Ответы на задачи С2:

1) Выделяем целочисленные переменные **i1** и **Sum**; в **i1** будем хранить номер первого в паре выбранных соседних элементов, а в **Sum** – их сумму. В **i1** записываем начальное значение 1, а в **Sum** – сумму первых двух элементов. В цикле рассматриваем все элементы массива **со второго до N-1**, если сумма текущего элемента и следующего за ним больше **Sum**, то запоминаем эту сумму в переменной **Sum**, а номер текущего элемента – в **i1**.

```
const N=30;
var A:array[1..N] of integer;
    i, i1, Sum: integer;
begin
  for i:=1 to N do readln(A[i]);
  i1:=1;
  Sum:=A[1]+A[2];
  for i:=2 to N-1 do
    if A[i]+A[i+1] > Sum then begin
      i1:=i;
      Sum:=A[i]+A[i+1]
    end;
  writeln(i1)
end.
```

2) Выделяем целочисленные переменные **k** и **max**; в **k** будем хранить количество элементов, равных максимальному, а в max – значение максимального элемента. В **k** записываем начальное значение 1, а в **max** – значение первого элемента. В цикле рассматриваем все элементы массива **со второго** до последнего. Если текущий элемент равен переменной **max**, то увеличиваем счетчик **k**. Если текущий элемент больше **max**, то нашли новый максимальный элемент, запоминаем его значение в переменной **max**, а в счетчик **k** записываем единицу. В конце цикла в **k** записано количество элементов, равных максимальному.

```
const N=30;
var a:array[1..N] of integer;
    max, k, i: integer;
begin
  for i:=1 to N do readln(A[i]);
  k:=1;
  max:=a[1];
  for i:=2 to N do begin
    if a[i]=max then k:=k+1;
    if a[i]>max then begin
      max:=a[i];
      k:=1
    end
  end;
  writeln(k)
end.
```

3) Введем целые переменные **Sum**, **Max** и **k**. В **Sum** будем хранить сумму трех последовательных элементов, начиная с текущего, а в **Max** – максимальную (на данный

момент) из этих сумм, а в **k** – номер первого элемента в цепочке с максимальной суммой. Сначала запишем в **Sum** и в **Max** сумму первых трех элементов, а в переменную **k** – единицу. В цикле рассматриваем все элементы массива **со второго** до **N-2**. Для получения очередной суммы вычитаем из **Sum** предыдущий элемент и добавляем элемент, который следует за текущим через один (можно также просто сложить текущий элемент и два следующих). Сравниваем **Sum** со значением переменной **Max**; если **Sum** больше, то заносим это значение в переменную **Max** и запоминаем номер текущего элемента в переменной **k**. По окончании работы алгоритма переменная **Max** содержит максимальную сумму трех подряд идущих элементов массива, а переменная **k** указывает на начало цепочки, которая состоит из элементов с номерами **k**, **k+1** и **k+2**.

```
const N=30;
var a:array[1..N] of integer;
    Max, Sum, i, k: integer;
begin
  for i:=1 to N do readln(A[i]);
  Sum:=a[1]+a[2]+a[3];
  Max:=Sum; k:=1;
  for i:=2 to N-2 do begin
    Sum:=Sum-a[i-1]+a[i+2]; { или Sum:=a[i]+a[i+1]+a[i+2];}
    if Sum >Max then begin
      Max:=Sum;
      k:=i
    end
  end;
  for i:=k to k+2 do
    writeln('A[',i,']=',a[i])
end.
```

4) Введем переменные **Min** (минимальная сумма элементов строки), **iMin** (номер строки с минимальной суммой) и **Sum** (сумма элементов текущей строки). Сначала в переменную **iMin** записываем 1, а в переменную **Min** – сумму элементов первой строки. Затем в цикле рассматриваем все строки, для каждой строки считаем сумму элементов и, если эта сумма меньше значения **Min**, записываем сумму в **Min**, а в **iMin** – номер текущей строки. В конце работы цикла в переменной **Min** будет находиться минимальная сумма элементов строки, а в **iMin** – номер этой строки.

```
const N=10; M=20;
var a:array[1..N,1..M] of integer;
    Min, Sum, iMin, i, k: integer;
begin
  { ввод матрицы N на N }
  Sum := 0;
  for k:=1 to M do Sum := Sum + a[1,k];
  Min := Sum;
  iMin := 1;
  for i:=2 to N do begin
    Sum := 0;
    for k:=1 to M do Sum := Sum + a[i,k];
    if Sum < Min then begin
      Min := Sum;
      iMin := i
    end
```

```
  end;
  writeln('Строка ', iMin, ' сумма ', Min )
end.
```

5) Введем переменные **Max** (значение максимального элемента), **iMax** (номер строки, в которой находится максимальный элемент), **Sum** (сумма элементов той строки, где находится максимальный элемент). Сначала определяем максимальный элемент в матрице: в переменную **Max** записываем значение **A[1,1]**, а в **iMax** – единицу (пока считаем, что максимальный элемент стоит в первой строке); затем в двойном цикле проходим все элементы матрицы, если очередной элемент больше **Max**, запоминаем его значение в **Max**, а номер строки – в **iMax**. После этого находим сумму элементов строки с номером **iMax**: в переменную **Sum** записываем 0, в цикле проходим все элементы этой строки, добавляя текущий элемент к **Sum**. Ответ находится в переменной **Sum**.

```
const N=10;
var A:array[1..N,1..N] of integer;
    Max, Sum, iMax, i, k: integer;
begin
  { ввод матрицы N на N }
  iMax := 1; Max := A[1,1];
  for i:=1 to N do
    for k:=1 to N do
      if A[i,k] > Max then begin
        Max := A[i,k];
        iMax := i
      end;
  Sum := 0;
  for k:=1 to N do
    Sum := Sum + A[iMax,k];
  writeln(Sum)
end.
```

6) Вводим целые переменные **Sum** (сумма минимальных элементов в каждой строке) и **Min** (минимальный элемент в текущей строке). Сначала в **Sum** записываем 0. В цикле рассматриваем все строки, для каждой определяем минимальный элемент и добавляем его к переменной **Sum**. Для поиска минимального элемента в строке записываем в **Min** значение первого элемента строки, а затем в цикле рассматриваем все остальные элементы строки, начиная со второго; если очередной элемент меньше **Min**, записываем его значение в **Min**. После выполнения программы результат находится в переменной **Sum**.

```
const N=6; M=10;
var A:array[1..N,1..M] of integer;
    Min, Sum, i, k: integer;
begin
  { ввод матрицы N на N }
  Sum := 0;
  for i:=1 to N do begin
    Min := A[i,1];
    for k:=2 to M do
      if A[i,k] < Min then Min := A[i,k];
    Sum := Sum + Min
  end;
```



```
  writeln(Sum)
end.
```

7) Нужно найти количество учеников, получивших оценку более 20 баллов (по условию – это не нуль), и общую сумму их оценок. Средняя оценка равна сумме оценок, деленной на их количество.
В переменной **x** будем считать учеников, сдавших экзамен, а в переменной **y** – накапливать сумму их оценок. Сначала в обе эти переменные запишем нули (начальные значения). Затем в цикле от 1 до 30 рассматриваем все элементы массива **A**. Если очередной элемент больше 20, увеличиваем переменную **x** на единицу (считаем этого ученика) и добавляем значение этого элемента массива к старому значению переменной **y**. После окончания цикла выводим результат деления **y** на **x**. Недостающая часть программы может выглядеть так:

```
x := 0; y := 0;  { не забыть начальные условия! }
for i:=1 to N do
  if A[i] > 20 then begin
    x := x + 1;
    y := y + A[i]
  end;
s := y / x;
writeln('Средний балл ', s:10:3);
```

