
```

108) (**А.А. Богданов – Danov1902 №27-2**)

Задача А. Решение на 2 балла

Решение на 2 балла выполняется в стандартном шаблоне, перебирающим все пары чисел из массива. Перебор осуществляется циклом в цикле – $O(N^2)$. Используется массив для хранения ВСЕХ элементов последовательности. В теле вложенного цикла в **a[j]** ищется ноль и после ноля все ненулевые числа образуют пары с **a[j]**, сумма которых проверяется на кратность трем, и затем пары считаются в **k**.

Решение на PascalABC на 2 балла:

```
var n,i,j,k,f:integer;
    a:array[1..10000] of integer;
begin
  read(n); k:=0;
  for i:=1 to n do read(a[i]);
  for i:=1 to n-2 do if a[i]<>0 then begin
    f:=0;
    for j:=i+1 to n do
      if a[j]=0 then f:=1
      else if (f=1)and((a[i]+a[j])mod 3=0) then k+=1;
  end;
  writeln(k);
end.
```

Задача Б. Решение на 4 балла

Последовательность можно разбить на несколько фрагментов, разделенных нулями. В каждом фрагменте подсчитываем количество элементов с остатками от деления на три – 0..2 (массивы **L,R**). Все элементы текущего фрагмента (**R**) образуют пары со всеми элементами предыдущих фрагментов (**L**). Пары считаются в момент окончания фрагмента, т.е. нахождения нулевого элемента. Числа с остатком 0 образуют пары с числами с остатком 0, а 1 с 2. В конце нулевого элемента нет и поэтому после основного цикла необходимо добавить в результат пары, образуемые последним фрагментом. Код вычисления пар дублируется и может быть вынесен в процедуру.

Решение на PascalABC.NET на 4 балла:

```
var n,i,j,k,x:integer;
    L,R:array[0..2] of integer;
begin
  read(n);
  k:=0;
  for j:=0 to 2 do (L[j],R[j]):=(0,0);
  for i:=1 to n do begin
    read(x);
    if x>0 then R[x mod 3]+=1
    else begin
      k+=L[0]*R[0]+L[1]*R[2]+L[2]*R[1];
      for j:=0 to 2 do begin
        L[j]:=L[j]+R[j];
        R[j]:=0;
```

```
      end;
    end;
  end;
  k+=L[0]*R[0]+L[1]*R[2]+L[2]*R[1];
  writeln(k);
end.
```

Дублирования кода вычисления количества пар можно избежать **(Муфаззалов Д.Ф., Уфа, УГАТУ)**. Отказавшись от комбинаторной формулы, решим задачу динамически, то есть будем находить количество пар для каждого ненулевого поступившего числа. Вместо двух массивов воспользуемся двумерным массивом.

Решение на балла (язык C++):

```
#include <iostream>
using namespace std;
int main()
{
    int n, j, rem[2][3] = {0}, a, ans = 0;
    for (cin >> n; n--;)
    {
        cin >> a;
        if (a)
        {
           ans += rem[1][(3 - a%3)%3];
           rem[0][a%3]++;
        }
        else
            for ( j = 0; j < 3; j++)
            {
                rem[1][j] += rem[0][j];
                rem[0][j] = 0;
            }
    }
    cout << ans;
    return 0;
}
```

109) (**А.А. Богданов – Danov1902 №27-3**)

Задача A. Решение на 2 балла

Решение на 2 балла выполняется в стандартном шаблоне, перебирающим все пары чисел из массива. Перебор осуществляется циклом в цикле – $O(N^2)$. Используется массив для хранения ВСЕХ элементов последовательности. В теле вложенного цикла в **`a[j]`** ищется ноль и после ноля все ненулевые числа образуют пары с **`a[j]`**, сумма которых проверяется на максимум, и затем сохраняется в **`maxs`**.

Решение на PascalABC.NET на 2 балла:

```
var n,i,j,f,maxs:integer;
    a:array[1..10000] of integer;
begin
  read(n); maxs:=0;
  for i:=1 to n do
    read(a[i]);
  for i:=1 to n-2 do
    if a[i]<>0 then begin
      f:=0;
      for j:=i+1 to n do
```

```
        if a[j]=0 then f:=1
        else if (f=1)and((a[i]+a[j])mod 3=0) and (a[i]+a[j]>maxs)
             then maxs := a[i]+a[j];
    end;
  writeln(maxs);
end.
```

Задача Б. Решение на 4 балла

Последовательность можно разбить на несколько фрагментов, разделенных нулями. В каждом фрагменте ищем максимальные элементы с остатками от деления на три – 0..2 (массивы **L, R**). Все максимумы текущего фрагмента (**R**) образуют пары с соответствующими максимумами предыдущих фрагментов (**L**). Максимальная сумма **maxs** вычисляется в момент окончания фрагмента, т.е. нахождения нулевого элемента. Максимумы элементов из **L** с остатком 0 образуют пары с максимумами из **R** с остатком 0, а 1 с 2. В конце нулевого элемента нет и поэтому необходимо еще раз выполнить код поиска максимума. Код поиска максимума дублируется и может быть вынесен в процедуру. Инициализацию элементов массивов **L** и **R** сделаем большим по модулю отрицательным числом, например -100000. Таким образом, мы исключим суммы пар с не найденными элементами **L** и **R**, т.к. в этом случае сумма будет отрицательной и меньшей, чем maxs, инициализированной 0.

Решение на PascalABC.NET на 4 балла:

```
var n,i,maxs,x:integer;
    L,R:array[0..2] of integer;
  procedure calcMaxS();begin
      maxs:=max(max(maxs,L[0]+R[0]),max(L[1]+R[2],L[2]+R[1]));
      for var j:=0 to 2 do begin
        L[j]:=max(L[j],R[j]);
        R[j]:= -100000;
      end;
    end;
begin
  read(n); maxs:=0;
  for i:=0 to 2 do (L[i],R[i]):=(-100000,-100000);
  for i:=1 to n do begin
    read(x);
    if x>0 then R[x mod 3] := max(R[x mod 3],x)
    else calcMaxS();
  end;
  calcMaxS();
  writeln(maxs);
end.
```

110) (**А.А. Богданов**)

Задача А. Решение на 2 балла

Используется массив для хранения ВСЕХ элементов последовательности. При загрузке элементов в массив запоминаем индекс максимального элемента. Перебор осуществляется циклом в цикле – $O(N^2)$. Первый цикл перебирает все элементы до максимального. Второй – все элементы после максимального. В теле вложенного цикла фильтруем пары с произведением кратным трём, которые считаем в **k**.

Решение на PascalABC.NET на 2 балла:

```
var a: array[1..10000] of integer;
    n, i, j, k, mi: integer;
begin
```

```
  read(n);
  mi:= 1; // индекс максимального
  k:= 0;
  for i:=1 to n do begin
    read(a[i]);
    if a[mi]<a[i] then mi:=i;
  end;
  for i:=1 to mi-1 do
    for j:=mi+1 to n do
      if a[i]*a[j] mod 3=0 then k+=1;
  writeln(k);
end.
```

Задача Б. Решение на 4 балла

Последовательность делится максимальным элементом на две части (левую и правую). Левая часть содержит **L1**/**L3** чисел некратных/кратных трем. Соответственно, **R1**/**R3** описывают правую часть последовательности. Левая часть последовательности удлиняется с нахождением каждого большего максимума. Считывая очередной элемент, мы увеличиваем счетчик **R1**/**R3** в соответствии с кратностью числа. При нахождении нового максимального числа содержимое **R1**/**R3** добавляется к счетчикам **L1**/**L3** и затем **R1**/**R3** обнуляются для подсчета чисел правой части последовательности. Прежний максимум становится элементом левой части, что отражается в инкременте соответствующего ему счетчика **L1**/**L3**. После окончания последовательности вычисляем количество пар, сочетая элементы множеств **L3** с **R3**, **L1** с **R3** и **L3** с **R1**.

Решение на PascalABC.NET на 4 балла:

```
var n, i, x, L1, L3, R1, R3, mx: integer;
begin
  (L3,L1,R3,R1):=(0,0,0,0);
  read(n);
  mx:= 0;
  for i:=1 to n do begin
    read(x);
    if x>mx then begin
      if mx>0 then
        if mx mod 3=0 then
             L3+= 1
        else L1+= 1;
      mx:= x;
      L3 += R3;
      L1 += R1;
      (R3,R1):=(0,0);
    end
    else if x mod 3 = 0 then
      R3 += 1
    else
      R1 += 1;
  end;

  writeln( L3*R3 + L3*R1 + L1*R3 );

end.
```

111) **Задача А. Решение на 2 балла**

Записываем все входные данные в массив, затем в двойном цикле рассматриваем все пары элементов, разница индексов которых не меньше, чем 4:

```
for i:=1 to n-4 do
  for j:=i+4 to n do
    ...
```

Если пара элементов **(a[i],a[j])** удовлетворяет условию и сумма этих элементов больше ранее найденного максимума:

```
if ((a[i]+a[j]) mod sk = 0) and
    (a[i] > a[j]) and (a[i]+a[j]>x+y) then begin
```

обновляем пару найденных элементов:

```
x := a[i];
y := a[j];
```

В начале в переменные **x** и **y** записываем нули. Если они остались равны нулю после завершения перебора, то ни одной подходящей пары не было обнаружено.

Решение на PascalABC.NET на 2 балла (А. Богданов):

```
const sk = 112;
var a:array[1..1000] of integer;
    n, i, j, x, y:integer;
begin
  x := 0; y := 0;
  read(n);
  for i:=1 to n do
    read(a[i]);
  for i:=1 to n-4 do
    for j:=i+4 to n do
      if ((a[i]+a[j]) mod sk = 0) and
          (a[i] > a[j]) and (a[i]+a[j]>x+y) then begin
        x := a[i];
        y := a[j];
      end;
  if x > 0 then begin
    writeln(x + y);
    writeln(x,' ',y);
  end
  else
    writeln(-1);
end.
```

Решение на Python на 2 балла (И. Гребенникова):

```
n = int(input())
a = []
for i in range(n):
  a.append(int(input()))
x = y = 0
for i in range(n-4):
  for j in range(i+4,n):
    if ( (a[i]+a[j]) % 112 == 0 and a[i] > a[j]
          and a[i]+a[j] > x+y ):
       x, y = a[i], a[j]
if x > 0:
  print(x + y)
  print(x, y)
else:
  print(-1)
```

Задача B. Решение на 4 балла

Заведем массив для хранения множества максимумов, размер которого не зависит от N (хотя рекордно большой для задач реального ЕГЭ). В **i**-той ячейке массива будем хранить максимальное число, которое при делении на 112 даёт остаток **i**. Этот элемент массива может быть первым (левым) элементом пары (вторым элементом в возможной паре будет очередное число, прочитанное из входного потока).

```
const sk = 112; // сумма должна быть кратна sk
var m: array[0..sk-1] of integer;
```

Пока забудем про то, что два интересующих нас элемента последовательности должны находиться на расстоянии не менее, чем 4.

В переменных **x** и **y** будем хранить найденную пару элементов, сначала обнуляем их, а также массив максимумов:

```
x := 0; y := 0;
for i:=0 to sk-1 do m[i]:=0; // обнуляем массив максимумов
```

Считываем в переменную **n** количество элементов последовательности:

```
read(n);
```

а в переменную **a** – её первый элемент:

```
read(a);
```

Теперь в цикле читаем и обрабатываем оставшиеся элементы. На каждом шаге цикла

1) добавляем только что прочитанный элемент к множеству элементов, которые могут быть первыми в паре; обновляем, если нужно, массив **m**:

```
k := a mod sk;
if a > m[k] then // нашли новый максимум с остатком k
  m[k] := a;
```

2) читаем очередной элемент

```
read(a); // считываем новый элемент и ищем ему пару
```

3) ищем ему пару: определяем для него остаток **k** и затем парный остаток **dk**, такой, что сумма остатков равна 0 или 112, так что сумма этих элементов делится на 112

```
k := a mod sk;
dk:= (sk - k) mod sk; // k + dk = 0 или 120
```

4) если первый элемент в паре (из массива **m**) больше, чем второй (текущий, из переменной **a**) и их сумма больше суммы запомненных ранее элементов, запроминаем новую пару:

```
if (m[dk] > a) and (m[dk]+a > x+y) then begin
  x := m[dk];
  y := a;
end;
```

5) при выводе остается проверить значение **x**: если оно равно 0, то ни одна пара не была найдена и следует вывести –1, а если нет – выводим сумму и отдельные элементы:

```
if x > 0 then begin
  writeln(x + y);
  writeln(x,' ',y);
end
else
  writeln(-1);
```

Теперь вспомним, что нам нужно, чтобы номера элементов последовательности отличались не менее, чем на 4. Для этого организуем очередь длиной 4 в виде массива (**подробности см. в разборе задачи 54**):

```
const d = 4; // смещение
var queue: array[0..d-1] of integer;
```

Только что прочитанный элемент участвует в обновлении массива **m** только пройдя через очередь, таким образом обеспечивается разница не менее, чем в **d**, между номерами всех элементов, находящихся в массиве **m**, и номером текущего элемента, только что прочитанного из потока.

Сначала заполняем очередь первыми элементами:

```
  for i:=0 to d-1 do  // первые d элементов в очередь
    read( queue[i] );
```

Элемент последовательности с номером **i** будет записываться в элемент очереди **queue[i mod d]**. В цикле для всех остальных элементов читаем первый элемент из очереди и добавляем его в список возможных первых элементов пары, изменяя при необходимости массив **m**:

```
  for i:=d to n-1 do begin
    first := queue[i mod d]; // первый элемент в очереди
    k := first mod sk;       // его остаток
    if first > m[k] then m[k] := first;
    ...
  end;
```

В конце очередной итерации цикла запиываем только что прочитанное значение в очередь на место удалённого первого элемента (подробности см. в разборе задачи 54):

```
  for i:=d to n-1 do begin
    ...
    read(a);
    ...
    queue[i mod d] := a;
  end;
```

Решение на PascalABC.NET на 4 балла (А. Богданов, К. Поляков):

```
const sk = 112; // сумма д/б кратна sk
      d = 4;
var m: array[0..sk-1] of integer;
    queue: array[0..d-1] of integer;
    n, i, first, a, k, dk, x, y:integer;
begin
  x := 0; y := 0;
  for i:=0 to sk-1 do m[i]:=0; // обнуляем массив максимумов

  read(n);

  for i:=0 to d-1 do  // считываем первые d элементов в очередь
    read( queue[i] );

  for i:=d to n-1 do begin
    first := queue[i mod d];
    k := first mod sk;
    if first > m[k] then m[k] := first;
    read(a); // считываем новый элемент и ищем ему пару
    k := a mod sk;
    dk:= (sk - k) mod sk; // k + dk = 0 или 120
    if (m[dk] > a) and (m[dk]+a > x+y) // левый больше? сумма больше?
    then (x, y):=(m[dk], a);
    queue[i mod d] := a;
  end;

  if x > 0 then begin
    writeln(x + y);
```

```
    writeln(x,' ',y);
  end
  else
    writeln(-1);
end.
```

Решение на Python на 4 балла:

```
sk = 112 # сумма д/б кратна sk
d = 4    # минимальная разница индексов
m = [0]*sk
queue = []
x = y = 0

n = int(input())

for i in range(d):  # первые d элементов в очередь
  queue.append( int(input()) )

for i in range(d, n):
  first = queue[i % d] # выбрали первый из очереди
  k = first % sk
  if first > m[k]:
    m[k] = first
  a = int(input())  # читаем следующий элемент
  k = a % sk        # и ищем ему пару
  dk = (sk - k) % sk
  if m[dk] > a and (m[dk]+a > x+y):
    x, y = m[dk], a
  queue[i % d] = a

if x > 0:
  print( x + y )
  print( x, y )
else:
  print( -1 )
```

112) (**Е. Джобс**)

Описание переменных

ost – массив, хранящий количество чисел с остатками при делении на 8; элемент ost[i] содержит количество чисел с остатком **i**

buf – массив-очередь для хранения последних 4 считанных значений

d – минимальная разность индексов элементов пары (здесь – 4)

Описание алгоритма

1) Считываем первые 4 элемента в очередь (см. разбор задачи 54).

```
buf = []
for i in range(d):
  buf.append(int(input()))
```

2) Последовательно считываем и обрабатываем **n-d** элементов. При считывании очередного значения количество чисел, с которыми можно составить пару, увеличивается на 1 – добавляем в это множество первый элемент из очереди.
   a) берём первый элемент из очереди, вычисляем его остаток (от деления на 8) и увеличиваем на 1 счётчик чисел с таким остатком;

б) у считанного числа вычисляем остаток **k** и *дополнительный остаток* **dk**, то есть такой остаток, который должны иметь все числа, сумма которых с новым числом кратна 8 (для 1 – 7, для 2 – 6 и т.д., для 0 – 0, от разности **8-k** берём остаток от деления на 8)

в) добавляем к количеству найденных пар количество уже выдвинутых из очереди чисел с дополнительным остатком **dk**

г) заносим в освободившийся элемент очереди новое (только что прочитанное) значение.

Решение на Python на 4 балла:

```
d = 4
n = int(input())
ost = [0]*8

buf = []
for i in range(d):
  buf.append(int(input()))

count = 0
for i in range(n-d):
  first = buf[i % d]
  k = first % 8
  ost[k] += 1
  x = int(input())
  k = x % 8
  dk = (8 - k) % 8  # дополнительный остаток
  count += ost[dk]
  buf[i % d] = x

print(count)
```

113) (**Е. Джобс**)

Нас интересует пара чисел с произведением, кратным 118, и нечётной суммой. Заметим, что число 118 раскладывается на простые сомножители 2 и 59.

Поскольку сумма двух чисел нечётная, одно из них чётное, второе – нечётное. При этом произведение, кратное 118, можно получить в двух случаях:

1) одно число – нечётное, кратное 59; второе – любое чётное;
2) одно число кратно 118; второе – любое нечётное.

Поэтому для поиска наибольшего такого произведения при последовательном вводе чисел нам достаточно найти 4 значения:

1) максимальное нечётное число, **maxOdd**
2) максимальное чётное число, **maxEven**
3) максимальное нечётное число, которое кратно 59, **max59**
4) максимальное число, которое кратно 118, **max118**

Тогда ответ – это максимальное из произведений **max59*maxEven**, **max118*maxOdd**.

Решение на Python на 4 балла:

```
maxOdd = maxEven = 0
max59 = max118 = 0

n = int(input())
for i in range(n):
  x = int(input())
  if x % 2 == 0:
    maxEven = max(maxEven, x)
    if x % 118 == 0:
      max118 = max(max118, x)
  else:
```

```
    maxOdd = max(maxOdd, x)
    if x % 59 == 0:
      max59 = max(max59, x)

s = max( maxEven*max59, maxOdd*max118 )

print(s)
```

Можно несколько оптимизировать программу, если хранить **maxEven** и **maxOdd** как два элемента массива **max1** (с индексами 0 и 1, соответственно), а **max118** и **max59** – как два элемента массива **max59** (с индексами 0 и 1, соответственно).

Решение на Python на 4 балла (Е. Джобс):

```
max1 = [0]*2
max59 = [0]*2

n = int(input())
for i in range(n):
  x = int(input())
  if x > max1[ x % 2 ]:
    max1[ x % 2 ] = x
  if x % 59 == 0 and x > max59[ x % 2 ]:
    max59[ x % 2 ] = x

s = max( max1[0]*max59[1], max1[1]*max59[0] )

print(s)
```

114) (**Е. Джобс**)

Описание алгоритма

Введём массив **a** для хранения минимальных значений, считанных до обрабатываемого, с остатком, равным индексу элемента. Это значит, что в любой момент **a[i]** содержит минимальное из всех полученных ранее значений, остаток от деления которых на 144 равен **i**.

```
a = [0]*144
```

В переменных **mini** и **minj** будут храниться два числа, которые дают минимальную сумму, кратную 144. Сначала запишем в эти переменные нули:

```
mini, minj = 0, 0
```

Так как по условию все числа положительные, нулевые значения означают, что ни одной пододящей пары не найдено.

Сумма пары (**mini** + **minj**) кратна 144, если сумма остатков элементов пары кратна 144, то есть равна 0 или 144. Получив новое число **x** из потока, делаем следующее:

1) находим остаток от деления x на 144:

```
rem1 = x % 144
```

2) находим остаток, который должно иметь второе число для образования пары с **x**:

```
if rem1 == 0:
  rem2 = 0
else:
  rem2 = 144 - rem1
```

это то же самое, что и

```
rem2 = (144 - rem1) % 144
```

3) находим минимальное из подходящих предыдущих чисел:

```
pair = a[rem2]
```

4) найдена новая подходящая пара, если

а) найдено предыдущее число, образующее пару с **x**, то есть **pair > 0**;

б) это число меньше, чем **x**;

в) ещё ни одной пары не найдено (**mini==0**) или новая пара имеет меньшую сумму (**pair + x <= mini + minj**)
условие **<=** (а не **<**) связано с тем, что требуется найти *последнюю* из подходящих пар с минимальной суммой

5) если найдена подходящая пара, запоминаем её:

```
mini, minj = pair, x
```

6) обновляем массив **a** с учётом нового значения **x**, если **x** меньше, чем **a[rem1]**, или это первое число с таким остатком (**a[rem1]==0**):

```
if 0 == a[rem1] or a[rem1] >= x:
  a[rem1] = x
```

Если первый элемент пары был перезаписан хотя бы один раз (**if mini != 0:**), выводим на экран найденную пару чисел, иначе выводим 0.

Решение на Python на 4 балла:

```
n = int(input())
a = [0]*144
mini, minj = 0, 0

for i in range(n):
    x = int(input())
    rem1 = x % 144
    rem2 = (144 - rem1) % 144
    pair = a[rem2]
    if pair > 0 and pair < x and \
      (mini == 0 or pair + x <= mini + minj):
        mini, minj = pair, x
    if 0 == a[rem1] or a[rem1] >= x:
        a[rem1] = x

if mini != 0:
    print(mini, minj)
else:
    print(0)
```

Решение на языке PascalABC.NET (Н. Чурсин)

```
const m = 144;
var
  i, n, x, l, r, p, s: integer;
  a: array[0..m - 1] of integer;
begin
  readln(n);
  for i := 0 to m - 1 do
    a[i] := 12001;
  s := 24001;
  for i := 1 to n do
  begin
    readln(x);
    p := x mod m;
    if (p = 0) and (x > a[0]) and (x + a[0] <= l + r) then begin
      l := a[0];
      r := x;
      s := l + r;
    end;
    if (p > 0) and (x > a[m - p]) and (x + a[m - p] <= s) then begin
      l := a[m - p];
```

```
      r := x;
      s := l + r;
    end;
    if x < a[p] then a[p] := x;
  end;
  if s = 24001 then writeln(0) else
    writeln(l, ' ', r);
end.
```

115) (**Е. Джобс**)

Описание алгоритма

Будем рассматривать пары элементов последовательности (a_j, a_i), в которых $j < i$. Пусть мы только что прочитали из потока **a[i]**. Чтобы найти пару с максимальной суммой, в которой второй элемент – **a[i]**, в качестве первого элемента **a[j]** нужно взять наибольший из всех предыдущих элементов, для которых остаток от деления **j** на **K** совпадает с остатком от деления **i** на **K** (в этом случае разность **i-j** кратна **K**). Перебирая все возможные максимальные суммы для всех элементов последовательности, читаемых из потока, находим максимальную из них и запоминаем образующие её значения.

Первые K значений читаем в массив с индексами от 0 до K-1, где будут храниться максимальные элементы, которые при делении на K дают остаток 0, 1, …, K-1:

```
ma = []
for i in range(K):
  ma.append( int(input()) )
```

Остальные данные читаем и обрабатываем в цикле:

```
for i in range(N - K):
  x = int(input())
  ...
```

Вычисляем остаток от деления номера **i** на **K**, берём максимальный из предыдущих элементов этой группы, проверяем полученную сумму и, если нужно, обновляем максимальные значения:

```
  r = i % K
  if ma[r] + x > max1 + max2:
    max1, max2 = ma[r] + x
```

Наконец, не забываем обновить элемент массива **ma**:

```
  if x > ma[r]:
    ma[r] = x
```

Полная программа на языке Python:

```
N, K = map(int, input().split())
ma = []
for i in range(K):
  ma.append( int(input()) )

max1, max2 = 0, 0
for i in range(N - K):
  x = int(input())
  r = i % K
  if ma[r] + x > max1 + max2:
    max1, max2 = ma[r], x
  if x > ma[r]:
    ma[r] = x

print(max1, max2)
```

116) (**Е. Джобс, К. Поляков**)

Описание алгоритма

Будем рассматривать пары элементов последовательности (a_j, a_i), в которых $j < i$. Пусть мы только что прочитали из потока **a[i]**. Чтобы найти пару с максимальной разностью, в которой второй элемент – **a[i]**, нужно проверить две разности:

1) **a[j]-a[i]**, где **a[j]** – наибольший из всех предыдущих элементов, для которых остаток от деления **j** на **K** совпадает с остатком от деления **i** на **K** (в этом случае разность **i-j** кратна **K**);
2) **a[i]-a[j]**, где **a[j]** – наименьший из всех предыдущих элементов, для которых остаток от деления **j** на **K** совпадает с остатком от деления **i** на **K**.

Перебирая все возможные максимальные разности для всех элементов последовательности, читаемых из потока, находим максимальную из них и запоминаем образующие её значения.

Вводим два массива **mi** (минимальные элементы) и **ma** (максимальные элементы) с индексами от 0 до K-1. Первые K значений читаем в оба массива:

```
ma, mi = [], []
for i in range(K):
  x = int(input())
  ma.append(x)
  mi.append(x)
```

Остальные данные чистаем в цикле:

```
for i in range(N - K):
  x = int(input())
  ...
```

Вычисляем остаток от деления номера **i** на **K**, находим максимальный и минимальный из предыдущих элементов этой группы, проверяем полученные разности и, если нужно, обновляем максимальные и минимальные значения:

```
  r = i % K
  if x - mi[r] > maxDiff:
    max1, max2, maxDiff = mi[r], x, x - mi[r]
  if ma[r] - x > maxDiff:
    max1, max2, maxDiff = ma[r], x, ma[r] - x
```

Наконец, не забываем обновить элементы массивов **ma** и **mi**:

```
  if x > ma[r]:
    ma[r] = x
  if x < mi[r]:
    mi[r] = x
```

Полная программа на языке Python:

```
N, K = map(int, input().split())
ma, mi = [], []
for i in range(K):
  x = int(input())
  ma.append(x)
  mi.append(x)

max1, max2, maxDiff = 0, 0, -1 # разность может быть 0
for i in range(N - K):
  x = int(input())
  r = i % K
  if x - mi[r] > maxDiff:
    max1, max2, maxDiff = mi[r], x, x - mi[r]
  if ma[r] - x > maxDiff:
    max1, max2, maxDiff = ma[r], x, ma[r] - x
  if x > ma[r]:
    ma[r] = x
  if x < mi[r]:
    mi[r] = x

print(max1, max2)
```

117) (**А. Кабанов**)

Нас интересуют пары чисел на расстоянии кратном 5. Разобьём нашу последовательность на 5 частей, по остатку от деления индекса на 5. Поскольку у чисел в одной группе индексы будут иметь одинаковый остаток, разница между ними будет кратна 5.

Для каждой части создадим массив в котором будем хранить максимальные значения, притом в **i** ячейке массива будет храниться число, которое при делении на 7 даёт остаток **i**.

Для удобства для записи максимальных значений будем использовать двумерный массив. Строки будут означать остаток от деления индекса на 5, а столбцы – остаток от деления значения числа на 7.

Рассматриваемые пары состоят из 2 чисел: входное число и элемент массива, такой что номер строки – остаток от деления индекса входного числа на 5, а номер столбца – дополняющий до кратного 7 остаток к входному числу.

Для каждого входящего числа **x** вычисляем **k** – остаток от деления индекса на 5, **r** – остаток от деления числа **x** на 7, **r1** – дополнительный остаток, такой что **r+r1** кратно 7.

Далее проверяем, что для числа **x** существует подходящее ему значение **a** и что их сумма максимальна. Если это так – записываем их в соответствующие переменные.

В конце цикла проверяем, если **x** больше соответствующего значения **a**, то записываем его в массив.

После проверки всех чисел выводим 0, если искомой пары не нашлось и саму пару при положительном исходе.

Решение на 2 балла на языке Python:

```
a = []
m1 = -1
m2 = -1
n = int(input())
for i in range(n):
  a.append( int(input()) )
for i in range(n-1):
  for j in range(i+1,n):
    if( (j-i) % 5 == 0 and (a[i]+a[j]) % 7 == 0 and
        (a[i]+a[j]) > (m1+m2)):
      m1 = a[i]
      m2 = a[j]
if m1 == -1:
    print(0)
else:
    print(m1,m2)
```

Решение на 4 балла на языке Python:

```
#массив максимальных значений
a = []
for i in range(5):
  a.append( [-1]*7 )
#Переменные для хранения искомой пары
m1 = -1
m2 = -1
n = int(input())
for i in range(n):
  x = int(input())
  k = i % 5
  r = x % 7
  r1 = (7 - r) % 7
  if a[k][r1]!=-1 and (x+a[k][r1]) > (m1+m2):
    m1 = x
    m2 = a[k][r1]
```

```
  if x > a[k][r]:
    a[k][r] = x
if m1 == -1:
    print(0)
else:
    print(m1, m2)
```

Решение на 4 балла на языке Паскаль:

```
var a: array [1..5,1..7] of integer;
    n,m1,m2,i,j,x,k,r,r1: integer;
begin
  m1:=-1;
  m2:=-1;
  for i:=1 to 5 do
    for j:=1 to 7 do
      a[i][j]:= -1;

  readln(n);
  for i:=1 to n do begin
    readln(x);
    k:= (i mod 5) + 1;
    r:= (x mod 7) + 1;
    r1:= (7 - x mod 7) mod 7 + 1;

    if (a[k][r1]<>-1) and ((x+a[k][r1]) > (m1+m2)) then begin
      m1:= x;
      m2:= a[k][r1]
    end;

    if x > a[k][r] then
      a[k][r]:= x;
  end;

  if m1 = -1 then
    writeln(0)
  else
    writeln(m1,' ',m2);
end.
```

118) (**А. Кабанов**)

Нас интересуют пары чисел с чётным произведением. Произведение будет чётным, если хотя бы одно число из пары чётно. Поэтому разобьём нашу последовательность на чётные и нечётные числа.

Для каждой части создадим массив, в котором будем хранить количество чисел (**k0** для чётных, **k1** для нечётных), притом в **i** ячейке массива будет храниться количество чисел, которое при делении на 13 даёт остаток **i**.

Общее количество пар равно суммарному числу пар, которые образуются элементами последовательности с предыдущими числами.

Для каждого входящего числа **x** вычисляем **ost** – остаток от деления числа на 13.

Если x чётный, то нужные пары образуются с чётными и нечётными числами, остаток от деления на 13 которых равен **ost**. Таким образом, число пар увеличится на **k0[ost]+k1[ost]**.

Если x нечётный, то нужные пары образуются только с чётными числами, остаток от деления на 13 которых равен **ost**. Таким образом, число пар увеличится на **k0[ost]**.

После увеличения общего количество пар обновляем счётчики количества чисел в зависимости от чётности **x**.

После проверки всех чисел выводим NO, если искомой пары не нашлось и общее количество пар при положительном исходе.

Решение на 4 балла (язык Python)

```
#количество чёт/нечёт чисел, разбитое по остаткам
k0=[0]*13
k1=[0]*13

#общее количество чисел
k=0

n = int(input())
for i in range(n):
    x = int(input())
    ost = x%13
    if x%2==0:
        k+= k0[ost]+k1[ost]
        k0[ost]+= 1
    else:
        k+= k0[ost]
        k1[ost]+=1

if k==0:
    print('NO')
else:
    print(k)
```

119) (**А. Кабанов**)

Нас интересуют пары чисел, произведение которых делится на 7. Произведение будет делиться на 7, если хотя бы одно число из пары делится на 7. Поэтому разобьём нашу последовательность на кратные и не кратные семи.

Для каждой части создадим массив, в котором будем хранить количество чисел (**k1** для не кратных 7, **k7** для кратных 7), притом в **i** ячейке массива будет храниться количество чисел, которое при делении на 10 даёт остаток **i**.

Общее количество пар равно суммарному числу пар, которые образуются элементами последовательности с предыдущими числами.

Для каждого входящего числа x вычисляем **ost** – остаток от деления числа на 10. **ost1** – дополнительный остаток, такой что **ost+ost1** кратно 10.

Если x делится на 7, то нужные пары образуются со всеми числами, остаток от деления на 10 которых равен **ost1**. Таким образом, число пар увеличится на **k1[ost1]+k7[ost1]**.

Если x не кратен 7, то нужные пары образуются только с числами, делящимися на 7, остаток от деления на 13 которых равен **ost1**. Таким образом, число пар увеличится на **k7[ost1]**.

После увеличения общего количество пар обновляем счётчики количества чисел в зависимости от делимости **x**.

После проверки всех чисел выводим NO, если искомой пары не нашлось и общее количество пар при положительном исходе.

Решение на 4 балла (язык Python)

```
#количество кратных/не кратных 7, разбитое по остаткам
k7=[0]*10
k1=[0]*10

#общее количество пар
k=0

n=int(input())
```

```
for i in range(n):
    x=int(input())
    ost=x%10
    ost1=(10-ost)%10

    if x%7==0:
        k+=k1[ost1]+k7[ost1]
        k7[ost]+=1
    else:
        k+=k7[ost1]
        k1[ost]+=1
print(k)
```

120) (**А. Кабанов**)

Нас интересуют пары чисел, произведение которых чётно. Произведение будет чётным, если хотя бы одно из чисел чётно. Поэтому разобьём нашу последовательность на чётные и нечётные числа. Для каждой части создадим массив, в котором будем хранить максимальные значения (**m0** для чётных, **m1** для нечётных), притом в **i** ячейке массива будет храниться число, которое при делении на 5 даёт остаток **i**.

Рассматриваемые пары состоят из 2 чисел: входное число и элементы массива, такие что их номер – дополняющий до кратного 5 остаток к входному числу. Притом элементы массива **m1** будут подходить только в том случае, если входное число чётно.

Для каждого входящего числа x вычисляем ost – остаток от деления на 5, **ost1** – дополнительный остаток, такой что **ost+ost1** кратно 5.

Далее проверяем, что для числа x существует подходящее ему значение **m0** или **m1** и что их сумма максимальна. Если это так – записываем их в соответствующие переменные.

В конце цикла проверяем, если **x** больше соответствующего значения массива **m0** или **m1**, то записываем его в массив.

После проверки всех чисел выводим 0, если искомой пары не нашлось и саму пару при положительном исходе.

Решение на 4 балла (язык Python):