8) Фактически нужно найти минимальный рост ученика среди всех учеников, имеющих рост 180 см или выше.
Значение минимального роста будем хранить в переменной **x**. Поскольку известно, что все ученики не выше 200 см, сначала в переменную **x** запишем 200 (или любое большее число). Затем в цикле от 1 до 30 рассматриваем все элементы массива **A**. Если очередной элемент больше или равен 180 (нашли игрока-баскетболиста) и одновременно меньше **x**, записываем значение этого элемента в переменную **x**. После окончания цикла выводим значение переменной **x**. Недостающая часть программы может выглядеть так:

```
x := 200;  { или любое число >= 200 }
for i:=1 to N do
  if (A[i] >= 180) and (A[i] < x) then
    x := A[i];
writeln('Рост ', x);
```

9) Нужно найти количество дней, когда была оттепель, и общую сумму температур в эти дни. Средняя температура вычисляется как сумме температур, деленная на количество дней. В переменной **x** будем считать дни оттепели, а в переменной **y** – накапливать сумму температур. Сначала в обе эти переменные запишем нули (начальные значения). Затем в цикле от 1 до 31 рассматриваем все элементы массива **A**. Если очередной элемент больше 0, увеличиваем переменную **x** на единицу (считаем этот день) и добавляем значение этого элемента массива к старому значению переменной **y**. После окончания цикла выводим результат деления **y** на **x**. Недостающая часть программы может выглядеть так:

```
x := 0; y := 0;  { не забыть начальные условия! }
for i:=1 to N do
  if A[i] > 0 then begin
    x := x + 1;
    y := y + A[i]
```

```
   end;
 s := y / x;
 writeln('Средняя температура ', s:10:3);
```

10) Фактически нужно найти в массиве минимальный элемент среди всех элементов, которые больше или равны 20. В отличие от классического алгоритма поиска минимального элемента в массиве, здесь требуется «отсечь» все элементы, которые меньше 20, с помощью дополнительного условия.

Записываем в переменную **min** начальное значение, равное 100. В цикле от первого элемента до тридцатого сравниваем элементы исходного массива с 20. Если текущий элемент больше или равен 20, то сравниваем значение текущего элемента массива со значением переменной **min**. Если текущий элемент массива меньше min, то записываем в **min** значение этого элемента массива. Переходим к следующему элементу. После завершения цикла выводим значение переменной **min**.

Недостающая часть программы может выглядеть так:

```
min := 100; { не забыть начальные условия! }
for i:=1 to N do
  if (a[i] >= 20) and (a[i] < min) then
    min := a[i];
writeln ( min );
```

11) В этой задаче нужно сначала найти среднее арифметическое всех элементов главной диагонали. Для этого требуется один простой (не вложенный!) цикл, потому что для элементов главной диагонали номер строки равен номеру столбца.

Пусть **N** (константа) – количество строк (и столбцов!) матрицы **A**. Введем вещественную переменную **sred**, в которой сначала подсчитаем сумму всех элементов главной диагонали. Введем целую переменную **i**, обозначающую номер строки. Запишем в **sred** начальное значение 0. В цикле изменяем **i** от 1 до **N** с шагом 1, добавляем к значению переменной **sred** значение элемента матрицы **A[i,i]**. После окончания цикла разделим **sred** на **N** (на главной диагонали всего **N** элементов), таким образом, в **sred** находится среднее значение элементов главной диагонали.

Теперь можно считать (только!) положительные элементы всей матрицы, которые больше **sred**. Вводим целочисленные переменные **j** (номер столбца) и **count** (счетчик «нужных» элементов) . В счетчик записываем начальное значение 0.

Организуем двойной цикл, перебирающий все комбинации **(i,j)** для **i=1..N** и **j=1..N**. В теле цикла проверяем элемент матрицы **A[i,j]**: если он больше нуля и больше **sred**, увеличиваем счетчик **count** на 1.

После окончания двойного цикла выводим значение **count**.

Программа может выглядеть так:

```
const N=5;
var A:array[1..N,1..N] of integer;
    i, j, count: integer;
    sred: real;
begin
  for i:=1 to N do    { ввод матрицы }
    for j:=1 to N do readln(A[i,j]);
  sred:=0;            { находим сумму главной диагонали }
  for i:=1 to N do
    sred := sred + A[i,i];
  sred := sred / N;   { находим среднее }
  count := 0;         { считаем нужные элементы }
  for i:=1 to N do
```



```
    for j:=1 to N do
      if (A[i,j] > 0) and (A[i,j] > sred) then
        count := count + 1;
  writeln(count)
end.
```

Заметим, что можно немного улучшить программу. В условном операторе в последнем двойном цикле можно заменить сложное условие простым, если вместо 0 и **sred** использовать максимальное из этих значений. Перед двойным циклом нужно добавить оператор

```
if sred < 0 then sred := 0;
```

а условный оператор изменить так:

```
if A[i,j] > sred then
  count := count + 1;
```

Во-вторых, можно немного более грамотно обработать условие **A[i,j] > sred**. Дело в том, что при делении в операторе

```
sred := sred / N;   { находим среднее }
```

может получиться вещественное число (с дробной частью). Вещественные числа (за редким исключением[1]) хранятся в памяти компьютера неточно, потому что в двоичном коде содержат (теоретически) бесконечное число разрядов. Поэтому лучше НЕ делить полученную сумму **sred** на **N**, а для проверки вместо условия **A[i,j] > sred/N** использовать равносильное ему **A[i,j]*N > sred**. Плюс в том, что в последнем случае все операции выполняются с целыми числами и ошибок из-за неточного представления дробных чисел в памяти гарантированно не будет. Однако, есть и минус: вместо одного деления (на всю программу) придется выполнять N^2 умножений (для каждого элемента матрицы). Вот еще одна версия программы:

```
const N=5;
var A:array[1..N,1..N] of integer;
    i, j, count: integer;
    sred: real;
begin
  for i:=1 to N do    { ввод матрицы }
    for j:=1 to N do readln(A[i,j]);
  sred:=0;            { находим сумму главной диагонали }
  for i:=1 to N do
    sred := sred + A[i,i];
  count := 0;         { считаем нужные элементы }
  if sred < 0 then sred := 0;
  for i:=1 to N do
    for j:=1 to N do
      if A[i,j]*N > sred then
        count := count + 1;
  writeln(count)
end.
```

12) Эта задача в целом напоминает упрощенный вариант предыдущей, поэтому можно сразу привести решение. Единственная тонкость – можно исключить операции с вещественными числами, которые (теоретически) могут привести к ошибкам. Подробно этот прием описан в решении предыдущей задачи.

Решение на естественном языке. Записываем в переменную **s** начальное значение 0. В цикле добавляем все значения элементов массива, от 1-ого до 30-ого, к значению

[1] Не равные сумме отрицательных степеней числа 2.

переменной **s**. После завершения цикла делим значение **s** на 30, чтобы найти среднее значение. Далее записываем в переменную **j** начальное значение 0. В цикле рассматриваем все элементы массива, от
1-ого до 30-ого, сравниваем значение текущего элемента со значением переменной **s**. Если значение текущего элемента больше значения **s**, увеличиваем счетчик **j** на 1. После завершения цикла выводим значение переменной **j**.

В приведенном выше решении на естественном языке дан «чистый» алгоритм. При работе на реальном компьютере нужно заботиться о точности вычислений, поэтому в программе на Паскале вместо проверки условия **a[i] > среднего** мы используем равносильное ему **a[i]*N > суммы элементов**. При этом переменную s можно было бы сделать и целой (но она вещественная по условию задачи, это нельзя менять!).

```
const N=30;
var a: array [1..N] of integer;
    i, j: integer;
    s: real;
begin
  for i:=1 to N do readln(a[i]);
  s:=0;
  for i:=1 to N do s:=s+a[i];
  j:=0;
  for i:=1 to N do
   if a[i]*N>s then j:=j+1;
  writeln(j)
end.
```

13) Хитрость этой задачи в том, что нужно найти *первый* элемент, равный **X**. Вроде бы все просто – в цикле перебираем все элементы массива, если очередной элемент равен **X**, записываем его номер в переменную **j**.

```
for i:=1 to N do
  if a[i] = x then
    j := i; { запомнили номер }
```

Однако при этом в конце цикла в переменной **j** оказывается номер *последнего* элемента, равного **X**. Чтобы выйти их этой ситуации, можно остановить цикл, как только мы нашли первый элемент, равный **X**. В Паскале (и в Си) это делается с помощью оператора **break** (досрочный выход из цикла):

```
for i:=1 to N do
  if a[i] = x then begin
    j := i; { запомнили номер }
    break;
  end;
```

Есть и другой способ – просматривать элементы с конца, при этом не нужен досрочный выход из цикла:

```
for i:=N downto 1 do
  if a[i] = x then
    j := i; { запомнили номер }
```

Как определить, что элемента, равного **X**, нет в массиве? Для этого до цикла в переменную **j** нужно записать любое значение, которое не может быть номером элемента массива, например, 0. Если по окончании цикла в переменной **j** остался 0, ни одного элемента, равного **X**, не нашли.