```
#массивы максимальных значений
m0=[-1]*5
m1=[-1]*5
#Переменные для хранения искомой пары
mi=-1
mj=-1

n=int(input())
for i in range(n):
    x=int(input())
    ost=x%5; ost1=(5-ost)%5

    if x%2==0 and m1[ost1]!=-1 and (m1[ost1]+x)>(mi+mj):
        mi, mj = m1[ost1], x

    if m0[ost1]!=-1 and (m0[ost1]+x)>(mi+mj):
        mi, mj = m0[ost1], x

    if x%2==0 and x>m0[ost]:
        m0[ost]=x
    if x%2!=0 and x>m1[ost]:
        m1[ost]=x
```

```
if mi==-1:
    print(0)
else:
    print(mi,mj)
```

Решение на 4 балла, язык Паскаль (М.В. Агапова):

```
var N,x,i,mi,mj,ost,ost1:integer;
    m0,m1:array[0..4] of integer;
  {массивы чётных и нечётных чисел с индексами, соответствующими
   остаткам вводимых чисел}
begin
   readln(N);
   mi:=0;      {левый элемент искомой пары}
   mj:=0;      {правый элемент искомой пары}
   for i:=0 to 4 do begin
       m0[i]:=0;
       m1[i]:=0
   end;
   for i:=1 to N do begin
          readln(x);
          ost:=x mod 5;
          ost1:=(5-ost)mod 5;
          if (x mod 2 = 0) and (m1[ost1]<>0) and (x+m1[ost1]>mi+mj)
  {проверяем чётность x, наличие пары в нечётном массиве
   и сравниваем с максимальной суммой}
            then  begin
                mi:=m1[ost1];
                mj:=x
            end;
          if (m0[ost1]<>0) and (x+m0[ost1]>mi+mj) {для любого x
проверяем наличие пары в чётном
массиве и сравниваем с максимальной суммой}
             then  begin
                mi:=m0[ost1];
                mj:=x
            end;
          if (x mod 2 = 0) and(x>m0[ost])
             then m0[ost]:=x;    {переобозначаем эл-т чётного массива}
          if (x mod 2 <> 0) and (x>m1[ost])
             then m1[ost]:=x;  {переобозначаем эл-т нечётного массива}
   end;
   if mi=0 then writeln(0) {не нашлось элементов}
           else writeln(mi,' ',mj)

end.
```

121) Простейший вариант решения – прочитать все данные в массив

```
N = int(input())
A = []
for i in range(N):
  x = int(input())
  A.append(x)
```

а затем перебрать все пары элементов массива:

```
for i in range(N-1):
```

```
    for j in range(i+1,N):
      ...
```

Нужно не забыть про особенность построения этих вложенных циклов:

а) внешний цикл должен заканчиваться на предпоследнем элементе (но это не критично, если правильно написан второй цикл);

б) внутренний цикл должен начинаться с индекса **i+1**, чтобы пары не учитывались повторно и не учитывались комбинации элемента с самим собой

Если при проверке очередной пары все условия выполняются, счётчик увеличивается на 1:

```
  if (A[i] + A[j]) % D == 0 and \
    (A[i] > M or A[j] > M):
    count += 1
```

Решение на 2 балла (язык Python):

```
N = int(input())
A = []
for i in range(N):
  x = int(input())
  A.append(x)
count = 0
D = 40
M = 30
for i in range(N-1):
  for j in range(i+1,N):
    if (A[i] + A[j]) % D == 0 and \
       (A[i] > M or A[j] > M):
      count += 1
print(count)
```

Решение на 4 балла.

Рассмотрим сначала более простую задачу: найти количество пар чисел, сумма которых кратна 40 (без ограничения на величину чисел). Это означает, что сумма остатков от деления этих двух чисел на 40 равна или 0, или 40. Для определения количества пар, сумма которых кратна 40, достаточно посчитать (с помощью массива **B** размером 40) количество чисел, которые при делении на 40 дают в остатке числа от 0 до 39:

```
  D = 40
  B = [0]*D
  for i in range(N):
    x = int(input())
    B[x % D] += 1
```

Теперь посчитаем общее количество пар. Числа с нулевым остатком комбинируются с теми же числами, каждое число – с оставшимися (кроме самого себя). При этом каждое число считается дважды, то есть произведение **B[0]*(B[0]-1)** нужно разделить на 2:

```
  count = 0
  count += B[0]*(B[0]-1) // 2
```

Поскольку число D = 40 чётное, два числа с остатками 20 могут образовать пару, таких пар будет (как и для остатка 0) ещё

```
  count += B[20]*(B[20]-1) // 2
```

Теперь рассмотрим пары чисел с разными остатками. Чтобы не учитывать пары дважды, переменная цикла будет проходить все значения от 1 до 19 (это меньший остаток из двух):

```
  for r in range(1, 20):
    count += B[r]*B[40-r]
```

Вернёмся к исходной задаче. Поскольку нас интересует еще и величина числа (хотя бы одно из пары должно быть больше 30), заведем два массива: в массиве **B** (*big*, большой) будем хранить

количество чисел, больших 30, а в массиве **S** (*small*, маленький) – количество чисел, меньших или равных 30:

```
D = 40
M = 30
N = int(input())
S = [0]*D
B = [0]*D
for i in range(N):
  x = int(input())
  if x > M:
    B[x % D] += 1
  else:
    S[x % D] += 1
```

Рассмотрим пары чисел с остатками, равными 0. Мы можем комбинировать маленькое число с большим (тут точно нет повторения), или большое с большим (здесь нужно учесть повторения, как было сделано выше):

```
count = 0
count += S[0]*B[0] + B[0]*(B[0]-1)//2
```

Аналогично добавляем количество пар, в которых каждый остаток равен 20:

```
count += S[20]*B[20] + B[20]*(B[20]-1)//2
```

При построении пар из чисел с разными остатками мы можем комбинировать маленькое число с большим, большое с маленьким или два больших:

```
for r in range(1, 20):
  count += S[r]*B[40-r] + S[40-r]*B[r] + B[r]*B[40-r]
```

Решение на 4 балла (язык Python):

```
D = 40
M = 30
N = int(input())

S = [0]*D
B = [0]*D

for i in range(N):
  x = int(input())
  if x > M:
    B[x % D] += 1
  else:
    S[x % D] += 1

count = 0
# остаток 0
count += S[0]*B[0] + B[0]*(B[0]-1)//2
# разные остатки r и 40-r > r
# остаток D // 2 = 20
count += S[D//2]*B[D//2] + \
         B[D//2]*(B[D//2]-1)//2
for r in range(1, D//2):
  count += S[r]*B[D-r] + S[D-r]*B[r] + B[r]*B[D-r]
print(count)
```

Решение на Паскале (М.В. Агапова):

```
const m=40;
var N,i,a,p,k:integer;
    rb,rm: array[0..m-1] of integer;
```

```
      { количество чисел >30 и <=30 с остатками,
        равными индексу массива}
begin
   readln(N);
   for i:=0 to m-1 do begin
     rb[i]:=0;
     rm[i]:=0
   end;
   k:=0;
   for i:=1 to N do begin
     readln(a);
     p:=a mod m;
     if a>30 then  begin
       k:=k+rm[(m-p) mod 40]
           +rb[(m-p) mod 40]; {учитываем количество пар
                               чисел <=30 и >30}
       rb[p]:=rb[p]+1   {увеличиваем количество чисел >30}
     end
```

```
      else begin
        k:=k+rb[m-p]; {учитываем количество пар чисел только >30}
        rm[p]:=rm[p]+1 {увеличиваем количество чисел <=30}
      end;
    end;
    writeln(k)
end.
```

Ещё одно решение (В. Бабий):
При решении «в лоб», т.е. при поиске непосредственно количества нужных по условию пар, возникает сложность для учёта всех возможных ситуаций. То есть нам необходимо найти количество пар, в которых оба числа больше 30, и количество пар, в которых только одно число больше 30. В результате при решении у ученика может возникнуть путаница с составлением выражений для вычисления количества пар. При этом нужно учесть всё, и ничего не забыть. В альтернативном решении мы находим общее количество пар, в которых сумма кратна 40, и количество пар, в которых сумма кратна 40, и оба элемента меньше 30. Затем из общего количества пар вычитаем не нужное, т.е. количество пар, в которых оба элемента меньше 30. Мы так же используем два массива, но вычисления упрощаются за счет того, что мы не перемножаем элементы из разных массивов.

Пример кода на языке Python (4 балла):

```
n = int(input())
a = [0]*40
b = [0]*40
for i in range(n):
  x = int(input())
  a[x % 40] += 1
  if (x <= 30):
    b[x % 40] += 1
R1 = (a[0]*(a[0]-1) // 2) + (a[20]*(a[20]-1) // 2)
R2 = (b[0]*(b[0]-1) // 2) + (b[20]*(b[20]-1) // 2)
for i in range(1, 20):
  R1 += a[i] * a[40-i]
  R2 += b[i] * b[40-i]
print(R1-R2)
```

Есть и другой способ избежать путанницы при составлении комбинаторного выражения для вычисления количества пар **(Д. Ф. Муфаззалов).** Решим задачу динамически, то есть будем находить количество пар поступившему числу, и накапливать их.
Найдем пару поступившему числу с точки зрения остатка от деления его на число 40:

- если остаток равен нулю, то оно может быть в паре с числом, у которого остаток отделения на число 40 равен нулю.
- если остаток не равен нулю (обозначим остаток переменной *r*), то оно может быть в паре с числом, у которого остаток отделения на число 40 равен числу (40-*r*).

Обобщим сказанное: если остаток от деления поступившего числа на число 40 равен числу *r*, то оно может быть в паре с числом, у которого остаток отделения на число 40 равен остатку от деления числа (40-*r*) на число 40.
Найдем пару поступившему числу с точки зрения сравнения его с числом 30:

- если поступившее число больше числа 30, то оно может быть в паре с любым числом.
- если поступившее число не больше числа 30, то оно может быть в паре только с числом, большим числа 30.

Будем сохранять количество чисел, имеющих соответсвующий осаток от деления на число 40, и соответствующий результат сравнения с числом 30, среди уже обработанных. Зная характеристики вновь поступившего числа, найдем количество пар для него среди обработанных чисел.

Решение на C++:

```
#include <iostream>
using namespace std;
int main()
{
    int ans=0, a[40][2]={0}, n, x, j, k;
    for(cin>>n; n--;)
    {
        cin>>x;
        j=(40-x%40)%40, k=x>30;
        ans+=a[j][1]+a[j][0]*k;
        a[x%40][k]++;
    }
    cout<<ans;
    return 0;
}
```

Решение на Python:

```
n = int(input())
a = [[0] * 2 for i in range(40)]
ans=0
for i in range(n):
  x = int(input())
  j=(40-x%40)%40
  ans+=a[j][1]+a[j][0]*(x>30)
  a[x%40][x>30]+=1
print(ans)
```

122) (**В. Бабий**) **Решение на 4 балла**.

Сумма двух чисел будет оканчиваться на 12 в двух случаях: если сумма двух последних разрядов у слагаемых равна 12, либо 112. К примеру:

2508 + 1604 = 4112, т.к. 08 + 04 = 12

3985 + 227 = 4212, т.к. 85 + 27 = 112

Таким образом нужно рассмотреть ситуации, в которых числа имеют окончания от 00 до 12 (00+12, 01+11, 02+10, … 06+06) и окончания от 13 до 99 (13+99, 14+98, 15+97 и т.д.)

Можно было бы взять массив с индексами от 0 до 99, и найти все максимальные числа, которые имеют окончания от 00 до 99, перебрать все возможные пары, и, таким образом, найти максимальную сумму, однако у нас есть еще одно условие – предыдущее по вводу число должно быть больше текущего. В этом случае возникает необходимость запоминать максимум с предыдущих шагов, чтобы при вводе текущего числа можно было проверить, является ли предыдущее число больше текущего.

Решение на 4 балла (язык Pascal):

```
var
  a: array[0..99] of integer;
  i, n, x, t, R: integer;
begin
  readln(n);
  R := 0;
  for i:=0 to 99 do
    a[i] := 0;
  readln(x);
  for i:=1 to n-1 do begin
    if x > a[x mod 100] then
      a[x mod 100] := x;
    readln(x);
```

```
    t := a[(112 - (x mod 100)) mod 100];
    if (x + t > R) and (t > x) then
      R := x + t;
  end;
  writeln(R);
end.
```

123) **Д.Ф. Муфаззалов** (УГАТУ, Уфа) Неэффективное по времени и по памяти решение состоит в сохранении всех элементов последовательности в массив, сортировке массива и выводе элемента, находящегося в середине.

Решение на 2 балла (язык C++):

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
    int n,i,k;
    vector <int> a;
    cin >> n;
    for( i=n; i--; a.push_back(k) )
      cin >> k;
    sort( a.begin(), a.end() );
    cout << a[n/2];
    return 0;
}
```

Можно сортировать не весь массив, а только так, чтобы все числа не меньше медианы оказались справа от нее, или все числа не больше медианы оказались слева от нее. Это можно сделать, например, используя сортировку пузырьком.

Решение на 2 балла (язык Pascal):

```
var
  a: array[1..10000] of byte;
  i, j, n: integer;
begin
  readln(n);
  for i := 1 to n do read(a[i]);
  for i := 1 to n div 2 + 1 do
    for j := 1 to n - i do
      if a[j] > a[j + 1] then swap(a[j], a[j + 1]);
  writeln(a[n div 2 + 1]);
end.
```

В упорядоченной по неубыванию последовательности все числа с одинаковым значением располагаются подряд. Тогда последовательность чисел с одинаковым значением можно заменить парой чисел – этим значением и количеством раз которое оно встретилось.

Воспользуемся тем, что набор различных чисел в последовательности ограниен. Будем хранить не сами числа, а количество раз, которое каждое из них встретилось в последовательности.

Положим значение счетчика равным количеству чисел слева от медианы плюс одно число. Начиная с наименьшего числа, пока значение счетчика не меньше количества раз, которое встретилось текущее число, будем уменьшать счетчик на это количество и

переходить к следующему числу. Когда счетчик станет меньше количества раз, которое встретилось текущее число, медиана будет совпадать со значением этого числа.

Решение на 4 балла (язык C++):

```
#include <iostream>
using namespace std;
int main()
{
    int i,n,a,b[101]={0};
    cin >> n;
    for ( i = 0; i < n; i++ ) {
      cin >> a;
      b[a]++;
      }
    n = n / 2;
    i = 0;
    while( n >= b[i] ) {
      n -= b[i];
      i++;
      }
    cout << i;
    return 0;
}
```

Решение на 4 балла (язык Python, К. Поляков):

```
n = int(input())

b = [0] * 101
for i in range(n):
  a = int(input())
  b[a] += 1

n = n // 2;
i = 0
while n >= b[i]:
  n -= b[i]
  i += 1

print(i)
```

Решение на 4 балла (язык Паскаль, Е.В. Беляева):

```
var // №123 Медиана числовой последовательности
  a: array[0..100] of byte;
  i, j, n, k, sum: integer;
begin
  readln(n);
  for i := 0 to 100 do a[i]:=0;
  for i := 0 to n-1 do begin
    read(k);
    inc(a[k]);
  end;
  sum:=0; k:=0;
  while sum<(n div 2)+1 do begin
      sum:=sum+a[k];
      inc(k);
```

```
  end;
  writeln;
  writeln(k-1);
end.
```

124) **К.М. Багдасарян** (Ковров)

Решение (2 балла, Python):

Считываем все числа в массив. Далее рассматриваем всевозможные пары для определения их НОД Для нахождения НОД используем улучшенный алгоритм Евклида (можно и обычный).

```
n = int(input())
d = [int(input()) for i in range(n)]
maxnod = 0
for i in range(n-1):
    for j in range(i+1,n):
        a = d[i]
        b = d[j]
        while b>0:
            (a,b) = (b,a%b)
        if a > maxnod:
            maxnod = a
print(maxnod)
```

Решение (4 балла, Python):

Создаем массив d из 100 элементов для хранения делителей чисел заданной последовательности. Если число **j** является делителем рассматриваемого числа **a**, то элементы массива **d[j]** и **d[a//j]** увеличиваются на 1.