Решение на естественном языке. Записываем в переменную **j** начальное значение 0. В цикле рассматриваем последовательно все элементы массива в обратном порядке, с 30-го до 1-го. Если очередной элемент равен **X**, записываем его номер в переменную **j**. Если после завершения цикла переменная **j** равна 0, выводим сообщение, что таких элементов нет, иначе выводим значение переменной **j**.

Решение на Паскале.

```
const N=30;
var a: array [1..N] of integer;
    i, j, x: integer;
begin
  for i:=1 to N do readln(a[i]);
  readln(x);
  j:=0;
  for i:=N downto 1 do
    if a[i] = x then
      j := i; { запомнили номер }
  if j = 0 then
       writeln('Нет таких элементов')
  else writeln(j)
end.
```

Можно также использовать цикл **while** со сложным условием:

```
i:=1;
while (i<=N) and (a[i]<> x) do
  i:=i+1;
```

Цикл остановится, когда **i** станет больше **N** или найдём элемент, равный **X**. Если после цикла переменная **i** больше **N**, значит, ни одного элемента, равного **X**, в массиве нет. Иначе в переменной **i** находится номер первого элемента, равного **X**.

```
  if i > N then
       writeln('Нет таких элементов')
  else writeln(i);
```

Это второй вариант решения задачи.

14) Сложность в том, что нужно найти не максимальный элемент, а второй по величине. Можно, конечно, сначала найти максимум, а потом искать следующий за ним, но можно сделать это за один проход по массиву. Нам нужны две переменные, **max** (максимальный элемент) и **max2** (второй максимум). Сначала выбираем максимальный из первых двух элементов и записываем его значение в **max**, а второй по величине записываем в **max2**:

```
if a[1] > a[2] then begin
  max:=a[1]; max2:=a[2];
end
else begin
  max:=a[2]; max2:=a[1];
end;
```

Затем в цикле перебираем все элементы, начиная с 3-го (первые два уже «задействованы»!) до последнего, 30-ого. Если очередной элемент **a[i]** больше, чем **max**, записываем значение **max** в **max2** (предыдущий максимум становится вторым), а значение **a[i]** – в **max**. Иначе, если **a[i]** больше, чем **max2**, записываем значение **a[i]** в **max2**. После завершения цикла выводим значение переменной **max2**. Вот решение на Паскале:

```
const N=30;
var a: array [1..N] of integer;
```

```
    i, k, max, max2: integer;
begin
  for i:=1 to N do readln(a[i]);
  if a[1] > a[2] then begin
    max:=a[1]; max2:=a[2]
  end
  else begin
    max:=a[2]; max2:=a[1]
  end;
  for i:=3 to N do
    if a[i] > max then begin
      max2 := max;
      max := a[i]
    end
    else if a[i] > max2 then max2 := a[i];
  writeln(max2)
end.
```

15) Очевидно, что нужно считать найденные положительные элементы (например, с помощью переменной **k**) и, когда **k** станет равно 3, запомнить номер текущего элемента (в переменной **j**).

Решение на естественном языке. Записываем в переменную **k** начальное значение 0. Затем в цикле перебираем все элементы массива с 1-ого по 30-ый. Если очередной элемент больше нуля, увеличиваем счетчик **k**. Если **k=3**, записываем номер текущего элемента в переменную **j**. Если после окончания цикла **k<3**, выводим сообщение, что в массиве меньше трех положительных элементов, иначе выводим значение переменной **j**.

Решение на Паскале.

```
const N=30;
var a: array [1..N] of integer;
    i, j, k: integer;
begin
  for i:=1 to N do readln(a[i]);
  k:=0;
  for i:=1 to N do
    if a[i] > 0 then begin
      k:=k+1;
      if k = 3 then j := i
    end;
  if k < 3 then
       writeln('Меньше трех положительных элементов')
  else writeln(j)
end.
```

16) Обратим внимание, что нужно найти не длину, а *сумму* наибольшей (то есть, самой длинной!) возрастающей последовательности (то есть, такой, в которой каждый следующий элемент строго больше предыдущего). В переменных **l** и **s** будем хранить длину и сумму *текущей* (рассматриваемой сейчас) последовательности, а в переменных **lmax** и **smax** – значения для наибольшей последовательности.

Решение на естественном языке. Записываем в переменную **lmax** начальное значение 0, в переменную **l** – значение 1, а в переменную **smax** – значение первого элемента массива. В цикле рассматриваем все элементы массива, начиная со 2-ого до

30-ого. Если очередной элемент больше предыдущего, увеличиваем переменную **l** на 1, а к переменной **s** добавляем значение этого элемента; иначе записываем 1 в переменную **l** и значение этого элемента в **s**. После этого (в теле цикла) сравниваем **l** и **lmax**; если **l > lmax** (нашли новую самую длинную возрастающую цепочку), записываем значение **s** в **smax**.

Решение на Паскале.

```
const N=30;
var a: array [1..N] of integer;
    i, l, lmax, s, smax: integer;
begin
  for i:=1 to N do readln(a[i]);
  lmax:=0; l:=1; s:=a[1];
  for i:=2 to N do begin
    if a[i] > a[i-1] then begin
      l:=l+1; s:=s+a[i]
    end
    else begin
      l:=1; s:=a[i]
    end;
    if l > lmax then begin
      lmax:=l;
      smax:=s
    end
  end;
  writeln(smax)
end.
```

17) Сначала нужно найти (в переменной **s**) среднее арифметическое всех элементов массива – считаем в цикле их сумму и делим на **N**. Затем перебираем во втором цикле все элементы массива и ищем тот, для которого модуль разности этого элемента и среднего арифметического наименьший.

Решение на естественном языке. Записываем в переменную **s** начальное значение 0. В цикле для всех элементов с 1-го до 30-го добавляем значение текущего элемента к переменной **s**. После окончания цикла делим значение переменной **s** на **N**, таким образом, получаем в переменной **s** среднее арифметическое всех элементов массива. Записываем в переменную **k** начальное значение 1. В цикле рассматриваем все элементы со 2-ого до 30-ого, если модуль разности текущего элемента и переменной **s** меньше, чем модуль аналогичной разности для **k**-ого элемента, записываем в переменную **k** номер текущего элемента. После окончания цикла выводим значение переменной **k**.

Решение на Паскале.

```
const N=30;
var a: array [1..N] of integer;
    i, k: integer;
    s, min: real;
begin
  for i:=1 to N do readln(a[i]);
  s:=0;
  for i:=1 to N do s:=s+a[i];
  s:=s/N;
  k:=1;
  for i:=2 to N do
```

```
    if abs(a[i]-s) < abs(a[k]-s) then
      k:=i;
  writeln(k)
end.
```

18) Очевидно, что нужно вывести *номера* минимального элемента массива и «второго» минимума. Поэтому эта задача решается аналогично задаче 14 (меняем названия переменных и знаки **>** на знаки **<**, храним в переменных **min** и **min2** не значения, а номера элементов):

```
const N=30;
var a: array [1..N] of integer;
    i, k, min, min2: integer;
begin
  for i:=1 to N do readln(a[i]);
  if a[1] < a[2] then begin
    min := 1; min2:= 2
  end
  else begin
    min:= 2; min2:= 1
  end;
  for i:=3 to N do
    if a[i] < a[min] then begin
      min2 := min;
      min := i
    end
    else if a[i] < a[min2] then min2 := i;
  writeln(min, ' ', min2)
end.
```

19) В этой задаче нужно перебрать все пары различных элементов. Для этого нужен двойной (вложенный) цикл. Чтобы не сравнивать элемент сам с собой, будем рассматривать только пары **(a[i],a[j])**, где **j>i**. Поэтому во внешнем цикле переменная **i** меняется от 1 до **N-1** (до предпоследнего), а во внутреннем – переменная **j** меняется от **i+1** до **N** (до конца массива):

```
for i:=1 to N-1 do
  for j:=i+1 to N do
    ...
```

Решение на естественном языке. Сначала считаем, что нужная пара – это первые два элемента. Записываем в переменные **min** и **min2** значения 1 и 2 соответственно, а в переменную **s** – модуль их разности. Организуем вложенный цикл. Во внешнем цикле перебираем значения переменной **i** от 1 до **N-1**. Во внутреннем цикле перебираем значения переменной **j** от **i+1** до **N**. В теле цикла: если модуль разности **a[i]-a[j]** оказывается меньше значения переменной s, записываем в переменную s этот модуль разности, а в переменные **min** и **min2** значения переменных **i** и **j** соответственно. После окончания двойного цикла выводим значения переменных **min** и **min2**.

Решение на Паскале.

```
const N=30;
var a: array [1..N] of integer;
    i, j, min, min2, s: integer;
begin
  for i:=1 to N do readln(a[i]);
  min:=1; min2:=2;
  s:=abs(a[1]-a[2]);
```



```
  for i:=1 to N-1 do
    for j:=i+1 to N do
      if abs(a[i]-a[j]) < s then begin
        s:=abs(a[i]-a[j]);
        min:=i; min2:=j
      end;
  writeln(min);
  writeln(min2)
end.
```

20) Среднее арифметическое – это сумма всех нужных элементов, деленная на их количество. Если значение *в десятичной системе* оканчивается на 5, это значит, что остаток от его деления на 10 (основание системы счисления) равен 5. Кроме того, есть еще один тонкий момент: при вычислении на компьютере остатка от деления *отрицательных* чисел (например, с помощью оператора **mod** в Паскале) этот остаток получается ***отрицательным***[2], например **-25 mod 10 = -5**.

Нам нужно использовать две переменных: счетчик найденных элементов и сумму; обе переменные сначала необходимо обнулить.

Затем надо пройти в цикле весь массив, и если очередной элемент при делении на 10 дает остаток 5 или –5, увеличить счетчик на 1, а сумму – на значение этого элемента.

Затем считаем среднее как отношение суммы к количеству. Поскольку сказано, что хотя бы один такой элемент есть, можно не опасаться деления на ноль.

Решение на естественном языке. Записываем в переменные **x** и **y** нулевые значения. В цикле перебираем значения переменной **i** от 1 до **N**. Если очередной элемент при делении на 10 дает в остатке 5 или –5, увеличиваем счетчик **x** на 1 и сумму **y** – на значение этого элемента. После окончания цикла записываем в переменную **s** результат деления **y** на **x**. Выводим значение переменной **s**.

Решение на Паскале.

При проверке остатка можно использовать сложное условие:

```
if (a[i] mod 10 = 5) or (a[i] mod 10 = -5) then
  begin ... end;
```

или перед применением операции mod взять модуль очередного элемента массива:

```
if abs(a[i]) mod 10 = 5 then begin ... end;
```

Еще один вариант условия предложил читатель **Ion**:

```
if (a[ i] mod 5 = 0) and (a[ i] mod 2 <> 0) then begin
...
end;
```

что означает «число делится на 5 и не делится на 2», то есть оканчивается на 5.

Один из вариантов решения:

```
const N=30;
var a: array [1..N] of integer;
    i, x, y: integer;
    s: real;
begin
  for i:=1 to N do readln(a[i]);
  x:=0; y:=0;
  for i:=1 to N do
    if abs(a[i]) mod 10 = 5 then begin
      x := x + 1;
```

[2] Заметим, что в математике остаток – это всегда целое *неотрицательное* число.

```
      y := y + a[i]
    end;
  s := y / x;
  writeln(s)
end.
```

Варианты задачи. Если требуется определить среднее арифметическое четных или нечетных элементов, нужно проверять делимость на 2, а не на 10. Если остаток от деления на 2 равен 0, то число четное.

21) Это вариант предыдущей задачи, меняется только условие отбора. Для нечетных элементов остаток от деления на 2 равен 1, поэтому условие выглядит так:

```
    if a[i] mod 2 = 1 then begin ...
```

Можно даже написать такое условие:

```
    if a[i] mod 2 <> 0 then begin ...
```

оно позволит решать задачу, в которой входные данные могут быть как положительными, так и отрицательными (для отрицательных нечетных чисел при вычислении остатка от деления на 2 получается (–1)).

Решение на естественном языке. Записываем в переменные **x** и **y** нулевые значения. В цикле перебираем значения переменной **i** от 1 до **N**. Если очередной элемент при делении на 2 дает в остатке 1, увеличиваем счетчик **x** на 1 и сумму **y** – на значение этого элемента. После окончания цикла записываем в переменную **s** результат деления **y** на **x**. Выводим значение переменной **s**.

Решение на Паскале.

```
const N=30;
var a: array [1..N] of integer;
    i, x, y: integer;
    s: real;
begin
  for i:=1 to N do readln(a[i]);
  x:=0; y:=0;
  for i:=1 to N do
    if a[i] mod 2 <> 0 then begin
      x := x + 1;
      y := y + a[i]
    end;
  s := y / x;
  writeln(s)
end.
```

22) Это вариант задачи, разобранной на с. 3 а файле C3.doc. Однако нужно учитывать, что счетчики в самом начале и при обнаружении неотрицательного элемента нужно устанавливать в 0.

Решение на естественном языке. Записываем в переменные **k** и **kMax** нулевые значения. В цикле перебираем значения переменной **i** от 1 до **N**. Если очередной элемент массива **A[i]** отрицательный, увеличиваем счетчик **k**, который обозначает длину текущей последовательности отрицательных элементов. Если очередной элемент массива **A[i]** неотрицательный, записываем в счетчик **k** ноль. Если k>kMax, записываем в **kMax** значение **k**. После окончания цикла выводим значение переменной **kMax**.

Решение на Паскале.

```
const N =30;
var A: array[1..N] of integer;
```



```
  i, k, kMax: integer;
begin
for i:=1 to N do readln(A[i]); { ввод массива }
k := 0;
kMax := 0;
for i:=1 to N do begin     { проходим весь массив }
  if A[i] < 0 then         { цепочка продолжается }
       k := k + 1
  else k := 0;             { цепочка закончилась }
  if k > kMax then kMax := k
end;
writeln(kMax);
end.
```

23) Фактически нужно найти максимум среди всех отрицательных элементов. Здесь есть тонкость – что записать в переменную **max**? Очевидно, что нельзя записать первый элемент массива (как это делается в классическом алгоритме поиска максимума), потому что он может не быть отрицательным. Выход – записать большое по модулю отрицательное число, которого заведомо нет в массиве, например, -100. При этом первый же отрицательный элемент окажется больше этого числа и начальное значение **max** будет заменено. Если значение -100 в переменной **max** останется после просмотра массива, то это будет означать, что отрицательных элементов нет.

Заметим, что если гарантируется, что в массиве есть хотя бы один отрицательный элемент, переменной **max** можно задать начальное значение -20 (наименьшее допустимое).

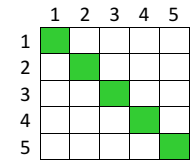
Решение на естественном языке. Записываем в переменную **max** начальное значение -100. В цикле перебираем значения переменной **i** от 1 до **N**. Если очередной элемент массива **A[i]** отрицательный и больше **max**, записываем в **max** значение этого элемента. После окончания цикла выводим значение переменной **max**.