Поиск делителей оформлен в виде цикла от 2 до корня квадратного от **a**. После обработки всех чисел необходимо пройтись по массиву **d** в обратном порядке для определения максимального НОД (максимальное значение можно было бы искать и в теле основного цикла).

```
s = 100
d = [0]*(s+1)  # элемент d[0] не будем использовать
n = int(input())
for i in range(n):
  a = int(input())
  d[a] += 1;
  m = int(a**(0.5)) + 1
  for j in range(2,m):
    if a % j == 0:
      d[j] += 1   # Если j - делитель, то увеличиваем d[j]
      k = a//j
      if k != j:
        d[k] += 1   # Если j - делитель, то k – тоже
                    #делитель, соответственно увеличиваем d[k]
i = s
while( d[i] < 2 and i > 1):
  i -= 1
print(i)
```

Решение (4 балла, C++):

```
#include <iostream>
```

```
#include <cmath>
using namespace std;
const int s=100;
int main() {
int d[s+1],n,i,j,k,a,m;
for (i=0; i<=s; i++) d[i]=0;
cin >> n;
for (i=0; i<n; i++)
 {
    cin >>a;
    d[a] += 1;
    m = int(sqrt(a));
    for(j=2; j<=m; j++)
    {
     if (a % j == 0)
     {
      d[j]+=1;    // Если j - делитель, то увеличиваем d[j]
      k=a/j;
      if (k!=j)
        d[k]+=1;  // Если j - делитель, то k - тоже делитель,
                  // соответственно увеличиваем d[k]
     }
    }
 }
i=s;
while( d[i]<2 && i>1 ) i-=1;
cout << i;
return 0;
}
```

Решение (4 балла, PascalABC):

```
const s=100;
var
d: array[1..100] of integer;
n,i,j,k,a,m: integer;
begin
 for i:=1 to s do d[i]:=0;
 readln(n);
 for i:=1 to n do
 begin
  read(a);
  inc(d[a]);
  m:=trunc(sqrt(a));
  for j:=2 to m do
  begin
   if a mod j = 0 then
   begin
    inc(d[j]);      // Если j - делитель, то увеличиваем d[j]
    k:=a div j;
    if k<>j then inc(d[k]);
                  // Если j - делитель, то k - тоже делитель,
                  // соответственно увеличиваем d[k]
   end;
  end;
```

```
 end;
 i:=s;
 while (d[i]<2) and (i>1) do i:=i-1;
 writeln(i);
end.
```

125) **(Досрочный ЕГЭ 2020 г.) Решение на 2 балла.** Считываем все числа в массив. Далее рассматриваем всевозможные пары чисел, ищем максимальную сумму среди всех пар, которые удовлетворяют заданному условию: разность чётная, одно из чисел делится на 17.

Решение (2 балла, Python):

```
N = int(input())

A = []
for i in range(N):
  A.append( int(input()) )

a, b = 0, 0
for i in range(N-1):
  for j in range(i+1,N):
    if (A[i]-A[j]) % 2 == 0 and \
       (A[i] % 17 == 0 or A[j] % 17 == 0) and \
       A[i]+A[j] > a+b:
      a, b = A[i], A[j]

print(a, b)
```

Решение на 4 балла. Данные считывать в массив нельзя. Легко понять, что максимальная сумма получается тогда, когда мы берём два максимальных числа, из которых можно образовать допустимую пару. Все числа можно разбить на 4 группы в зависимости от чётности и делимости на 17.

1: чётные, делятся на 17 2: чётные, не делятся на 17

3: нечётные, делятся на 17 4: нечётные, не делятся на 17

Для получения чётной разности нам можно составлять пару из двух чётных чисел или из двух нечётных. Кроме того, одно из них (или даже оба!) должно делиться на 17, то есть в ходить в группу 1 или в группу 3. Таким образом, допустимые комбинации групп:

1 + 1, 1 + 2, 3 + 3, 3 + 4

Если мы берём в сумму числа из одной группы (например, 1+1), то нужно брать максимальное число из группы и «второй максимум» – число, которое стоит на втором месте после сортировки всех чисел этой группы. Конечно, такую сумму можно составить, если оба этих числа ненулевые. Вспомним **алгоритм для поиска двух максимумов**. Первый («главный») обозначим через **M**, второй – через **m**. Если вновь полученное число **x** больше, чем **M**, заменяем m на прошлое значение **M**, а **M** – на **x**:

```
N = int(input(N))
M, m = 0, 0
for i in range(N):
  x = int(input())
  if x > M:
    M, m = x, M
  elif x > m:
    m = x
```

Этот алгоритм нужно повторить 4 раза, для каждой из описанных выше групп чисел. Чтобы не писать много сложенных условных операторов, вместо отдельных переменных **M** и **m** будем

использовать двухмерные массивы. Первый индекс элемента такого массива – это остаток от деления чисел на 2 (0 – для чётных и 1 для нечётных), а второй равен 0 для чисел, которые делятся на 17, и равен 1 для остальных. Таким образом, группа 1 имеет индексы [0][0], группа 2 – индексы [0][1], группа 3 – индексы [1][0] и группа 4 – индексы [1][1].

Сначала обнулим массивы **M** и **m**:

```
M = [[0, 0], [0, 0]]
m = [[0, 0], [0, 0]]
```

Теперь в цикле читаем данные, вычисляем индексы соответствующей группы и используем алгоритм поиска двух максимумов:

```
for i in range(N):
  x = int(input())
  d2 = x % 2
  d17 = 0 if x % 17 == 0 else 1
  if x > M[d2][d17]:
    M[d2][d17], m[d2][d17] = x, M[d2][d17]
  elif x > m[d2][d17]:
    m[d2][d17] = x
```

Теперь остается найти пару с максимальной суммой.

Введем массив **answers** из двух элементов, который содержит отобранную на данный момент подходящую пару с максимальной суммой. Сначала обнулим его:

```
answers = [0, 0]
```

Введём вспомогательную функцию, которая проверяет, что оба числа **a** и **b** возможной пару ненулевые, и, если сумма этой пары больше, чем сумма элементов массива **answers**, обновляет массив **answers**.

```
def check( a, b, answers ):
  if a*b != 0 and (a+b) > sum(answers):
    answers[0] = a
    answers[1] = b
```

Теперь вызовем эту функцию 4 раза. После этого в массиве **answers** останется ответ.

```
check( M[0][0], m[0][0], answers )  # 1 + 1
check( M[0][0], M[0][1], answers )  # 1 + 2
check( M[1][0], m[1][0], answers )  # 3 + 3
check( M[1][0], M[1][1], answers )  # 3 + 4
```

Полная программа на языке Python:

```
N = int(input())

M = [[0, 0], [0, 0]]
m = [[0, 0], [0, 0]]

for i in range(N):
  x = int(input())
  d2 = x % 2
  d17 = 0 if x % 17 == 0 else 1
  if x > M[d2][d17]:
    m[d2][d17] = M[d2][d17]
    M[d2][d17] = x
  elif x > m[d2][d17]:
    m[d2][d17] = x

def check( a, b, answers ):
  if a*b != 0 and (a+b) > sum(answers):
    answers[0] = a
    answers[1] = b
```

```
answers = [0, 0]
check( M[0][0], m[0][0], answers )  # 1 + 1
check( M[0][0], M[0][1], answers )  # 1 + 2
check( M[1][0], m[1][0], answers )  # 3 + 3
check( M[1][0], M[1][1], answers )  # 3 + 4

print( *answers )
```

Программа на языке PascalABC.NET без массива (А. Богданов):

```
// (ai-aj)%2=0 & ai*aj%17=0, ? max({ai+aj})
var m17p,m2p, // max четный & %17, max четный (отличный от первого)
    m17n,m2n, // max нечет & %17, max нечет (отличный от первого)
    i,n,x,L,R:integer;
begin
  (m17p,m2p,m17n,m2n):=(0,0,0,0);
  read(n);
  for i:=1 to n do begin
    read(x);
    if x mod 2 = 0 then begin
      if (x mod 17=0) and (x>m17p) then begin
        if m2p<m17p then m2p:=m17p;
        m17p:=x;
      end else if x>m2p then
        m2p := x;
    end else begin
      if (x mod 17=0) and (x>m17n) then begin
        if m2n<m17n then m2n:=m17n;
        m17n:=x;
      end else if x>m2n then
        m2n := x;
    end;
  end;
  (L,R):=(0,0);
  if (m17p*m2p>0)and(m17p+m2p>L+R) then (L,R):=(m17p,m2p);
  if (m17n*m2n>0)and(m17n+m2n>L+R) then (L,R):=(m17n,m2n);
  print(L,R);
end.
// 5 34 12 51 52 51 => 51 51
```

Ещё один вариант решения на языке PascalABC.NET (А. Богданов):

```
// (ai-aj)%2=0 & ai*aj%17=0, ? max({ai+aj})
var m17,m2:array[0..1] of integer;
    i,n,x,L,R,p,q:integer;
begin
  (m17[0],m2[0],m17[1],m2[1]):=(0,0,0,0);
  read(n);
  for i:=1 to n do begin
    read(x);
    (p,q):=(x mod 2, x mod 17);
    if (q=0) and (x>m17[p]) then begin
      if m2[p]<m17[p] then m2[p]:=m17[p];
      m17[p]:=x;
    end else if x>m2[p] then
      m2[p] := x;
  end;
  (L,R):=(0,0);
```

```
  if (m17[0]*m2[0]>0)and(m17[0]+m2[0]>L+R) then (L,R):=(m17[0],m2[0]);
  if (m17[1]*m2[1]>0)and(m17[1]+m2[1]>L+R) then (L,R):=(m17[1],m2[1]);
  print(L,R);
end.
// 5 34 12 51 52 51 => 51 51
```

Д.Ф. Муфаззалов (УГАТУ, Уфа) Если вновь поступившее число кратно числу 17, то оно может быть в паре с любым числом с такой же четностью, среди тех, что уже обработаны. Если это число не кратно числу 17, то оно может быть в паре только с числом, кратным числу 17, с такой же четностью. Будем хранить 4 числа:

- максимальное четное среди уже обработанных, кратное числу 17;
- максимальное нечетное среди уже обработанных, кратное числу 17;
- максимальное четное среди уже обработанных;
- максимальное нечетное среди уже обработанных.

Каждому новому числу будем выбирать пару в одной из этих категорий, и если пара этих чисел дает большую сумму, чем сумма в данный момент, обновим сумму и пару.

Решение на языке C++:

```
#include <iostream>
using namespace std;
int main()
{
    int b,m1=0,m2=0,n,a,k[2][2]= {0},i,j;
    for (cin>>n; n--;)
    {
        cin>>a;
        i=a%2, j=a%17;
        b=k[i][!j];
        if (a+b>m1+m2 && b) m1=b, m2=a;
        if (!j) k[i][0]=max(a,k[i][0]);
        k[i][1]=max(a,k[i][1]);
    }
    cout<<m1<< ' '<<m2;
    return 0;
}
```

Решение на языке PascalABC.NET 3.0:

```
var
  k: array[boolean, boolean] of integer;
  v, b, m1, m2, n, a: integer;
  i, j: boolean;
begin
  readln(n);
  for v := 1 to n do
  begin
    read(a);
    i := a mod 2 > 0;
    j := a mod 17 = 0;
    b := k[i][not j];
    if (a + b > m1 + m2) and (b > 0) then
    begin
     m1 := b;
     m2 := a
    end;
    if j then k[i][j] := max(a, k[i][j]);
```

```
    k[i][false] := max(a, k[i][false]);
  end;
  writeln(m1, ' ', m2);
end.
```

Решение на языке Python (К. Поляков):

```
n = int(input())

k = [[0, 0], [0, 0]]
m1, m2 = 0, 0
for i in range(n):
  a = int(input())
  r2 = a % 2
  r17 = 0 if a % 17 == 0 else 1
  b = k[r2][1-r17]; # сохранить четность
  if b > 0 and a + b > m1 + m2:
    m1, m2 = a, b
  if r17 == 0:
    k[r2][0] = max(a, k[r2][0])
  k[r2][1] = max(a, k[r2][1])

if m1*m2 > 0:
  print( m1, m2 )
else:
  print( 0, 0 )
```

126) **Д.Ф. Муфаззалов** (УГАТУ, Уфа) Решение методом грубой силы, предполагающее перебор всех возможных перестановок элементов последовательности, рассматривать не будем из-за его крайней неффективности и сложности реализации за отведенное на экзамене время.

Сохраним все числа последовательностей и приведем каждую последовательность к некоторому общему виду, например, к упорядоченному по неубыванию. Сравним все пары элементов двух последовательностей, имеющих одинаковые номера. Если во всех таких парах числа равны, выведем 1, иначе выведем 0. Сложность такого решения по времени в худшем случае составляет $O(N^2)$, сложность по памяти – $O(N)$.

Решение на 2 балла (язык Pascal):

```
var
  n, j, k: word;
  i: boolean;
  a: array[boolean, 1..10000] of word;
begin

 readln(n);
 for j := 1 to n do
   for i := false to true do read(a[i, j]);

 for i := false to true do
   for j := 1 to n do
     for k := 1 to n - j do
       if (a[i, k] < a[i, k + 1]) then
         swap(a[i, k], a[i, k + 1]);

  j := 1;
  while (j < n) and (a[false, j] = a[true, j]) do
```

```
    j := j + 1;
  write(ord(a[false, j] = a[true, j]));
end.
```

Сортировку можно осуществить встроенными средствами языка программирования.

Решение на 2 балла (язык C++):

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
    int n;
    cin >> n;
    vector <int> a[2];
    int i,j,k;
    for(j=0; j<n; j++)
      for(i=0;i<2;i++) {
        cin>>k;
        a[i].push_back(k);
        }
    for(i=0;i<2;i++)
      sort(a[i].begin(), a[i].end());
    for(i=0; i<n-1 && a[0][i]==a[1][i]; i++);
    cout << (a[0][i]==a[1][i]);
    return 0;
}
```

Попробуем обойтись без сортировки. Посчитаем количество раз, которое появлялось каждое число в обеих последовательностях. Если каждое число появлялось одинакое количество раз в обеих последовательностях, их можно сделать одинаковыми.

Решение на 2 балла (язык Python):

```
from collections import Counter
n=int(input())
a,b=[0]*n,[0]*n
for i in range(n):
  r=[int(x) for x in input().split()]
  a[i], b[i]=r[0], r[1]
print(int (Counter(a)==Counter(b)))
```

Сложность такого решения по времени составляет $O(N \cdot \log(N))$, сложность по памяти – $O(N)$. Недостатком этого решения является неэффективность по памяти – все введенные числа сохраняются.

Не будем сохранять последовательности целиком, а, считав очередное число последовательности, увеличим на единицу счетчик количества раз, которое это число встретилось в данной последовательности. Это возможно, так как набор различных чисел ограничен. Если каждое число появлялось одинакое количество раз в обеих последовательностях, их можно сделать одинаковыми.

Решение на 4 балла (язык C++):

```
#include <iostream>
using namespace std;
int main()
{
```

```
    int n,a,i,b[2][1001]={0};
    for(cin>>n; n--;)
      for(i=2; i--; b[i][a]++)  cin>>a;
    for(i=0; i<1000 && b[0][i]==b[1][i]; i++);
    cout<<(b[0][i]==b[1][i]);
    return 0;
}
```

Решение на 4 балла (язык Pascal):

```
var
  b: array[1..2, 1..1000] of integer;
  n, a, i, j: integer;
begin
  readln(n);
  for j := 1 to n do
    for I := 1 to 2 do
    begin
      read(a);
      b[i, a] := b[i, a] + 1;
    end;
  i := 1;
  while (i < 1000) and (b[1][i] = b[2][i]) do i := i + 1;
  writeln(ord(b[2][i] = b[1][i]));
end.
```

Можно хранить количество не всех возможных чисел, а только тех, которые были введены.

Тогда придется потратить дополнительное время на проверку того, что поступившее число было введено ранее. Сложность такого решения по времени $O(N \cdot \log N)$.

Решение на 2 балла (язык C++):

```
#include <iostream>
#include <map>
using namespace std;
int main()
{
    int n,ans=1,a,i;
    typedef map <int,int> m;
    m::iterator t;
    m b[2];
    for(cin>>n; n--; )
        for (i=0; i<2; i++)
        {
            cin>>a;
            if (b[i].find(a)==b[i].end())
                b[i][a]=0;
            b[i][a]++;
        }
    for (t=b[0].begin(); t!=b[0].end()&& ans; t++)
        ans=t->second==b[1][t->first];
    cout<<ans;
    return 0;
}
```

Решение на Python на 4 балла с использованием словарей (**А. Куриленко**)

```
cnt1 = { }
cnt2 = { }
N = int(input())
for i in range(N):
  a, b = map(int, input().split())
  cnt1[a] = cnt1.get(a, 0) + 1
  cnt2[b] = cnt2.get(b, 0) + 1

for k in cnt1:
  if cnt2.get(k, 0) != cnt1[k]:
    print(0)
    break
else:
  print(1)
```

Решение на PascalABC.NET на 4 балла (**А. Фахуртдинова, Москва**)

Для решения задачи достаточно завести одномерный массив длины 1001 (k[0..1000]). При считывании очередной пары входных данных **(x,y)** элемент массива **k[x]** наращивается на 1, элемент **k[y]** уменьшается на 1. Затем требуется проанализировать элементы массива **k**, если все элементы будут равны 0, значит можно составить одинаковые последовательности, если хотя бы один элемент не будет равен 0, значит нельзя.