Решение на Паскале.

```
const N=30;
var A: array [1..N] of integer;
    i, max: integer;
begin
  for i:=1 to N do readln(A[i]);
  max:=-100; { любое число <= -20 }
  for i:=1 to N do
    if (A[i] < 0) and (A[i] > max) then
      max:=A[i];
  writeln(max)
end.
```

24) Среднее арифметическое – это сумма элементов, деленная на их количество. Главная диагональ квадратной матрицы A размера N на N – это элементы, у которых индексы строки и столбца совпадают:

A[1,1], A[2,2], A[3,3], ..., A[N,N]



Главная ловушка таких задач: кажется, что если используется матрица, то в решении будет двойной цикл. Например, некоторые считают сумму элементов главной диагонали так:

```
S:=0;
for i:=1 to N do
  for j:=1 to N do
```

```
    S:=S+A[i,i];
```

В самом деле, в этом цикле каждый элемент главной диагонали добавляется в сумму N раз. Для того, чтобы понять ошибку, можно посмотреть на то, как расположены нужные нам элементы в матрице – они «вытянуты» в одну линию. В то же время, вложенный цикл предназначен для обработки областей (например, прямоугольных или треугольных). Второй цикл здесь не нужен, вот правильное решение:

```
S:=0;
for i:=1 to N do
  S:=S+A[i,i];
```

Далее все просто: делим сумму на N (количество элементов главной диагонали) и получаем среднее арифметическое.

Решение на естественном языке. Записываем в переменную **S** начальное значение 0. В цикле перебираем значения переменной **i** от 1 до **N**. На каждом шаге цикла добавляем к сумме S значение очередного элемента главной диагонали матрицы **A[i,i]**. После окончания цикла выводим значение **S/N**.

Решение на Паскале.

```
const N=5;
var A: array [1..N,1..N] of integer;
    i, S: integer;
begin
  { ввод матрицы A }
  S:=0;
  for i:=1 to N do
    S:=S+A[i,i];
  writeln(S/N)
end.
```

25) Эта задача похожа на задачу 23. Здесь нужно найти минимальное из трехзначных чисел, то есть из таких, которые удовлетворяют условию

```
(100 <= A[i]) and (A[i] < 1000)
```

В переменную **min** вначале мы должны записать начальное значение, которое заведомо больше любого трехзначного числа в массиве, например, 1000. Если это значение останется в переменной **min** после проверки всего массива, это значит, что в массиве нет трехзначных чисел.

Решение на естественном языке. Записываем в переменную **min** начальное значение 1000. В цикле перебираем значения переменной **i** от 1 до **N**. Если очередной элемент массива – трехзначное число (больше или равно 100 и меньше 1000) и меньше, чем значение переменной **min**, записываем значение этого элемента в **min**. После окончания цикла проверяем значение **min**: если оно равно 1000, выводим на экран сообщение «Нет таких элементов», иначе выводим значение **min**.

Решение на Паскале.

```
const N=30;
var A: array [1..N] of integer;
    i, min: integer;
begin
  for i:=1 to N do readln(A[i]);
  min:=1000;
  for i:=1 to N do
    if (100 <= A[i]) and (A[i] < 1000) and
       (A[i] < min) then min := A[i];
  if min = 1000 then
```



```
      writeln('Нет таких элементов')
  else writeln(min)
end.
```

26) Очевидно, что главная проблема в этой задаче – поиск количества делителей числа. Пусть нужно найти число делителей для **A[i]** – элемента массива **A** с индексом **i**. «Лобовое» решение выглядит так (просто проверяем делимость **A[i]** на все числа от 1 до **A[i]**, если делится – остаток от деления равен 0 – увеличиваем счетчик делителей **k**) :

```
k:=0;
for j:=1 to A[i] do
  if A[i] mod j = 0 then k:= k + 1;
```

Это не лучшее решение (по скорости, см. ниже), но в задаче С2 не требуется оптимальность, поэтому можно остановиться на этом варианте. Остальная часть сводится к задаче поиска в массиве максимального элемента и его номера, только вместо значений элементов массива нужно использовать при сравнении количество делителей. Обозначим через **imax** и **kmax** соответственно номер и значение элемента, имеющего максимальное количество делителей. Сначала в **imax** запишем 0 – это будет означать, что ни одного числа еще не рассмотрено. Затем в цикле перебираем все элементы массива, считаем для каждого количество делителей в переменной **k**. Если количество делителей очередного элемента массива больше, чем для всех предыдущих элементов (**k>kmax**), запоминаем номер и значение этого элемента:

```
imax:=0; { не обязательно! }
kmax:=0;
for i:=1 to N do begin
  k:=0;
  for j:=1 to A[i] do
    if A[i] mod j = 0 then k:= k + 1;
  if k > kmax then begin
    imax := i; kmax := k
  end
end;
```

Вообще говоря, строчка **imax:=0** в программе не обязательна – это значение все равно будет затерто на первом шаге цикла. Заметим, что при рассмотрении первого элемента массива (при **i=1**) условие **k > kmax** выполняется гарантированно, потому что начальное значение **kmax** равно 0, а количество делителей **kmax** будет больше нуля.

Выводить нужно элемент с номером **imax**.

Решение на естественном языке. Записываем в переменную **kmax** значение 0. Затем в цикле перебираем значения переменной **i** от 1 до **N**. Считаем количество делителей очередного элемента массива **A[i]** следующим образом:

1. записываем в переменную **k** значение 0;
2. в цикле изменяем переменную **j** от 1 до **A[i]**; если остаток от деления **A[i]** на **j** равен 0, то увеличиваем значение **k** на 1.

Если полученное число делителей больше **kmax**, записываем значение **i** в **imax** и значение **k** в **kmax**.

Выводим значение **imax** – номер элемента, имеющего наибольшее число делителей.

Решение на Паскале.

```
const N=30;
var a: array[1..N] of integer;
    i,j,k,imax,kmax: integer;
begin
kmax:=0;
```

```
for i:=1 to N do readln(a[i]);
for i:=1 to N do begin
  k:=0;
  for j:=1 to a[i] do
    if a[i] mod j = 0 then k:= k + 1;
  if k > kmax then begin
    imax := i; kmax := k
  end
end;
writeln(imax)
end.
```

Теперь подумаем про оптимизацию. Делители числа составляют пары, поэтому их можно и считать парами. При этом перебор можно выполнять от 1 до квадратного корня из числа. Исключение – единица (имеющая единственный делитель) и числа, составляющие полный квадрат, например, делитель 6 числа 36 не имеет пары. Попробуйте разобраться в такой реализации:

```
  if a[i] = 1 then k:=1
  else k:=2;
  j:=2;
  while j*j < a[i] do begin
    if a[i] mod j = 0 then k := k + 2;
    j:=j+1
  end;
  if j*j = a[i] then k:=k+1;
```

Она длиннее, чем «лобовой» вариант, но работает несколько быстрее. Конечно, профессионал написал бы функцию для вычисления количества делителей числа. Однако здесь возникают сложности из-за того, что нельзя вводить новые локальные переменные (можно обойтись и тем, что есть, но нужно это делать очень аккуратно).
Еще раз заметим, что в задаче С2 не требуется оптимальность решения, поэтому нужно выбирать самый простой вариант, в котором меньше вероятности сделать ошибку.

27) В этой задаче нужно считать количество элементов, делящихся на первый элемент массива, и одновременно накапливать их сумму. Каждый раз, когда найден положительный элемент, остаток от деления которого на **a[1]** дает 0, нужно увеличить счетчик на 1 и добавить к сумме значение этого элемента. Искомое среднее арифметическое получим как частное от деления суммы на количество найденных элементов.