```
var k:array[0..1000] of integer;
n,i,x,y,f :integer;
begin
readln(n);
  for i:=0 to 1000 do k[i]:=0;
  for i:=1 to n do begin
    readln(x,y);
    k[x]:=k[x]+1;
    k[y]:=k[y]-1
  end;
  f:=1;
  for i:=0 to 1000 do
    if k[i]<>0 then f:=0;
  writeln(f)
end.
```

127) **А. Богданов** (Danov) Решение полным перебором предполагает перебор всех пар с фильтрацией пар по расстоянию и четности.

Решение на 2 балла (Pascal)

```
var a:array[1..10000] of integer;
    i,j,k,n,s:integer;
begin
  read(n);
  for i:=1 to n do read(a[i]);
  k:=0;
  for i:=1 to n-1 do
    for j:=i+1 to n do begin
      s := a[i]+a[j];
      if (j-i>=3) and (s mod 2 = 1) or
         (j-i>=4) and (s mod 2 = 0) then
        k += 1;
```

```
    end;
  print(k);
end.
```

Можно перебирать правые элементы от расстояния 3 и более, тогда условие фильтра во внутреннем цикле можно записать немного короче, исключив проверку расстояния на 3 и более.

Решение на 2 балла (Pascal)

```
var a:array[1..10000] of integer;
    i,j,k,n,s:integer;
begin
  read(n);
  for i:=1 to n do read(a[i]);
  k:=0;
  for i:=1 to n-3 do
    for j:=i+3 to n do begin
      s := a[i]+a[j];
      if (s mod 2 = 1)or (j-i>=4) and (s mod 2=0) then
        k += 1;
    end;
  print(k)
end.
```

Далее, если заметить, что с расстояния 4 и более мы считаем все пары, как с четной, так и с нечетной суммой, можно упростить решение и получить решение на 3 балла, с хранением всех элементов (теряем 1 балл), но с линейной сложностью алгоритма - O(N).

Решение на 3 балла (Pascal)

```
var a:array[1..10000] of integer;
    i,j,k,n,s:integer;
begin
  read(n);
  for i:=1 to n do read(a[i]);
  k:=0;
  for i:=1 to n-3 do
    k += (a[i]+a[i+3]) mod 2 + (n-(i+4)+1);
  print(k);
end.
```

Уже из этого решения можно заметить, что в основном цикле мы однократно проходимся по массиву и, соответственно, можно предложить решение с буфером. Если делать решение в лоб, строго по условию задачи, получим достаточно большое решение на 4 балла:

Решение на 4 балла (Pascal)

```
var b:array[0..3] of integer;
    i,j,k,k0,k1,n,x:integer;
begin
  read(n);
  for i:=0 to 3 do read(b[i]);
  k := (b[0]+b[1]) mod 2; // первая пара j-i=3 внутри буфера
  (k0,k1):=(0,0);
  for i:=4 to n-1 do begin
    if b[0] mod 2 =0 then k0+=1 else k1+=1;
    b[0]:=b[1]; b[1]:=b[2]; b[2]:=b[3];
```

```
    read(x); b[3]:=x;
    if x mod 2=0 then k+=k0 else k+=k1; // задумайтесь
    if x mod 2=1 then k+=k0 else k+=k1; // над этой парой строк
    k += (b[0]+x) mod 2;
  end;
  print(k);
end.
```

Но это решение можно сократить и прийти к решению похожему на предыдущее. Если подумать, то на расстоянии 4 и более мы считаем все пары безусловно. На этом расстоянии пара будет либо с нечетной суммой на расстоянии 3 и более, либо с четной на расстоянии 4 и более. Кол-во этих пар можно вычислить вне цикла. А в цикле подсчитать только пары на расстоянии ровно 3 с нечетной суммой. Итоговый код:

Решение на 4 балла (компактное, Pascal):

```
var b:array[1..4] of integer;
    i,j,k,n:integer;
begin
  read(n);
  for i:=1 to 3 do read(b[i]);
  k := (n-3)*(n-4)div 2;
  for i:=4 to n do begin
    read(b[4]);
    k+=(b[1]+b[4])mod 2;
    for j:=1 to 3 do b[j]:=b[j+1];
  end;
  print(k);
end.
```

Решение на 4 балла (Python):

```
n = int(input())

b = [0, 0, 0, 0]
for i in range(3):
  b[i] = int(input())

k = (n-3)*(n-4) // 2

for i in range(3,n):
  b[3] = int(input())
  k += (b[0] + b[3]) % 2
  for j in range(3):
    b[j] = b[j+1]

print(k)
```

128) **А. Богданов** (Danov) Для неэффективного решения удобно сохранить всё в массиве. Найти индексы первого минимального и последнего равного ему. Затем запустить перебор пар, состоящих из левого элемента до первого индекса и правого после последнего.

Решение на 2 балла (Pascal)

```
var a:array[1..10000] of integer;
    i,j,iL,iR,s,n:integer;
begin
  read(n);
  iL := 2;
```

```
  s := 0;
  for i:=1 to n do read(a[i]);

  for i:=2 to n-1 do
    if a[iL]>a[i] then (iL,iR) := (i,i)
    else if a[iL]=a[i] then iR := i;

  for i:=1 to iL-1 do
    for j:=iR+1 to n do
      if a[i]+a[j]>s then
        s := a[i]+a[j];
  print(s);
end.
```

В первом алгоритме можно заметить, что поиск максимального левого элемента и максимального правого производится независимо и может быть реализован последовательно. Таким образом, сложность алгоритма становится линейной и решение оценивается в 3 балла. 1 балл снимается за хранение всех элементов.

Решение на 3 балла:

```
var a:array[1..10000] of integer;
    i,j,iL,iR,L,R,n:integer;
begin
  read(n);
  for i:=1 to n do read(a[i]);

  iL := 2;
  for i:=2 to n-1 do
    if a[iL]>a[i] then (iL,iR) := (i,i)
    else if a[iL]=a[i] then iR := i;

  L := a[1];
  for i:=2 to iL-1 do
    if L<a[i] then L := a[i];
  R := a[iR+1];
  for i:=iR+2 to n do
    if R<a[i] then R := a[i];

  print(L+R);
end.
```

Предыдущий алгоритм содержит четыре прохода, которые можно объединить в один проход без хранения всех элементов. Последовательность чисел делится на три части. До первого минимального (L - максимальный слева), между минимальными mn (M - максимальный средний), правее последнего минимального (R - максимальный справа). Покажем это схемой: *L*mn*M*mn*R* или *L*mn*R*

Решение на 4 балла:

```
var i,j,n,x,mn,L,M,R:integer;
begin
  (M,R) := (0,0);
  read(n,L,mn);
  for i:=3 to n-1 do begin
    read(x);
    if x<mn then begin
      if L<mn then L:=mn;
```

```
      if L<M then L:=M;
      if L<R then L:=R;
      (M,R):=(0,0);
      mn := x
    end else
      if x>R then R:=x;
    if x=mn then begin
      if M<R then M:=R;
      R:=0;
    end;
  end;
  read(x);
  if x>R then R:=x;
  print(L+R);
end.
```

Решение на 4 балла (аналогичное предыдущему и компактное, за счет применения функции max и кортежей)

```
var i,n,x,mn,L,M,R:integer;
begin
  (M,R) := (0,0);
  read(n,L,mn);
  for i:=3 to n-1 do begin
    read(x);
    if x<mn then begin
      L := max(L,mn,M,R);
      (mn,M,R):=(x,0,0);
    end else
      R:=max(x,R);
    if x=mn then
      (M,R):=(max(M,R),0);
  end;
  read(x);
  print(L+max(R,x));
end.
```

Решение на 4 балла (Python):

```
n = int(input())
L = int(input())
mn = int(input())

M, R = 0, 0
for i in range(3,n):
   x = int(input())
   if x < mn:
     L = max(L, mn, M, R)
     mn, M, R = x, 0, 0
   else:
     R = max(x,R)
   if x == mn:
     M, R = max(M,R), 0

x = int(input())
print( L + max(R,x) )
```

Решение на 4 балла (Д. Муфаззалов, Уфа)

Будем хранить 4 числа:

1. минимальное число из уже обработанных (mi),
2. максимум из тех чисел, что были ранее минимального (L),
3. максимум из тех, что были позднее минимального (R),
4. максимум из уже обработанных чисел (ma).

Считаем первый элемент последовательности, это L. Считаем второй элемент, это mi. Вычислим ma. Зададим R значением меньше минимального возможного (нулем). Получив каждое следующее число, будем обновлять R; если поступившее число меньше или равно mi, то R будет среди последующих чисел, поэтому зададим его нулем; если поступившее число меньше mi, обновим mi и L (L=ma). Обновим ma.

Последний элемент последовательности считаем отдельно, он может повлиять только на R.

Решение на C++:

```
#include <iostream>
int n,num,L,R,mi,i,ma;
using namespace std;
int main()
{
    cin >> n >> L >> mi;
    for( ma = max(L,mi),i=3; i < n; i++ )
    {
        cin >> num;
        R = max(R, num);
        if( num <= mi )
        {
            R = 0;
            if( num < mi )
            {
                mi = num;
                L = ma;
            }
        }
        ma = max( ma, num );
    }
    cin >> num;
    cout << L + max(R,num);
    return 0;
}
```

129) Для неэффективного решения удобно сохранить всё в массиве. Затем выполняем перебор всех возможных пар, и увеличиваем счётчик каждый раз, когда очередная пара соответствует условию задачи.

Возможные подводные камни:

а) чтобы избежать повторов, будем рассматривать только пары **a[i]-a[j]**, где **j>i**, поэтому второй цикл нужно начинать не с 0, а с **i+1**

```
for i in range(n-1):
    for j in range(i+1, n)
```

б) при проверке условия делимость разности эту разность обязательно взять в скобки, иначе сначала выполнится операция взятия остатка, а потом уже вычитание:

```
   if (a[i]-a[j]) % m == 0 and ...
```

Решение на 2 балла (Python)

```
m = 60
b = 80

n = int(input())
a = [0]*n
for i in range(n):
   a[i] = int(input())

count = 0
for i in range(n-1):
   for j in range(i+1, n):
      if (a[i]-a[j]) % m == 0 and \
         (a[i] > b or a[j] > b):
         count += 1

print( count )
```

Для построения эффективного решения нужно сообразить, что разность двух чисел делится на **m** тогда и только тогда, когда оба числа имеют ОДИНАКОВЫЕ остатки от деления на **m**.

Будем использовать метод динамического программирования. Пусть мы прочитали из входного потока очередное число **x**, и остаток от его деления на **m** равен **p**. С какими предыдущими числами оно может образовать пару?

Если **x>b**, то оно образует пару со ВСЕМИ предыдущими числами, у которых такой же остаток **p**. Если x<=b, то такое число образует пару только с теми числами, имеющими остаток **p**, которые больше, чем **b**. Поэтому нам нужно для всех возможных остатков **p** (от 0 до **m-1**) запоминать

а) количество предыдущих чисел, имеющих остаток **p** (**p**-чисел), которые больше, чем **b**; заведём для этого массив **A[0..m-1]**;

б) количество ВСЕХ предыдущих **p**-чисел, эти значения будем хранить в массиве **a[0..m-1]**.

Получив очередное число **x**, нам нужно найти остаток от его деления на **m**:

```
   p = x % m
```

проверить, сравнить его с **b** и выбрать один из вариантов увеличения счётчика **count**:

```
   if x > b:
     count += a[p] # все предыдущие с тем же остатком
   else:
     count += A[p] # только те, которые больше, чем b
```

После этого величиваем на 1 соответствующий счётчик в массиве:

```
   if x > b:
     A[p] += 1  # увеличиваем только для x > b
   a[p] += 1    # увеличиваем всегда
```

Естественно, что два условных оператора с одинаковыми условиями можно совместить.

Полная программа (Python 3):

```
m = 60
b = 80
```

```
n = int(input())

a = [0]*m
A = [0]*m
count = 0
for i in range(n):
  x = int(input())
  p = x % m
  if x > b:
    count += a[p] # все предыдущие с тем же остатком
    A[p] += 1
  else:
    count += A[p] # только те, которые больше, чем b
  a[p] += 1

print( count )
```

Комбинаторный метод на языке Python (А. Куриленко):

```
m, b = 60, 80
cnt = [[0, 0] for i in range(m)]

N = int(input())
for i in range(N):
    x = int(input())
    cnt[x % m][x > b] += 1

ans = 0
for r in range(m):
    ans += (cnt[r][0] * cnt[r][1] +
            cnt[r][1] * (cnt[r][1] - 1) // 2)

print(ans)
```

Программа на языке C++ (Д.Ф. Муфаззалов)

```
#include <iostream>
using namespace std;
int m,n,i,a[60][2],s,r;
int main()
{
    for (cin>>n, i=0; i<n; i++)
    {
        cin>>m; // вводим новое число
        r=m%60; // остаток от деления на 60
        m=m>80; // определяем, больше ли число, чем 80
        s+=a[r][m]; добавляем к ответу количество пар с этим числом
        a[r][0]+=m;// только те, которые больше b
        a[r][1]++; // все предыдущие с тем же остатком
    }
    cout<<s;
    return 0;
}
```

Программа на языке PascalABC.NET (Н. Чурсин)

```
const m = 60; b = 80;
var
  i, n, k, x, p: integer;
```

```
  a, c: array[0..m - 1] of integer;// a - количество чисел > b,
                                   // c - количество всех чисел
begin
  readln(n);
  for i := 0 to m - 1 do begin
    a[i] := 0;
    c[i] := 0;
  end;
  for i := 1 to n do begin
    readln(x);
    p := x mod m;
    if x > b then begin
      k := k + c[p]
      a[p] := a[p] + 1;
    end
    else
      k := k + a[p];
    c[p] := c[p] + 1;
  end;
  writeln(k);
end.
```

130) (**Муфаззалов Д.Ф.**) Сохраним все элементы последовательности в массив, переберем все возможные пары и посчитаем количество тех из них, сумма элементов которых равна числу 20. Чтобы избежать двойного счета и не рассматривать пары, в которые входит один элемент последовательности дважды, будем рассматривать только такие пары, в которых номер первого элемента меньше номера второго элемента.

Решение на 2 балла, язык Pascal, метод грубой силы

```
var
  b: array[1..10000] of integer;
  i, n, a, j: integer;
begin
  readln(n);
  for i := 1 to n do readln(b[i]);
  for i := 1 to n do
    for j := i + 1 to n do
      if b[i] + b[j] = 20 then a := a + 1;
  writeln(a);
end.
```

Глобальные переменные обнуляются, поэтому дополнительная инициализация счетчика не требуется.

Для эффективного решения заметим, что среди положительных чисел сумму, равную числу 20, могут составить только числа, меньшие двадцати. При этом возможны только такие пары:

1+19

2+18

… (1)

9+11

10+10.

Посчитаем, сколько раз встретилось каждое такое число. Для этого понадобится массив счетчиков на 19 элементов. Тогда количество пар для каждой строки в (1) кроме последней равно произведению количества раз, которое встретилось первое слагаемое, на количество раз, которое встретилось второе слагаемое. Количество пар для последней сроки равно половине произведения количества раз, которое встретилось число 10, на это же количество без одного.

Решение на 4 балла, язык C++, комбинаторный метод

```
#include <iostream>
using namespace std;
int n,m,i,a,b[20];
int main()
{
    cin>>n;
    a = 0;
    for (i=0; i<n; i++)
    {
        cin>>m;
        if (m<20) b[m]++;
    }
    for (i=1;i<10;i++) a+=b[i]*b[20-i];
    a+=b[10]*(b[10]-1)/2;
    cout<<a;
    return 0;
}
```

Глобальные переменные обнуляются, поэтому дополнительная инициализация суммы и массива счетчиков не требуется.

Задачу можно решить методом динамического программирования. По мере поступления каждого элемента последовательности будем считать, сколько раз встретилось каждое число меньше числа 20. Тогда количество пар с этим поступившим элементом равно количеству раз, которое встретилось на данный момент парное ему число из (1).

Решение на 4 балла, язык C++, метод динамического программирования

```
#include <iostream>
using namespace std;
int n,m,i,a,b[20];
int main()
{
    cin>>n;
    a = 0;
    for (i=0; i<n; i++)
    {
        cin>>m;
        if (m<20)
        {
            a+=b[20-m];
            b[m]++;
        }
    }
    cout<<a;
    return 0;
}
```

Аналогичное решение без использования оператора условия:

```
#include <iostream>
using namespace std;
int n,m,i,a,b[20];
int main()
{
    a = 0;
    for (cin>>n, i=0; i<n; i++)
    {
        cin>>m;
        a+=b[max(20-m,0)];
        b[(m<20)*m]+=m<20;
    }
    cout<<a;
    return 0;
}
```

Решение на языке PascalABC.NET (Н. Чурсин)

```
var
  i, n, x, k: integer;
  a: array[1..19] of integer;
begin
  readln(n);
  k := 0;
  for i := 1 to 19 do
    a[i] := 0;
  for i := 1 to n do begin
    readln(x);
    if x < 20 then begin
      k := k + a[20 - x];
      a[x] := a[x] + 1;
    end
  end;
  writeln(k);

end.
```

131) (**Кабанов А.М.**) **Решение на 2 балла**. Сохраним все элементы последовательности в массив, переберем все возможные пары, в которых номер первого элемента меньше номера второго элемента и посчитаем количество тех из них, где первый элемент больше второго и сумма элементов не превышает 34.

Решение на языке Python

```
n=int(input())
a=[]
for i in range(n):
    a.append(int(input()))
k=0
for i in range(n-1):
    for j in range(i+1,n):
        if a[i]>a[j] and a[i]+a[j]<=34:
            k=k+1
print(k)
```

Решение на языке Pascal

```
var a: array[1..10000] of integer;
```

```
    i, n, k, j: integer;
begin
  readln(n);
  for i := 1 to n do readln(a[i]);
  k:=0;
  for i := 1 to n-1 do
    for j := i + 1 to n do
      if (a[i]>a[j]) and (a[i]+a[j]<=34) then k := k + 1;
  writeln(k);
end.
```

Решение на 4 балла. По мере поступления каждого элемента последовательности будем считать в массиве a, сколько раз встретилось каждое число меньше числа 34, при этом **a[i]** будет обозначать количество чисел **i**. Каждое вновь введённое число **x** будет образовать подходящие пары с числами, которые с одной стороны больше **x**, а с другой не более чем **34-x**. Подсчитав количество чисел в этом диапазоне, получим необходимое количество пар.

Решение на языке Python

```
n=int(input())
a=[0]*34
k=0
for i in range(n):
    x=int(input())
    if x<34:
        for j in range(x+1,34-x+1):
          k+=a[j]
        a[x]+=1
print(k)
```

Решение на языке Pascal