Решение на естественном языке. Записываем в переменные **x** и **y** – нулевые начальные значения. В цикле перебираем значения переменной **i** от 1 до **N**. Если очередной элемент больше нуля и при делении на a[1] дает в остатке 0, увеличиваем переменную y на 1 и добавляем текущий элемент к предыдущему значению переменной **x**. После окончания цикла делим **x** на **y** и записываем результат в переменную **s**. Выводим значение **s**.

Решение на Паскале.

```
const N=40;
var a: array[1..N] of integer;
    i, x, y: integer;
    s: real;
begin
for i:=1 to N do readln(a[i]);
```



```
x:= 0; y:= 0;
for i:=1 to N do
  if (a[i] > 0) and (a[i] mod a[1] = 0) then begin
    y:= y + 1;
    x:= x + a[i]
  end;
s:= x / y;
writeln(s)
end.
```

28) В этой задаче есть хитрость, связанная с особенностями вычисления остатка от деления отрицательных чисел в языках программирования. Чтобы понять суть проблемы, нужно вернуться к основам математики (теории чисел). Дело в том, что в теории чисел остаток от деления – положительное число. Например, при делении числа «–31» на 5 получается «–7» и остаток 4, потому что

$$-31 = 5 \cdot (-7) + 4$$

Однако в большинстве современных языков программирования (в том числе в Паскале и в Си) частное и остаток вычисляются иначе, не в соответствии с теорией чисел. Например, при делении числа «–31» на 5 получается «–6» и остаток «–1». Эта особенность может сыграть свою роль. Например, при проверке нечетности числа следующий оператор для отрицательных чисел сработает неверно :

```
  if A[i] mod 2 = 1 then
    writeln('Число ', A[i], ' нечетное');
```

Дело в том, что для нечетных отрицательных чисел этот остаток будет равен -1 и условный оператор не сработает. Правильно писать так:

```
  if A[i] mod 2 <> 0 then
    writeln('Число ', A[i], ' нечетное');
```

Заметим, что в алгоритмическом языке (в системе КуМир) такой проблемы нет, КуМир считает остаток правильно с точки зрения математики – для всех нечетных чисел остаток будет равен 1.
Вторая проблема – какое число записать в переменную **min** в качестве начального значения? Поскольку диапазон чисел по условию ограничен (от -1000 до 1000), для этого можно использовать любое число вне этого диапазона, например, 9999.

Решение на естественном языке. Записываем в переменную **min** начальное значение 9999, которого по условию не может быть в исходном массиве. В цикле перебираем значения переменной **i** от 1 до **N**. Если очередной элемент массива **a[i]** нечетный (остаток от его деления на 2 не равен нулю) и при делении на 5 дает в остатке 0, то сравниваем его со значением переменной **min**. Если этот элемент меньше, чем **min**, записываем в переменную **min** значение **a[i]**. После окончания цикла выводим значение **min**.

Решение на Паскале.

```
const N=20;
var a: array[1..N] of integer;
    i, j, min: integer;
begin
for i:=1 to N do readln(a[i]);
min:= 9999;
for i:=1 to N do
  if (a[i] mod 2 <> 0) and
     (a[i] mod 5 = 0) then
    if a[i] < min then min:=a[i];
```

```
writeln(min)
end.
```

Еще один вариант, в котором все условия объединены в одном условном операторе (читатель **Ion**).

Решение на Паскале.

```
const N=20;
var a: array[1..N] of integer;
    i, j, min: integer;
begin
for i:=1 to N do readln(a[i]);
min:= 9999;
for i:=1 to N do
  if (a[i] mod 2 <> 0) and
     (a[i] mod 5 = 0) and
     (a[i] < min) then min:=a[i];
writeln(min);
end.
```

29) Простая задача. Приведем сразу решение.

Решение на естественном языке. Записываем в переменную **s** начальное значение 0. В цикле перебираем значения переменной **i** от 1 до **N**. Если значение очередного элемента массива **a[i]** кратно 13 (остаток от его деления на 13 равен нулю), то считаем сумму значения **a[i]** и переменной **s** и записываем ее в переменную **s**. После окончания цикла выводим значение **s**.

Решение на Паскале.

```
const N=30;
var a: array[1..N] of integer;
    i, j, s: integer;
begin
for i:=1 to N do readln(a[i]);
s:= 0;
for i:=1 to N do
  if a[i] mod 13 = 0 then
    s:= s + a[i];
writeln(s);
end.
```

30) В этой задаче нужно найти среднее арифметическое нечётных трехзначных чисел, то есть таких, для которых выполняется условие **100<=X<=999**. В одной целой переменной (с именем **s**) будем накапливать сумму нужных элементов, а в другой (с именем **j**) будем считать эти элементы.

Решение на естественном языке. Записываем в переменные **s** и **j** начальные значения 0. В цикле перебираем значения переменной **i** от 1 до **N**. Если значение очередного элемента массива **a[i]** одновременно нечётное, больше 99 и меньше 1000, увеличиваем переменную **s** на значение **a[i]** и увеличиваем значение переменной **j** (счётчик найденных чисел) на 1. После окончания цикла проверяем значение **j**: если оно равно 0, то выводим сообщение «Нет таких чисел», иначе выводим отношение **s/j**.

Решение на Паскале.

```
const N=30;
var a: array[1..N] of integer;
    i, j, s: integer;
```



```
begin
for i:=1 to N do readln(a[i]);
s:= 0; j:= 0;
for i:=1 to N do
  if (99 < a[i]) and (a[i] < 1000) and
     (a[i] mod 2 <> 0) then begin
    s:= s + a[i];
    j:= j + 1;  { или Inc(j); }
  end;
if j = 0 then
    writeln('Нет таких чисел')
else writeln(s/j)
end.
```

31) В этой задаче в программе на Паскале использован тип **longint**: это длинное целое число размером 32 бита. Цель – сделать так, чтобы произведение нескольких целых чисел, каждое из которых находится в интервале [1;100], поместилось в такую переменную. Заметим, что все переменные, кроме **p** (в которой накапливается произведение), можно было отнести к типу **integer**.

Нужно найти произведение чисел, поэтому начальное значения для переменной **p** нужно взять равным 1. Отбираем все четные числа (то есть, остаток от деления на 2 равен 0), которые не оканчиваются на 0, то есть остаток от их деления на 10 не равен 0.

Решение на естественном языке. Записываем в переменную **p** начальное значение 1. В цикле перебираем значения переменной **i** от 1 до **N**. Если значение очередного элемента массива **a[i]** при делении на 2 даёт остаток 0, а при делении на 10 – не 0, умножаем переменную **p** на значение **a[i]**. После окончания цикла выводим значение **p**.

Решение на Паскале.

```
const N=30;
var a: array [1..N] of longint;
    i, j, p: longint;
begin
  for i:=1 to N do readln(a[i]);
  p:= 1;
  for i:=1 to N do
  if (a[i] mod 2 = 0) and (a[i] mod 10 <> 0) then
    p:= p*a[i];
  writeln(p)
end.
```

32) Эта задача полностью аналогична задаче 31.

Решение на естественном языке. Записываем в переменную **p** начальное значение 1. В цикле перебираем значения переменной **i** от 1 до **N**. Если значение очередного элемента массива **a[i]** больше или равно 10, меньше или равно 99 и при делении на 6 даёт ненулевой остаток (все условия должны выполняться одновременно), умножаем переменную **p** на значение **a[i]**. После окончания цикла выводим значение **p**.

Решение на Паскале.

```
const N=30;
var a: array [1..N] of longint;
    i, j, p: longint;
begin
  for i:=1 to N do readln(a[i]);
```

```
  p:= 1;
  for i:=1 to N do
  if (10 <= a[i]) and (a[i] <= 99) and
     (a[i] mod 6 <> 0) then
    p:= p*a[i];
  writeln(p)
end.
```

33) Согласно условию, нужно вывести «наименьшее положительное нечетное число», содержащееся в массиве, то есть, выбирать нужно из чисел, которые больше нуля и не делятся на 2, то есть дают остаток 1 при делении на 2. Тогда получаем такой цикл:

```
  if число a[i] подходит и
     число a[i] < минимального then
    минимальное:= a[i];
```

Минимальное число будем хранить в переменной **m**. Возникает вопрос: какое должно быть начальное значение этой переменной? Мы не можем принять его равным **a[1]**, как это обычно делается при поиске минимального из всех, потому что первый элемент массива может оказаться неподходящим (чётным или отрицательным). Это начальное значение должно отличаться от всех «подходящих» элементов массива, чтобы мы могли обнаружить ситуацию, когда встретилось первое подходящее число. Например, можно принять его равным нулю. Тогда получаем псевдокод

```
  if число a[i] подходит и
     (это первое подходящее число или
      число a[i] < минимального) then
    минимальное:= a[i];
```

Условие «это первое подходящее число» запишется как «**m = 0**».