```
var a: array[1..33] of integer;
    i, n, k, j, x: integer;
begin
  readln(n);
  k:=0;
  for i := 1 to n do
  begin
    readln(x);
    if x < 34 then
    begin
      for j := x+1 to 34-x do
        k:=k+a[j];
      a[x]:=a[x]+1;
    end;
  end;
  writeln(k);
end.
```

132) (**Муфаззалов Д.Ф.**) Сохраним все элементы последовательности в массив, переберем все возможные тройки и посчитаем количество тех из них, сумма элементов которых равна числу 20. Чтобы избежать двойного счета и не рассматривать тройки, в которые элемент последовательности входит более одного раза, будем рассматривать только такие тройки, в которых номер первого элемента меньше номера второго элемента, а номер второго элемента меньше номера третьего элемента.

Решение на 2 балла, язык C++, метод грубой силы

```
#include <iostream>
using namespace std;
int n,m,b[10000],a,i,j,k;
int main()
{
    cin>>n;
    for(i=0; i<n; i++) cin>>b[i];
    for (i=0; i<n; i++)
        for (j=i+1; j<n; j++)
            for(k=j+1; k<n; k++)
                if (b[i]+b[j]+b[k]==20)  a++;
    cout<<a;
    return 0;
}
```

Глобальные переменные обнуляются автоматически, поэтому дополнительная инициализация счетчика не требуется.

Для эффективного решения заметим, что три положительных числа могут составить сумму, равную числу 20, только если эти числа меньше девятнадцати. При этом возможны только такие тройки (с точностью до набора значения в тройке):

1+1+18

1+2+17

…

1+9+10

2+3+15

2+4+14

…

2+8+8

… (1)

6+7+7.

Будем называть такие тройки разбиением числа 20 на три слагаемых.

Посчитаем, сколько раз встретилось каждое такое число. Для этого понадобится массив счетчиков на 18 элементов. Тогда количество троек для каждой строки в (1) кроме тех, в которых есть одинаковые слагаемые, равно произведению количеств раз, которое встретилось каждое слагаемое. Количество троек для других строк равно половине произведения количества раз, которое встретилось повторяющееся число, на это же количество без одного и на количество раз, которое встретилось неповторяющееся число. Заметим, что первое слагаемое не может быть равно третьему и все три слагаемых равны между собой быть не могут.

Сгенерируем все разбиения из (1) перебором.

Первые два слагаемых будем перебирать среди всех возможных пар их значений, а третье будем находить из условия равенства суммы двадцати. Будем следить, чтобы каждое следующее слагаемое было не меньше предыдущего.

Решение на 4 балла, язык C++, комбинаторный метод , генерация перебором

```
using namespace std;
int n,m,b[19],a,i,j,k;
int main()
{
    cin>>n;
    for(i=0; i<n; i++)
    {
        cin>>m;
        if (m<19) b[m]++;
    }
    for (i=1; i<7; i++)// первое слагаемое
    {
     for (j=i+1; j<10; j++) // второе слагаемое
     {
      k=20-i-j;//третье слагаемое
       if (k==j) a+=b[i]*(b[k]-1)*b[k]/2;//тройки с одинаковыми
                                  //    вторым и третьим числами
          else
         if (k>j) a+=b[i]*b[j]*b[k];// тройки без одинаковых чисел
     }
     a+=b[i]*(b[i]-1)/2*b[20-i-i]; //тройки с одинаковыми
                                       //вторым и первым числами
    }
    cout<<a;
    return 0;
}
```

Глобальные переменные обнуляются автоматически, поэтому дополнительная инициализация счетчиков не требуется.

Разбиения из (1) можно сгенерировать рекурсивно.

Поставим на первую позицию такого разбиения число 0 и применим рекурсивный алгоритм:

Если не все слагаемые в разбиении найдены и введенных чисел досточно, для составления уже имеющейся части разбиения:

1) поставим на следующую позицию то же число, что и на текущей, и вызовем рекурсивный алгоритм;
2) найдем количество подразбиений до текущего слагаемого, которые можно составить из введенных чисел. Для этого умножим количество подразбиений до позиции, предшествующей той, где первый раз встретилось текущее слагаемое, на число сочетаний из количества раз, которое встретилось текущее слагаемое при вводе, по количество раз, которое встретилось это слагаемое в разбиении (2).
3) поставим на следующую позицию все возможные другие числа и вызовем рекурсивный алгоритм.

Если же все слагаемые в разбиении найдены, посчитаем количество таких разбиений, как в (2), и увеличим ответ на это количество.

Разбиения с нулевыми слагаеммыми будут отбрасываться, как только появится такое слагаемое, так как их невозможно составить из введенных чисел.

Решение на 4 балла, язык C++, комбинаторный метод, рекурсивная генерация разбиений

```
#include <iostream>
#define e 20
```

```
#define R 3
#define L long long
using namespace std;
L c(L n,L m) // число сочетаний из N по M
{   L p=1,i;
    for (i=n-m+1; i<=n; i++) p*=i;
    for (i=2; i<=m; i++) p/=i;
    return p;
}
L n,m,b[e],i,ans;
void f(L m, L k, L h, L s, L a)
{
    if (m*a) // если не все слагаемые в разбиении расставлены и
             // достаточно чисел для этого разбиения
    {   f(m-1,k,h+1,s+k,a); // ставим на следующую позицию то же
                            //число (k), что и на текущей
        a*=c(b[k],h); // находим количество подразбиений
                              // до текущего слагаемого
                            // ставим на следующую позицию другие
                            //возможные числа (k)
        for (k++; k<=e-s; k++) f(m-1,k,1,s+k,a);
    }
    else ans+=a*c(b[k],h)*(s==e);
     // если все слагаемые в разбиении расставлены,
     // находим количество таких разбиений
}
int main()
{   cin>>n;
    for(i=0; i<n; i++)
    {
        cin>>m;
        if (m<e) b[m]++; // находим количество появлений каждого
                         // нужного числа
    }
    f(R,0,0,0,1); // ставим на первую позицию в разбиении 0 и
                  //запускаем генерацию разбиений
    cout<<ans;
    return 0;
}
```

Глобальные переменные обнуляются автоматически, поэтому дополнительная инициализация счетчиков не требуется.

Заметим, что это решение является универсальным по длине разбиения, то есть может использоваться без изменения программы при разных ее значениях, чего нельзя сказать о переборном алгоритме с циклами. При R равном 2, эта программа является и решением задачи 130.

Более простое для реализации решение получится, если находить количество троек, которое может составить каждое вновь поступившее число с числами, введенными до него (динамическое программирование).

Для этого понадобится знать, сколько раз встретилось каждое из возможных чисел. Если поступившее число больше восемнадцати, то оно не может составить ни одну тройку.

Если число меньше девятнадцати, то оно может составить тройку со всеми возможными парами, составленными из поступивших ранее чисел и дающих нужную сумму.

Если поступившее число нечетное, то оно не может образовать тройку с двумя одинаковыми числами, тогда количество таких троек равно произведению количеств раз, которое встретилось каждое число пары (комбинаторный метод). Если поступившее число четное, то оно образует тройку и с двумя одинаковыми числами (при этом значение этих одинаковых чисел максимальное среди всех значений в парах), тогда количество троек равно половине произведения количества раз, которое встретилось это повторяющееся число, на это же количество без одного.

Переберем все возможные пары, подходящие для поступившего числа, и сложим количество троек, которые эти пары с ним образуют.

Необходимо следить, чтобы второе число пары было не меньше первого. Это достигается ограничением перебора значения первого числа пары.

Решение на 4 балла, язык C++, комбинированный (динамическое программирование и комбинаторика) метод

```
#include <iostream>
using namespace std;
int n,m,b[19],a,i,j;
int main()
{
    cin>>n;
    for(i=0; i<n; i++)
    {
        cin>>m;
        if (m<19)
        {
            for (j=1; j<(21-m)/2; j++) a+=b[j]*b[20-m-j];
            if (m%2==0) a+=b[j]*(b[j]-1)/2;
            b[m]++;//увеличиваем счетчик для поступившего числа
        }
    }
    cout<<a;
    return 0;
}
```

Глобальные переменные обнуляются автоматически, поэтому дополнительная инициализация счетчиков не требуется.

Другое решение состоит в полностью динамическом нахождении количества троек, которое образует вновь поступившее число с ранее введенными числами. Оно равно количеству пар, сумма элементов которых с поступившим числом равна двадцати.

При поступлении каждого числа будем обновлять количество пар, которое оно может составить с введенными ранее числами. Это количество пар нужно хранить в массиве на девятнадцать элементов.

Поступившее число образует пары с ранее введенными числами, поэтому нужно хранить количество каждого из таких чисел в массиве на двадцать элементов.

Решение на 4 балла, язык C++, метод динамического программирования

```
#include <iostream>
using namespace std;
int n,m,b[20],c[19],a,i,j;
```

```
int main()
{
    cin>>n;
    for(i=0; i<n; i++)
    {
        cin>>m;
        if (m<19)
        {
            a+=b[20-m];//количество троек
            for(j=1;j<20-m;j++) b[j+m]+=c[j];//количество пар
            c[m]++; //увеличиваем счетчик для поступившего числа
        }
    }
    cout<<a;
    return 0;
}
```

Глобальные переменные в C++ обнуляются автоматически, поэтому дополнительная инициализация счетчиков не требуется.

Решение на 4 балла, язык PascalABC.NET (Е.В. Беляева, г. Ульяновск)

```
var n,m,a,i,j,k:integer;
    b:array[0..19] of integer;
begin
  readln(n);
  for i:=0 to n-1 do b[m]:=0;
  for i:=0 to n-1 do
      begin
        readln(m);
        if m<19 then inc(b[m]);
      end;
   for i:=1 to 6 do
   begin
     for j:=i+1 to 9 do
     begin
       k:=20-i-j;
        if k=j then a:=a+trunc(b[i]*(b[k]-1)*b[k]/2)
          else
             if k>j then a:=a+b[i]*b[j]*b[k];
     end;
     a:=a+trunc(b[i]*(b[i]-1)/2*b[20-i-i]);
   end;
   writeln(a);
end.
```

133) (**Муфаззалов Д.Ф.**) Сохраним все элементы последовательности в массив, переберем все возможные пары и посчитаем количество тех из них, сумма элементов которых больше числа 20. Чтобы избежать двойного счета и не рассматривать пары, в которые входит один элемент последовательности дважды, будем рассматривать только такие пары, в которых номер первого элемента меньше номера второго элемента.

Решение на 2 балла, язык Pascal, метод грубой силы

```
var a:array[1..10000] of integer;
ans,i,n,j:integer;
begin
  readln(n);
```

```
  for i:=1 to n do readln(a[i]);
  for i:=1 to n do
    for j:=i+1 to  n do
      if a[i]+a[j]>20 then ans:=ans+1;
    writeln(ans);
end.
```

Глобальные переменные обнуляются автоматически, поэтому дополнительная инициализация счетчика не требуется.

Для эффективного решения заметим, что если число больше 19, то оно может быть в паре с любым числом, чтобы их сумма была больше 20.

Если же число (m) меньше 20 (назовем их «маленькими», а остальные «большими»), то оно должно быть в паре с числом, большим чем (20-m). Найдем количество во входной последовательности каждого из чисел меньше 20 в элементах массива с номерами от 1 до 19, количество чисел больше 19 в элементе массива с номером 0 и расмотрим все такие комбинации.

Решение на 4 балла, язык Pascal, комбинаторный метод

```
var
  a: array[0..19] of integer;
  ans, i, n, m, j: integer;

begin
  readln(n);
  for i := 1 to n do
  begin
    readln(m);
    if m < 20 then a[m] := a[m] + 1 else a[0] := a[0] + 1;
  end;
  ans := a[0] * (n - a[0]); //большие числа со всеми остальными
  ans :=ans+ a[0] * (a[0]-1) div 2; //большие числа с собой
  for i := 0 to 19 do
    for j := max(i + 1, 21 - i) to 19 do
       ans := ans + a[i] * a[j]; //пары для маленьких чисел
  for i := 11 to 19 do ans := ans + a[i] * (a[i] - 1) div 2;
                                  //пары одинаковых маленьких чисел
  writeln(ans);
end.
```

Глобальные переменные обнуляются автоматически, поэтому дополнительная инициализация счетчиков не требуется.

Можно считать возможные пары для каждого поступившего числа среди введенных ранее. Для этого будем хранить количество каждого из чисел меньше 20 и чисел больше 19 среди введенных ранее чисел. Логика формирования пар такая же, как в предыдущем решении.

Решение на 4 балла, язык C++, метод динамического программирования

```
#include <iostream>
using namespace std;
int n,m,i,a,b[20],j;
int main()
{
    cin>>n;
```

```
    for (i=0; i<n; i++)
    {
        cin>>m;
        for (j=max(1,21-m); j<20; j++)  a+=b[j];
        a+=b[0];
        b[(m<20)*m]++;
    }
    cout<<a;
    return 0;
}
```

Глобальные переменные обнуляются автоматически, поэтому дополнительная инициализация счетчиков не требуется.

Решение на 4 балла, язык Python, метод динамического программирования

```
n=int(input())
b=[0]*20
ans=0
for i in range(n):
    m=int(input())
    for j in range(max(1,21-m),20):
       ans+=b[j]
    ans=ans+b[0]
    r=(m<20)*m
    b[r]=b[r]+1
print(ans)
```

Решение на 4 балла, язык Python метод динамического программирования (К.Ю. Поляков)

Можно рассуждать иначе: найти число всех возможных пар (оно равно **n*(n-1)/2**, где **n** – количество чисел) и вычесть из него число пар, сумма которых меньше или равна 20. В элементе массива **b[i]** будем хранить количество предыдущих элементов, которые равны **i**. Если очередной элемент **x**, прочитанный из входного потока, меньше, чем 20, количество пар с суммой <= 20, которые он может образовать с предыдущими элементами, равно **b[1]+b[2]+...+b[20-x]**.

```
n = int(input())

b = [0]*20
count20 = 0
for i in range(n):
  x = int(input())
  for j in range(1,21-x):
    count20 += b[j]
  if x < 20:
    b[x] += 1

print( n*(n-1)//2 - count20 )
```

Решение на 4 балла, язык PascalABC.NET, метод динамического программирования (А. Богданов)

```
var s:array[1..20] of integer;
    i,j,k,n,x:integer;
begin
  for i:=1 to 20 do s[i]:=0;
  read(n);
  k:=0;
```

```
  for i:=1 to n do begin
    read(x);
    for j:=max(1,21-x) to 20 do
      k+=s[j];
    s[min(x,20)]+=1;
  end;
  print(k);
end.
```

134) (**А.С. Юсуфов**) Сохраним все элементы последовательности в массив, переберем все возможные пары и посчитаем количество тех из них, сумма элементов которых больше числа 20. Чтобы избежать двойного счета и не рассматривать пары, в которые входит один элемент последовательности дважды, будем рассматривать только такие пары, в которых номер первого элемента меньше номера второго элемента.

Решение на 2 балла, язык C++

```
#include <iostream>
using namespace std;
int main() {
  int n; int a[10000];
  int l = 0; int r = 0;
  cin >> n;
  for (int i = 0; i < n; i++)
       cin >> a[i];
  for (int i = 0; i < n - 1; i++)
       for (int j = i + 4; j < n; j++)
            if (a[i] + a[j] > l + r)
                 if (a[i] % 13 == 0 || a[j] % 13 == 0) {
                      l = a[i]; r = a[j];
                 }
  if (l > 0 && r > 0)
       cout << l << " " << r << endl;
  else
       cout << "NO" << endl;
}
```

Решение на 4 балла

Создадим массив на 5 элементов, в котором **a[0]** и **a[4]** образуют нужную пару. Заполним 4 из них, а последний пятый элемент **a[4]** будем обрабатывать в следующем цикле, который идет до **n-4**; .Запоминаем максимальный левый элемент кратный 13 и некратный 13 в переменные **maxlk** и **maxl** соответственно. После ввода последнего элемента **a[4]** проверяем, если он кратный 13, то проверяем его в паре с максимальным кратным 13 (**maxlk**) или максимальным некратным 13(**maxl**), иначе проверяем его только с кратным 13 (**maxlk**), после этого делаем сдвиг массива **a**. Выводим пару.

Решение на 4 балла, язык C++

```
#include <iostream>
using namespace std;
int main() {
  const int k = 13;
  int n; int a[5]; int maxl = 0; int maxlk = 0;
  int l = 0; int r = 0;
  cin >> n;
  for (int i = 0; i < 4; i++)
       cin >> a[i];
```

```
  for (int i = 0; i < n - 4; i++) {
       if (a[0] % k == 0 && a[0] > maxlk)
            maxlk = a[0];
       else
            if (a[0] > maxl)
                 maxl = a[0];
       cin >> a[4];
       if (a[4] % k == 0) {
            if (a[4] + maxlk > l + r) {
                 l = maxlk;
                 r = a[4];
            }
            if (a[4] + maxl > l + r) {
                 l = maxl;
                 r = a[4];
            }
       }
       else {
            if (a[4] + maxlk > l + r) {
                 l = maxlk;
                 r = a[4];
            }
       }
       a[0] = a[1];
       a[1] = a[2];
       a[2] = a[3];
       a[3] = a[4];
  }


  if (l > 0 && r > 0)
       cout << l << " " << r << endl;
  else
       cout << "NO" << endl;
}
```

Решение на 4 балла, язык Python (К.Ю. Поляков)

Очередь из 4-х элементов хранится в массиве queue. Для продвижения очереди используется цепочка

```
z = queue.pop(0)
queue.append(x)
```

Первый элемент очереди выходит в переменную **z** (и удаляется из очереди), а в конец очереди добавляется значение **x**.

Здесь **max13** – максимальное предыдущее число, кратное 13, вышедшее из очереди, а **maxAll** – максимальное из всех предыдущих чисел, которые вышли из очереди.

```
n = int(input())

queue = []          # заполняем очередь первыми 4-мя числами
for i in range(4):
  queue.append( int(input()) )

a, b = 0, 0              # эти переменные хранят пару чисел
maxAll, max13 = 0, 0
```

```
for i in range(n-4):     # в цикле читаем остальные n-4 числа
  x = int(input())
  z = queue.pop(0)       # продвижение очереди
  queue.append(x)
                         # обновление максимумов
  if z % 13 == 0 and z > max13:
    max13 = z
  if z > maxAll:
    maxAll = z
                         # проверка пар, в которых второе число x
  if x % 13 == 0:        # если x делится на 13, он стыкуется со всеми
    if x + maxAll > a + b:
      a, b = maxAll, x
  else:          # если x НЕ делится на 13, он стыкуется с теми,
    if x + max13 > a + b:    # которые делятся на 13
      a, b = max13, x

if a * b > 0:
  print(a, b)
else:
  print('NO')
```

Решение на 4 балла, язык PascalABC.NET (Н. Чурсин)

```
var
  i, x, n, max, maxk13, l, r, j: integer;
  a: array[1..4] of integer;

begin
  max := 0; maxk13 := 0;
  l := 0; r := 0;
  readln(n);
  for i := 1 to 4 do
    readln(a[i]);
  for i := 5 to n do begin
    readln(x);
    if (a[1] mod 13 = 0) and (a[1] > maxk13) then
       maxk13 := a[1]; // Поиск максимального кратного 13
    if a[1] > max then
       max := a[1]; // Поиск максимального из всех
    if x mod 13 = 0 then
      if (max > 0) and (max + x > l + r) then begin
        l := max; r := x;
      end
    else
      if (maxk13 > 0) and (maxk13 + x > l + r) then begin
        l := maxk13; r := x;
      end;
    for j := 1 to 3 do // Сдвиг буфера
      a[j] := a[j+1];
    a[4] := x;
  end;
  if l = 0 then
    writeln('NO')
  else
    print(l, r);
```

```
end.
```

Решение на 4 балла с массивом, язык Python (К.Ю. Поляков)

Здесь **maxPrev[0]** – максимальное предыдущее число, кратное 13, вышедшее из очереди, а **maxPrev[1]** – максимальное из всех предыдущих чисел, которые вышли из очереди.

```
n = int(input())
queue = []
for i in range(4):
  queue.append( int(input()) )

a, b = 0, 0
maxPrev = [0, 0]
for i in range(n-4):
  x = int(input())
  z = queue.pop(0)
  queue.append(x)
  zr13 = int(z % 13 > 0)
  maxPrev[zr13] = max(z, maxPrev[zr13])
  maxPrev[1] = max(z, maxPrev[1])
  xr13 = int(x % 13 > 0)
  if maxPrev[1-xr13]+x > a+b:
    a, b = maxPrev[1-xr13], x

if a * b > 0:
  print(a, b)
else:
  print('NO')
```

135) (**А.А. Богданов**) Для **решения на 2 балла** подходит стандартный шаблон перебора всех пар с проверкой нечетности разности элементов и сравнения расстояния между элементами с младшей цифрой правого элемента пары.

У этой задачи есть простое **решение на 3 балла**. Расстояние между элементами не может превышать 9, поэтому второй цикл необязательно заканчивать на N. И таким образом сложность алгоритма получится O(N), т.к. внутренний цикл не будет зависеть от N.

```
var a:array[1..10000] of integer;
    i,j,k,n:integer;
begin
  read(n);
  for i:=1 to n do read(a[i]);
  k:=0;
  for i:=1 to n-1 do
    for j:=i+1 to n do  // to min(i+9,n) do  - на 3 балла
      if (j-i=a[j] mod 10)and
         ((a[i]-a[j])mod 2<>0)then
           k+=1;
  print(k);
end.
```

Другое **решение на 3 балла** предполагает просмотр массива с конца, тогда внутренний цикл не требуется и сложность алгоритма будет O(N). Обратите внимание, что проверка на расстояние 0 не требуется, т.к. результат разности одного и того же числа будет чётным, а мы рассматриваем только пары с нечетной разность.

```
var a:array[1..10000] of integer;
    i,j,k,n:integer;
begin
```

```
  read(n);
  for i:=1 to n do read(a[i]);
  k:=0;
  for j:=n downto 2 do begin // выбор правого элемента
    i := j-a[j] mod 10; // позиция левого элемента пары
    if i>0 then // проверка выхода за пределы массива
      if (a[i]-a[j])mod 2<>0 then // проверка на нечет
        k+=1;
  end;
  print(k);
end.
```

Решение на 4 балла. Будем хранить в массиве 10 последних элементов. Если с каждым новым элементом сдвигать массив вправо, а очередной элемент писать в нулевую ячейку, тогда младшая цифра очередного (правого) числа будет индексом ячейки массива с левым элементом пары. Если этот элемент отличен от нуля (он уже загружен) и разность элементов пары нечетна, тогда увеличиваем счетчик найденных пар.

```
var b:array[0..9] of integer=(0,0,0,0,0,0,0,0,0,0);
    i,j,k,n,q:integer;
begin
  read(n);
  k:=0;
  for i:=1 to n do begin
    for j:=9 downto 1 do b[j]:=b[j-1];
    read(b[0]);
    q := b[0] mod 10;
    if (b[q]>0)and((b[0]-b[q])mod 2<>0) then k+=1;
  end;
  print(k);
end.
```

Буфер можно реализовать без сдвига элементов массива с помощью кольцевого буфера. Алгоритм будет эффективнее, но будет оцениваться также на 4 балла. В этом случае усложняется формула расчета парного индекса и алгоритм становится сложным для понимания.

```
var b:array[0..9] of integer=(0,0,0,0,0,0,0,0,0,0);
    i,j,k,n,q:integer;
begin
  read(n);
  k:=0;
  for i:=0 to n-1 do begin
    j := i mod 10;
    read(b[j]);
    q := (10-b[j] mod 10 + j)mod 10;
    if (b[q]>0)and((b[j]-b[q])mod 2<>0) then k+=1;
  end;
  print(k);
end.
```

(Д.Ф. Муфаззалов, Уфа) Проверки на четность и на наличие пары можно избежать, если хранить не сами числа, а признак наличия/отсутствия на соответствующем расстоянии от вновь введенного числа парного ему четного числа и нечетного числа. Это потребует двух массивов (двумерного массива) таких признаков. После сдвига буфера на освободившемся месте в строке, совпадающей по четности с введенным числом, нужно записать единицу 1 (наличие), а в другой строке записать ноль (отсутствие).

```
#include <iostream>
using namespace std;
```

```
int i,n,m,a[2][10],b,k,j,r;
int main()
{
    for (cin >> n; i < n; i++)
    {
        cin >> m, r=m%2;
        for (k=0; k<2; k++)
        {
            for (j = 9; j; j--) a[k][j] = a[k][j-1];
            a[k][0]=k==r;
        }
        b+=a[!r][m%10];
    }
    cout << b;
    return 0;
}
```

Глобальные переменные обнуляются автоматически, поэтому дополнительная их инициализация не требуется.

136) (**А.А. Богданов**) Решение **на 2 балла** с помощью стандартного шаблона перебора всех пар:

```
var a:array[1..10000] of integer;
    i,j,k,n,L,R:integer;
begin
  read(n);
  (L,R):=(0,0);
  for i:=1 to n do read(a[i]);
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if (j-i=a[i] mod 10)and((a[i]+a[j])mod 2<>0)and
         (a[i]+a[j]>L+R) then
           (L,R):=(a[i],a[j]);
  print(L,R);
end.
```

Решение на 3 балла. Можно заметить, что индекс второго элемента можно вычислить прямым смещением от индекса **i**, добавлением последней цифры **a[i]** элемента. И затем, перед обращением к элементу необходимо проверить, что полученный индекс не выходит за пределы массива. Случай расстояния 0 не нужно обрабатывать отдельно, т.к. удвоенное число будет четным, а мы рассматриваем только пары с нечетной суммой.

```
var a:array[1..10000] of integer;
    i,j,k,n,L,R:integer;
begin
  read(n);
  (L,R):=(0,0);
  for i:=1 to n do read(a[i]);
  for i:=1 to n-1 do begin
    j:=i+a[i] mod 10;
    if (j<=n)and((a[i]+a[j])mod 2<>0)and
       (a[i]+a[j]>L+R) then
         (L,R):=(a[i],a[j]);
    end;
  print(L,R);
end.
```

Решение на 4 балла. Максимальное расстояние между элементами не может быть больше 9, поэтому используем буфер для хранения последних 10 чисел. Первоначально заполним буфер. Для случая **N**<10 будем считывать только **N** элементов. Тогда инициализацию цикла запишем так:

```
   for i:=0 to min(n-1,9) do read(b[i]);
```

Основной цикл будет выполняться **N-1** раз для каждого вытесняемого из буфера элемента, кроме последнего, которому справа уже нет пары. В начале итерации рассмотрим пару между вытесняемым из буфера элементом (с индексом 0) и парным ему элементом, с индексом в буфере равным младшей цифре вытесняемого элемента. Если элемент в буфере больше нуля, тогда тестируем пару.

Часть элементов исходной последовательности уже считана в буфер. Поэтому, чтение следующего элемента будет по условию. Если последовательность еще не закончилась, в буфер будет помещен считанный элемент, иначе 0.

```
var b:array[0..9] of integer=(0,0,0,0,0,0,0,0,0,0);
    i,j,n,q,L,R:integer;
begin
  read(n);
  for i:=0 to min(n-1,9) do read(b[i]);
  (L,R):=(0,0);
  for i:=2 to n do begin
    q := b[0] mod 10;
    if (b[q]>0)and((b[0]+b[q])mod 2<>0)and(b[0]+b[q]>L+R)then
      (L,R):=(b[0],b[q]);
    for j:=0 to 8 do b[j]:=b[j+1]; // сдвиг буфера влево
    if i+10<=n then
      read(b[9])
    else
      b[9]:=0;
  end;
  print(L,R);
end.
```

(Муфаззалов Д.Ф.) От сдвига буфера можно отказаться, если смещать номер элемента, вытесняемого из буфера, во время ввода каждого значения. При этом буфер будем считать замкнутым в кольцо. Введенное число будем помещать на место числа, вытесненного из буфера.

Решение на 4 балла, язык C++, поиск пары числу, вытесненному из буфера, без сдвига буфера:

```
#include <iostream>
using namespace std;
int n,i,k,a[10],a1,a2,m;
int main()
{
    cin >> n;
    for (i=0; i<min(n,10); i++)
      cin >> a[i];

    for (i=0; i<n-1; i++)
    {
        m = i%10; // позиция элемента, вытесняемого из буфера
        k = (m+a[m]%10)%10;// позиция элемента, находящегося на
                           // расстоянии, равном младшей цифре
                           //вытесняемого значения
        if( a[k] && (a[m]+a[k]) % 2 &&
            a[m]+a[k] > a1+a2 ) {
          a1 = a[m];
```

```
          a2 = a[k];
          }
        if( i+10 < n)
          cin >> a[m];
        else a[m]=0;
    }
    cout << a1 << ' ' << a2;
    return 0;
}
```

Глобальные переменные обнуляются, поэтому дополнительная инициализация переменных не требуется.

Решение на 4 балла, язык Pascal ABC.NET, поиск пары числу, вытесненному из буфера, без сдвига буфера:

```
var
  b: array[0..9] of integer;
  i, n, q, L, R, m: integer;
begin
  read(n);
  for i := 0 to min(n - 1, 9) do read(b[i]);
  for i := 0 to n - 2 do
  begin
    m := i mod 10; // позиция элемента, вытесняемого из буфера
    q := (m + b[m] mod 10) mod 10;// позиция элемента, находящегося
                        // на расстоянии, равном младшей цифре
                         //вытесняемого значения
    if (b[q] > 0) and ((b[m] + b[q]) mod 2 <> 0)
                  and (b[m] + b[q] > L + R) then
      (L, R) := (b[m], b[q]);

    if i + 10 < n then
         read(b[m])
    else b[m] := 0;
  end;
  print(L, R);
end.
```

Глобальные переменные обнуляются, поэтому дополнительная инициализация переменных не требуется.

Можно использовать другую идею: подбирать каждому поступившему числу пару. Такая пара может находиться на расстоянии не более 9, поэтому в буфере нужно хранить 9 (для удобства 10, в нулевом элементе можно хранить вновь введенное число) последних введенных чисел. При поступлении нового числа нужно пройти по числам в буфере, и если последняя цифра числа равна расстоянию от него до введенного числа, то оно составляет пару с введенным числом. Затем буфер нужно сдвинуть вправо на 1 позицию. Проверку числа в буфере на ноль можно убрать, так как расстояние, равное нулю, не рассматривается. Глобальные переменные обнуляются, поэтому дополнительная инициализация переменных не требуется.

Решение на 4 балла, язык C++, поиск пары введенному числу, со сдвигом буфера:

```
#include <iostream>
using namespace std;
int n,num,i,j,a[10],a1,a2;
int main()
{
 cin>>n;
```

```
 for( i=0; i < n; i++ )
 {
  cin >> a[0];
  for( j=1; j<10; j++ ) // поиск пары
   if( a[j]%10 == j && a[j]+a[0] > a1+a2 &&
       (a[j]+a[0]) % 2 ) {
     a1 = a[j];
     a2 = a[0];
     }
   for( j=8; j>=0; j-- )
     a[j+1] = a[j]; // сдвиг буфера вправо
 }
 cout<<a1<< ' '<<a2;
 return 0;
}
```

Решение на 4 балла, язык Pascal, поиск пары введенному числу, со сдвигом буфера:

```
var
  a: array[0..9] of integer;
  n, num, i, j, a1, a2: integer;
begin
  readln(n);
  for i := 0 to n - 1 do
  begin
    readln(a[0]);
    for j := 1 to 9 do // поиск пары
      if (a[j] mod 10 = j) and (a[j] + a[0] > a1 + a2)
        and ((a[j] + a[0]) mod 2 <> 0) then (a1, a2) := (a[j], a[0]);
    for J := 8 downto 0 do a[j + 1] := a[j]; // сдвиг буфера вправо
  end;
  writeln(a1, ' ', a2);
end.
```

Глобальные переменные обнуляются, поэтому дополнительная инициализация переменных не требуется.

Как и в предыдущем решении, от сдвига буфера можно отказаться.

Решение на 4 балла, язык Pascal, поиск пары введенному числу, без сдвига буфера:

```
var
  a: array[0..9] of integer;
  n, i, j, k, a1, a2, m: integer;
begin
  readln(n);
  for i := 0 to n - 1 do//смещение позиции введенного числа вправо
  begin
    m := i mod 10;
    readln(a[m]);
    for j := 1 to 9 do // поиск пары
    begin
      k := (10 + m - j) mod 10; // номер кандидата в пары(1)
      if (a[k] mod 10 = j) and (a[k] + a[m] > a1 + a2)
                  and ((a[k] + a[m]) mod 2 <> 0) then
        (a1, a2) := (a[k], a[m]);
    end
```

```
  end;
  writeln(a1, ' ', a2);
end.
```

Глобальные переменные обнуляются, поэтому дополнительная инициализация переменных не требуется.

Формулу номера кандидата в пару (1) можно упростить, если смещать позицию введенного числа в буфере не вправо, а влево. Тогда ранее введенные будут находиться в буфере справа от введенного числа.

Решение на 4 балла, язык C++, поиск пары введенному числу, без сдвига буфера:

```
#include <iostream>
using namespace std;
int n,j,k,a[10],a1,a2,m;
int main()
{
 for( cin >> n; n--; )// смещение позиции введенного числа влево
 {
  m = n%10;
  cin >> a[m];
  for (j=1; j<10; j++)// поиск пары
  {
   k = (m+j)%10; // номер кандидата в пары
   if( a[k]%10 == j  && a[k]+a[m] > a1+a2  &&
       (a[k]+a[m]) % 2 ) {
     a1 = a[k];
     a2 = a[m];
     }
  }
 }
 cout << a1 << ' ' << a2;
 return 0;
}
```

Глобальные переменные обнуляются, поэтому дополнительная инициализация переменных не требуется.

137) Применим метод динамического программирования: получив очередное число **x** из входного потока, будем подбирать ему наилучшую пару из всех предыдущих значений последовательности. Если **x** делится на **C**, он может образовать пару с любым из предыдущих элементов, имеющих другой остаток от деления на **D**. Если **x** НЕ делится на **C**, он может образовать пару с только с теми из предыдущих элементов, которые имеют другой остаток от деления на **D** и делятся на **C**. Поэтому нам нужно хранить в двух массивах максимумы из всех чисел (распределённых по остаткам), и максимумы чисел, делящихся на C, также распределенных по остаткам. Введём два массива: **maxAll** и **maxC**:

```
   maxAll = [0]*D
   maxC = [0]*D
```

Элемент **maxAll[i]** – это максимум из всех предыдущих чисел, которые дают остаток **i** при делении на **D**. Элемент **maxC[i]** – это максимум из всех предыдущих чисел, которые делятся на **C** и дают остаток **i** при делении на **D**.

Пусть мы получили очередное число **x** из входного потока. определим его индексы делимости на **D** и **C**, а также остаток от деления на **D**:

```
     indD = 0 if x % D == 0 else 1
```

```
    indC = 0 if x % C == 0 else 1
    r = x % D
```

Будем искать пару. Если число делится на C, ищем в массиве maxAll максимум среди всех элементов, кроме элемента с индексом k, иначе ведём поиск в массиве maxC:

```
  pair = 0
  if indC == 0:
    for k in range(D):
      if r != k:
        pair = max(pair, maxAll[k])
  else:
    for k in range(D):
      if r != k:
        pair = max(pair, maxC[k])
```

Проверяем новую пару, если она оказывается лучше, чем предыдущая **(L,R)**, запоминаем новые значения элементов в переменных **L** и **R**:

```
  if pair > 0 and pair + x > L + R:
    L, R = pair, x
```

Обратите внимание на первую часть условия **pair>0**, она означает «нашли пару». Значение **pair=0** говорит о том, что пара не найдена.

Теперь нужно обновить массивы **maxAll** и **maxC**, точнее, их элементы, которые соответствуют остатку **r**:

```
  if indC == 0:
    maxC[r] = max(maxC[r], x)
  maxAll[r] = max(maxAll[r], x)
```

Полное решения на языке Python:

```
D = 140
C = 7
n = int(input())

maxAll = [0]*D
maxC = [0]*D
L, R = 0, 0

for i in range(n):
  x = int(input())
  indD = 0 if x % D == 0 else 1
  indC = 0 if x % C == 0 else 1
  r = x % D
  pair = 0
  if indC == 0:
    for k in range(D):
      if r != k:
        pair = max(pair, maxAll[k])
    maxC[r] = max(maxC[r], x)
  else:
    for k in range(D):
      if r != k:
        pair = max(pair, maxC[k])
  if pair > 0 and pair + x > L + R:
    L, R = pair, x
  maxAll[r] = max(maxAll[r], x)
print( L, R )
```

Используя технику, описанную в статье http://kpolyakov.spb.ru/download/ege27info.pdf, можно объединить массивы **maxAll** и **maxC** в один двухмерный массив из двух строк. Это немного сокращает программу:

```
D = 140
C = 7
n = int(input())

maxC = [[0]*D, [0]*D]
L, R = 0, 0

for i in range(n):
  x = int(input())
  indD = 0 if x % D == 0 else 1
  indC = 0 if x % C == 0 else 1
  r = x % D
  pair = 0
  for k in range(D):
    if r != k:
      pair = max(pair, maxC[indC][k])
  if pair > 0 and pair + x > L + R:
    L, R = pair, x
  maxC[0][r] = max(maxC[0][r], x)
  maxC[1][r] = max(maxC[1][r], x*(1-indC))

print( L, R )
```

Приведённое решение имеет один недостаток: внутри основного цикла есть ещё один цикл, выполняющий **D** итераций, так что сложность алгоритма оценивается как O(N*D), а не O(N). Можно применить другой подход: сначала в основном цикле посчитать максимальные значения а) из всех в массиве **maxAll**; б) делящиеся на **C** в массиве **maxC**:

```
  for i in range(n):
    x = int(input())
    r = x % D
    if x % C == 0:
      maxC[r] = max(maxC[r], x)
    maxAll[r] = max(maxAll[r], x)
```

а затем, уже после цикла, вычислять результат. Итак, у нас есть два массива с максимальными значениями, распределёнными по остаткам. Организуем цикл, в котором будем на каждом шаге **i** вычислять переменные **mAll** (максимальное значения из всех, у которых остаток не больше i) и **mC** (максимальное значения из всех, делящихся на **C**, у которых остаток не больше **i**):

```
  mC, mAll = 0, 0
  for i in range(1,D):
    ...
    mC = max(mC, maxC[i])
    mAll = max(mAll, maxAll[i])
```

Вместо многоточия нужно добавить операторы, которые выбирают максимум на данном этапе. Число **maxC[i]** может обрабовать пару со всеми, лучшая пара – с **mAll**. Если число maxAll[i] делится на **C**, проверяем его пару с **mAll**, если нет – с **mC**. Из всех пар выбираем лучшую:

```
  L, R = 0, 0
  mC, mAll = 0, 0
  for i in range(1,D):
    if maxC[i]+mAll > L+R:
      L, R = maxC[i], mAll
```

```
    if maxAll[i] % C == 0:
      if maxAll[i]+mAll > L+R:
        L, R = maxAll[i], mAll
    else:
      if maxAll[i]+mC > L+R:
        L, R = maxAll[i], mC
    mC = max(mC, maxC[i])
    mAll = max(mAll, maxAll[i])
```

Приведём полную программу:

```
D = 140
C = 7
n = int(input())

maxAll = [0]*D
maxC = [0]*D
for i in range(n):
  x = int(input())
  r = x % D
  if x % C == 0:
    maxC[r] = max(maxC[r], x)
  maxAll[r] = max(maxAll[r], x)

L, R = 0, 0
mC, mAll = 0, 0
for i in range(1,D):
  if maxC[i]+mAll > L+R:
     L, R = maxC[i], mAll
  if maxAll[i] % C == 0:
    if maxAll[i]+mAll > L+R:
      L, R = maxAll[i], mAll
  else:
    if maxAll[i]+mC > L+R:
      L, R = maxAll[i], mC
  mC = max(mC, maxC[i])
  mAll = max(mAll, maxAll[i])

print( L, R )
```

(**А. Богданов, К. Поляков**) Недостаток такого решения – зависимость количества операций и выделяемой памяти от **D**. Действительно, если увеличить **D** в k раз, то размер массивов **maxC** и **maxAll** тоже увеличится в k раз. Кроме того, количество итераций второго цикла (**for i in range(1,D)**) тоже увеличивается примерно в k раз.

Будем рассуждать следующим образом. В паре два числа, одно из которых делится на **C**, а второе – не обязательно. Если не учитывать условие с остатками, решение – это максимальное число, которое делится на **C**, и максимальное из оставшихся. Второе число также может делиться на **C**, поэтому нам нужно найти 1) два максимальных числа, делящихся на **C**, назовём их **maxC** и **maxC2**; 2) максимальное из чисел, которые НЕ делятся на **C**, назовём его **maxOther**.

Будем писать решение на языке PascalABC.NET. Сначала записываем в переменные **maxC**, **maxC2**, **maxOther** начальные значения (барьеры):

```
  const D = 120;
        C = 7;
        z = -20000;
  var maxC, maxC2, maxOther,
```

```
    i, x, n, L, R:integer;
...
(maxC, maxC2, maxOther) := (z, z, z);
```

В основном цикле читаем данные и ищём все нужные максимумы:

```
for i:= 1 to n do begin
  read(x);
  if x mod C = 0 then begin
    if x > maxC then begin
      maxC2 := maxC;
      maxC := x;
    end
    else if x > maxC2 then
      maxC2 := x;
  end
  else if x > maxOther then
      maxOther := x;
end;
```

Теперь остаётся проверить пары (maxC1, maxC2) и (maxC1, maxOther), выбрав из них наилучшую:

```
(L, R) := (0, 0);
if maxC+maxC2 > L+R then (L,R):=(maxC,maxC2);
if maxC+maxOther > L+R then (L,R):=(maxC,maxOther);
print(L,R);
```

Если подходящей пары нет (ни одно число не делится на **C**), то в переменной **maxC** останется барьерное значение -20000, и ни в одном из условных операторов условие не выполнится, так как при 1<=x<=10000 сумма не будет больше нуля. В этом случае программа выведет два нуля.

Но пока мы не учитывали условие разных остатков от деления на **D**. Может получиться так, что **maxC**, **maxC2** и **maxOther** имеют одинаковые остатки. Значит, при поиске двух максимумов, делящихся на **C**, нам нужно выбирать их так, чтобы они имели разные остатки. Кроме того, нам нужен также и второй максимум из чисел, которые не делятся на **C**, также с учетом условия разных остатков. Назовём эту переменную **maxOther2**.

Поскольку нам часто придётся проверять условие «остатки от деления двух чисел на **D** различны», оформим для этого логическую функцию:

```
function diffD(y, x:integer):= (y mod d) <> (x mod d);
```

Теперь при обновлении второго максимума проверяем условие различия остатков числа-кандидата и первого максимума:

```
if x mod C = 0 then begin
  if x > maxC then begin
    if diffD(maxC,x) then maxC2 := maxC;
    maxC := x;
  end
  else if (x > maxC2)and diffD(maxC,x) then
    maxC2 := x;
end
```

Аналогичные вызовы функции **diffD** нужно добавить и в алгоритм обновления **maxOther2**.

Когда основной цикл завершён, нужно проверить пары (**maxC**, **maxC2**) и (**maxC**, **maxOther**). В последней паре у обоих чисел могут оказаться одинаковые остатки. Поэтому дополнительно проверяем пары (**maxC**, **maxOther2**) и (**maxC2**, **maxOther**). Для проверки условия различия остатков вызывается функция **diffD**:

```
(L,R):=(0,0);
if maxC+maxC2 > L+R then
```

```
    (L,R):= (maxC,maxC2);
  if (maxC+maxOther > L+R) and diffD(maxC,maxOther) then
    (L,R):= (maxC,maxOther);
  if (maxC+maxOther2 > L+R) then
    (L,R):= (maxC,maxOther2);
  if (maxC2+maxOther > L+R) then
    (L,R):= (maxC2,maxOther);
  print(L,R);
```

Разберёмся, почему функция **diffD** вызывается только в одном условном операторе.

1) переменные **maxC** и **maxC2** имеют разные остатки по построению;

2) пары (**maxC,maxOther2**) и (**maxC2,maxOther**) могут «сыграть» только тогда, когда в паре (**maxC,maxOther**) оказались одинаковые остатки; в этом случае по построению имеем, что в каждой и пар (**maxC,maxOther2**) и (**maxC2,maxOther**) остатки разные.

Если какая-то из переменных не обновится, в ней останется барьерное значение -20000, и сумма гарантированно не сможет стать больше нуля (в переменных **L** и **R** останутся нули).

Приведём полную программу:

```
const D = 120; C = 7; z = -20000;
function diffD(y,x:integer):= y mod d <> x mod d;
var maxC, maxC2, maxOther, maxOther2,
    i,x,n,L,R:integer;
begin
  read(n);
  (maxC,maxC2,maxOther,maxOther2):=(z,z,z,z);
  for i:= 1 to n do begin
    read(x);
    if x mod C = 0 then begin
      if x > maxC then begin
        if diffD(maxC,x) then maxC2 := maxC;
        maxC := x;
      end
      else if (x > maxC2)and diffD(maxC,x) then
        maxC2 := x;
    end else if x > maxOther then begin
        if diffD(maxOther,x) then maxOther2 := maxOther;
        maxOther := x;
      end
      else if (x > maxOther2)and diffD(maxOther,x) then
        maxOther2 := x;
  end;
  (L,R):=(0,0);
  if maxC+maxC2 > L+R then
    (L,R):= (maxC,maxC2);
  if (maxC+maxOther > L+R)and diffD(maxC,maxOther) then
    (L,R):=(maxC,maxOther);
  if (maxC+maxOther2 > L+R) then
    (L,R):= (maxC,maxOther2);
  if (maxC2+maxOther > L+R) then
    (L,R):= (maxC2,maxOther);
  print(L,R);
end.
```

(**А. Куриленко, К. Поляков**) Четыре переменных в предыдущем решении удобно объединить в двухмерный массив ma, который инициализируется барьерными значениями. Будем писать программу на языке Python. Инициализируем переменные:

```
D, C = 140, 7
z = -20000
ma = [ [z]*2 for i in range(2) ]
```

Первый индекс «отвечает» за делимость на **C**, второй определяет номер максимума (0 – первый максимум, 1 – второй). Запишем соответствие между элементами этого массива и переменными в предыдущей программе:

```
ma[0][0] – maxC          ma[0][1] – maxC2
ma[1][0] – maxOther      ma[1][1] – maxOther2
```

Определим, как и ранее, функцию **diffD**:

```
def diffD(x, y):
  return (x % D != y % D)
```

Вводим данные и во время работы цикла определяем все нужные максимумы:

```
N = int(input())
for i in range(N):
  x = int(input())
  k = 0 if x % c == 0 else 1
  if x > ma[k][0]:
    if diffD(x, ma[k][0]):
        ma[k][1] = ma[k][0]
    ma[k][0] = x
  elif x > ma[k][1] and diffD(x, ma[k][0]):
    ma[k][1] = x
```

Для каждого введённого **x** определяем индекс делимости на **C**:

```
k = 0 if x % c == 0 else 1
```

Это значение выступает как номер строки в массиве.

После завершения цикла остается выбрать лучшую пару:

```
L, R = 0, 0
if ma[0][0]+ma[0][1] > L+R:
    L, R = ma[0][0], ma[0][1]
if ma[0][0]+ma[1][0] > L+R  and  diffD(ma[0][0], ma[1][0]):
    L, R = ma[0][0], ma[1][0]
if ma[0][0]+ma[1][1] > L+R  and  diffD(ma[0][0], ma[1][1]):
    L, R = ma[0][0], ma[1][1]
if ma[0][1]+ma[1][0] > L+R  and  diffD(ma[0][1], ma[1][0]):
    L, R = ma[0][1], ma[1][0]
```

Приведём полную программу:

```
def diffD(x, y):
  return (x % D != y % D)

D, C = 140, 7
ma = [ [-10000] * 2 for i in range(2) ]

N = int(input())
for i in range(N):
    x = int(input())
    k = 0 if x % C == 0 else 1
    if x > ma[k][0]:
        if diffD(x, ma[k][0]):
```

```
            ma[k][1] = ma[k][0]
        ma[k][0] = x
    elif x > ma[k][1] and diff(x, ma[k][0]):
        ma[k][1] = x

L, R = 0, 0
if ma[0][0]+ma[0][1] > L+R:
    L, R = ma[0][0], ma[0][1]
if ma[0][0]+ma[1][0] > L+R  and  diffD(ma[0][0], ma[1][0]):
    L, R = ma[0][0], ma[1][0]
if ma[0][0]+ma[1][1] > L+R:
    L, R = ma[0][0], ma[1][1]
if ma[0][1]+ma[1][0] > L+R:
    L, R = ma[0][1], ma[1][0]
print(L, R)
```

Так же, как и в предыдущей программе, в последних двух случаях нет необходимости проверять, что элементы пары имеют различные остатки от деления.

138) (**А. Богданов**)

Решение на 2 балла с использованием стандартного шаблона перебора всех возможных пар. Для разнообразия используем функциональных подход, доступный в последней версии PascalABC.Net:

```
const D=37;C=7;
begin
  var a := ReadArrInteger(ReadInteger); // читаем в массив
  var res := a.Combinations(2) // перебор сочетаний по 2
  .Where(x->(x.Sum mod D>0)and // сумма некратна D
            (x.Count(e->e mod C=0)=1)) // одно число кратно C
  .OrderBy(x->x.Sum) // сортируем пары по сумме
  .LastOrDefault;    // берем последнюю, с max суммой
  if res = nil then print(0,0) // если такой нет, выводим нули
  else res.Print; // или то что нашли
end.
```

Решение на 4 балла предполагает нахождение ответа на ходу. Для каждого очередного элемента последовательности X будем рассматривать две возможные пары с двумя предшествующими ему максимумами (m1,m2) с различными остатками от деления на D. Хотя бы с одним из двух найденных максимумов X образует пару с суммой некратной D. Если сумма превышает ранее найденную, элементы пары сохраняются в переменных (L,R). Для выполнения условия кратности C только одного из чисел пары, будем искать две пары максимумов. Одну пару кратных C (с индексом 0) и вторую некратных C (с индексом 1). Кратность X определяет рассматриваемую пару максимумов. Переменных для поиска максимумов инициализируем барьерным значением -10001, что позволит избежать дополнительных проверок наличия найденных максимумов.

```
const D = 37;
      C = 7;
      z = -10001;
var i, x, n, p, q, L, R:integer;
    m1, m2: array of integer;
begin
  read(n);
  (L,R):=(0,0);
  (m1, m2) := ( Arr(z,z), Arr(z,z) );
  loop n do begin
    read(x);
    p := sign(x mod C);
    q := 1-p;
```

```
    if (m1[q]+x > L+R) and ((m1[q]+x) mod d > 0) then
      (L, R):= (m1[q], x);
    if (m2[q]+x > L+R) and ((m2[q]+x) mod d > 0) then
      (L, R):= (m2[q], x);

    if x > m1[p] then begin
      if x mod d <> m1[p] mod d then
        m2[p] := m1[p];
      m1[p] := x;
    end else if (x > m2[p]) and (x mod d <> m1[p] mod d) then
      m2[p] := x;
  end;
  print(L,R);
end.
```

Решение на 4 балла на языке Python (**К. Поляков**). Здесь функция **diffD** проверяет, верно ли, что её аргументы дают различные остатки при делении на **D** (см. разбор задачи 137).

```
def diffD(x, y):
  return (x % D != y % D)

D, C = 37, 7
z = -10001

n = int(input())
L, R = 0, 0
m1, m2 = [z, z], [z, z]

for i in range(n):
  x = int(input())

  p = 0 if x % C == 0 else 1
  q = 1 - p

  if m1[q]+x > L+R and (m1[q]+x) % D > 0:
    L, R = m1[q], x
  if m2[q]+x > L+R and (m2[q]+x) % D > 0:
    L, R = m2[q], x

  if x > m1[p]:
    if diffD(x, m1[p]):
      m2[p] = m1[p]
    m1[p] = x
  elif x > m2[p] and diffD(x, m1[p]):
    m2[p] = x;

print(L, R)
```