Решение на естественном языке. Записываем в переменную **m** начальное значение 0. В цикле перебираем значения переменной **i** от 1 до **N**. Если значение очередного элемента массива **a[i]** больше нуля и при делении на 2 даёт в остатке 1, а также **m=0** или значение **a[i]** меньше, чем **m**, записываем в переменную **m** значение **a[i]**. После окончания цикла выводим значение **m**.

Решение на Паскале.

```
const N=70;
var a: array [1..N] of integer;
    i, j, m: integer;
begin
  for i:=1 to N do readln(a[i]);
  m:= 0;
  for i:=1 to N do
  if (a[i] > 0) and (a[i] mod 2 = 1) and
     ((a[i] < m) or (m = 0)) then
    m:= a[i];
  writeln(m)
end.
```

34) Согласно условию, нужно вывести наименьший положительный элемент массива, десятичная запись которого оканчивается на 7, то есть, при делении на 10 (основание системы счисления) дает остаток 7: **a[i] mod 10 = 7**.

Тогда получаем такой цикл:

```
  if число a[i] подходит и
     число a[i] < минимального then
    минимальное:= a[i];
```


Минимальное число будем хранить в переменной **m**. Возникает вопрос: какое должно быть начальное значение этой переменной? Мы не можем принять его равным **a[1]**, как это обычно делается при поиске минимального из всех, потому что первый элемент массива может оказаться неподходящим (чётным или отрицательным). Это начальное значение должно отличаться от всех «подходящих» элементов массива, чтобы мы могли обнаружить ситуацию, когда встретилось первое подходящее число. Например, можно принять его равным нулю. Тогда получаем псевдокод

```
  if число a[i] подходит и
     (это первое подходящее число или
      число a[i] < минимального) then
    минимальное:= a[i];
```

Условие «это первое подходящее число» запишется как «**m = 0**».

Решение на естественном языке. Записываем в переменную **m** начальное значение 0. В цикле перебираем значения переменной **i** от 1 до **N**. Если значение очередного элемента массива **a[i]** больше нуля и при делении на 10 даёт в остатке 7, а также **m=0** или значение **a[i]** меньше, чем **m**, записываем в переменную **m** значение **a[i]**. После окончания цикла выводим значение **m**.

Решение на Паскале.

```
const N=70;
var a: array [1..N] of integer;
    i, j, m: integer;
begin
  for i:=1 to N do readln(a[i]);
  m:= 0;
  for i:=1 to N do
  if (a[i] > 0) and (a[i] mod 10 = 7) and
     ((a[i] < m) or (m = 0)) then
    m:= a[i];
  writeln(m)
end.
```

35) У этой задачи есть одна новая особенность: требуется найти ***эффективный*** алгоритм решения. Что это значит? Итак, нужно найти пару элементов массива, разность которых максимальна. Можно, например, сделать двойной цикл и проверять разности для каждой пары элементов (запоминая максимальную):

```
x := 0;
for i:=1 to N do
  for j:=1 to N do
    if abs(a[i]-a[j]) > x then
      x:= abs(a[i]-a[j]);
```

но этот алгоритм будет неэффективным, потому что имеет сложность $O(N^2)$: при увеличении размера массива N в k раз количество сравнений увеличивается в k^2 раз.

Будем искать лучшее решение, позволяющее найти ответ за один проход по массиву (без вложенного цикла). Легко понять, что наибольшая разность между максимальным и минимальным элементами, их можно легко найти за один проход с помощью классических алгоритмов.

Решение на естественном языке. Записываем в переменные **x** и **y** начальные значения **a[1]**. В цикле перебираем значения переменной **i** от 1 до **N**. Если значение очередного элемента массива **a[i]** больше, чем **x**, записываем в переменную **x** значение **a[i]**. Если значение очередного элемента массива **a[i]** меньше, чем **y**,

записываем в переменную **y** значение **a[i]**. После окончания цикла выводим значение **x-y**.

Решение на Паскале.

```
const N=70;
var a: array [1..N] of integer;
    i, j, x, y: integer;
begin
  for i:=1 to N do readln(a[i]);
  x:= a[1]; y:= a[1];
  for i:=1 to N do begin
    if a[i] > x then x:= a[i];
    if a[i] < y then y:= a[i];
  end;
  writeln(x-y)
end.
```

36) Нужно найти и вывести наименьшую нечётную сумму двух соседних элементов массива. Для вычисления текущей суммы двух соседних элементов массива будем использовать переменную **x**, а для хранения наименьшей суммы – переменную **y**. В цикле рассматриваем все пары соседних элементов, если их сумма нечётная (**x mod 2 <> 0**) и меньше, чем **y**, записываем эту сумму в **y**.

Осталось решить проблему начального значения для **y** – как определить, что ни одна подходящая пара еще не была найдена? Для этого можно использовать тот факт, что по условию нужная сумма – *нечётная*. Поэтому в качестве начального значения можно принять любое *чётное* число, например, **y = 0** (если же требуется искать *чётную* сумму, берём любое *нечётное* число).

Решение на естественном языке. Записываем в переменную **y** начальное значение 0. В цикле перебираем значения переменной **i** от 1 до **N-1**. На каждом шаге цикла в переменную **x** записываем сумму очередного элемента массива **a[i]** и следующего элемента **a[i+1]**. Если эта сумма нечётная (при делении на 2 даёт ненулевой остаток) и меньше, чем **y**, или **y=0**, записываем значение **x** в переменную **y**. После окончания цикла выводим значение переменной **y**.

Решение на Паскале.

```
const N=70;
var a: array [1..N] of integer;
    i, j, x, y: integer;
begin
  for i:=1 to N do readln(a[i]);
  y:= 0;
  for i:=1 to N-1 do begin
    x:= a[i] + a[i+1];
    if (x mod 2 <> 0) and
       ((y = 0) or (x < y))
       then y:=x
  end;
  writeln(y)
end.
```

37) Нужно найти сумму элементов, удовлетворяющих некоторому условию. Общее решение этой задачи выглядит так:

```
s:=0;
for i:=1 to 2014 do
  if <условие выполняется> then
    s := s + a[i];
```


В данной задаче условие записывается так:

• число трёхзначное
• последняя цифра числа – 9
• две последних цифры числа – НЕ 99
	`(100 <= a[i]) and (a[i] <= 999)`
	`(a[i] mod 10 = 9)`
	`(a[i] mod 100 <> 99)`

Для проверки трёхзначности числа проверяем принадлежность диапазону [100,999], последняя цифра – это остаток от деления на 10, а две последних цифры – остаток от деления на 100. Все условия нужно объединить с помощью операции «и» (**and**).

Теперь внимательно почитаем, что нужно вывести, если таких чисел нет – по условию ответ должен быть «-1». Поскольку все числа неотрицательные, сумма в этом случае будет равна нулю. Вот полная программа

```
const N=2014;
var a: array [1..N] of integer;
    i, j, s: integer;
begin
  for i:=1 to N do
   readln(a[i]);
  s:=0;
  for i:=1 to 2014 do
  if (100 <= a[i]) and (a[i] <= 999) and
     (a[i] mod 10 = 9) and
     (a[i] mod 100 <> 99)
    then s := s + a[i];
  if s = 0 then
       writeln(-1)
  else writeln(s)
end.
```

38) В цикле нужно сравнивать очередной элемент массива (**a[i]**) с удвоенным предыдущим то есть с **2*a[i-1]**. Если выполняется условие **a[i]>2*a[i-1]**, то увеличиваем счётчик (в качестве счётчика можно использовать переменную **j** или **k**). Единственная проблема – не забыть начать цикл не с 1, а с 2, чтобы не обратиться к несуществующему элементу **a[0]**:

```
k:=0;
for i:=2 to N do
  if a[i] > 2*a[i-1] then
    k:=k+1;
writeln(k);
```

Еще один вариант правильного решения, использующий деление:

```
k:=0;
for i:=2 to N do
  if a[i]/a[i-1] > 2 then
    k:=k+1;
writeln(k);
```

39) В цикле нужно находить сумму элемента массива (**a[i]**) с симметричным ему элементом, который имеет номер **N-i+1** (так чтобы для **i=1** мы получили **N**). Если выполняется эта сумма больше 20, то увеличиваем счётчик – переменную **k**.

ВАЖНО: поскольку мы просматриваем массив фактически сразу с двух концов, нужно остановиться на середине, то есть в цикле сделать **N div 2** шагов, а не **N**:

```
k:=0;
for i:=1 to N div 2 do
  if a[i]+a[N+1-i] > 20 then
    k:= k + 1;
writeln(k);
```

Второй вариант – использовать вспомогательную переменную **j**, которая будет изменяться, начиная с **N**, и уменьшаться на 1 с каждым шагом цикла:

```
k:= 0;
j:= N;
for i:=1 to N div 2 do begin
  if a[i]+a[j] > 20 then
    k:= k + 1;
  j:= j - 1
end;
writeln(k);
```

40) Задача достаточно проста, но с одним подвохом. Очевидно, что условие локального минимума для элемента **a[i]** запишется так: **a[i-1] > a[i] < a[i+1]**. Напишем цикл, где переменная **i** изменяется от 2 дол N-1 (чтобы не выйти за границы массива):

```
k:=0;
for i:=2 to N-1 do
  if (a[i-1] > a[i]) and (a[i] < a[i+1]) then
    k:=k+1;
```

Но это ещё не все, поскольку есть крайние элементы, **a[1]** и **a[N]**. У них только один сосед, и из нужно проверять отдельно, вне цикла. Логичнее проверить первый элемент перед циклом, а последний – после цикла, но это не принципиально. Окончательное решение:

```
k:=0;
if a[1] < a[2] then k:=k+1;
for i:=2 to N-1 do
  if (a[i-1] > a[i]) and (a[i] < a[i+1]) then
    k:=k+1;
if a[N-1] > a[N] then k:=k+1;
writeln(k);
```

Не забываем выводить ответ!

41) В этой довольно простой задаче нужно не забыть три момента:

1. В программе на Паскале каждое из отношений в составе сложного условия нужно брать в скобки.
2. Если рассматриваются пары **(a[i],a[i+1])**, то перебор значений **i** в цикле нужно делать от 1 до **N-1**, а не до **N** (чтобы не было выхода за границы массива).
3. Для проверки нечётности произведения нужно взять остаток от его деления на 2, причём в Паскале (и в Си) этот остаток может быть как положительным, так и отрицательным (если произведение меньше нуля), поэтому нужно использовать условие

```
a[i]*a[i+1] mod 2 <> 0
```

а не

```
a[i]*a[i+1] mod 2 = 1
```

Окончательное решение:

```
k:=0;
for i:=1 to N-1 do
  if (a[i]*a[i+1] mod 2 <> 0) and (a[i]+a[i+1] > 0) then
```



```
    k:=k+1;
writeln(k);
```

Не забываем выводить ответ!

42) Решение полностью аналогично предыдущей задаче:

```
k := 0;
for i := 1 to N-1 do
  if ((a[i]+a[i+1]) mod 2=0) and
     ((a[i]+a[i+1]) mod 4<>0) then
    k := k + 1;
writeln(k);
```

43) Решение полностью аналогично предыдущим задачам:

```
k := 0;
for i := 1 to N-1 do
  if ((a[i]+a[i+1]) mod 7 = 0) and
     (a[i]*a[i+1] > 0) then
    k := k + 1;
writeln(k);
```

44) Решение полностью аналогично предыдущим задачам:

```
k := 0;
for i := 1 to N-1 do
  if ((a[i]+a[i+1]) mod 6 <> 0) and
     (a[i]*a[i+1] < 1000) then
    k := k + 1;
writeln(k);
```

45) В этой задаче нам нужно подсчитать:

– количество чётных элементов;
– количество нечётных элементов;
– максимальный чётный элемент;
– максимальный нечётный элемент;

Ещё одну переменную нужно использовать как переменную цикла, поэтому всего требуется 5 переменных. А нам разрешено использовать только 4!

Чтобы выйти из положения, нужно сообразить, что достаточно искать только количество чётных элементов, а количество нечётных будет равно N минус количество чётных.

Поскольку по условию все элементы неотрицательны, начальные значения для поиска максимальных элементов можно взять равными нулю (или отрицательными):

```
j:= 0;  { количество чётных }
k:=-1; { максимальный чётный }
m:=-1; { максимальный нечётный }
for i:=1 to N do begin
  if a[i] mod 2 = 0 then begin { если чётный }
    j:= j + 1;
    if a[i] > k then k:=a[i];
  end
  else { если нечётный }
    if a[i] > m then m:=a[i];
end;
if j >= N-j then { если чётных больше }
     writeln(k)
else writeln(m); { если нечётных больше }
```

46) Решение полностью аналогично решению предыдущей задачи. Начальные значения для поиска минимумов нужно выбрать больше, чем максимальное возможное значение элементов массива:

```
j:=0;
k:=10001;
m:=10001;
for i:=1 to N do begin
  if a[i] mod 2 = 0 then begin
     j:= j + 1;
     if a[i] < k then k:=a[i];
  end
  else
  if a[i] < m then m:=a[i];
end;
if j <= N-j then
     writeln(k)
else writeln(m);
```

47) Решение аналогично решению задачи 44:

```
k := 0;
for i := 1 to N-1 do
  if (a[i] mod 3 = 0) and
     (a[i+1] mod 3 = 0) then
    k := k + 1;
writeln(k);
```

48) В этой задаче нужно во время обработки прохода по массиву считать

а) сумму всех элементов
б) количество чётных элементов
в) количество нечётных элементов.

Главная проблема состоит в том, что у нас есть всего две переменные (**i** и **k**). Например, можно сначала найти сумму всех элементов, а затем, в зависимости от чётности этой сумму, определять результат на втором проходе по массиву (напомним, что в этом задании эффективность программы не оценивается!):

```
k:=0;
for i:=1 to N do k:=k+a[i];
if k mod 2 = 0 then begin
  k:=0;
  for i:=1 to N do
    if a[i] mod 2 <> 0 then k:= k + 1;
end
else begin
  k:=0;
  for i:=1 to N do
    if a[i] mod 2 = 0 then k:= k + 1;
end;
writeln(k);
```

Можно сделать и красиво, за один проход. Это решение основано на двух идеях:

а) количество чётных элементов + количество нечётных элементов равно N
б) сумма всех элементов нечётна тогда и только тогда, когда количество нечётных элементов нечётно.

Это значит, что достаточно подсчитать только количество нечётных элементов **k**. Если оно окажется нечётно, то сумма элементов нечётная и нужно вывести количество чётных элементов, которое равно **N-k**.

```
k:=0;
for i:=1 to N do
  if a[i] mod 2 <> 0 then k:=k+1;
if k mod 2 = 1 then
  writeln(N-k)
else
  writeln(k);
```

49) Очень простая задача с одной ловушкой – нельзя выйти за границы массива. Например, при нумерации элементов массива, начиная с 1, заканчивать цикл нужно на паре **(a[N-1],a[N])**. Если переменная цикла хранит номер первого элемента в паре, заканчивать цикл нужно на **N-1**:

```
k := 0;
for i := 1 to N-1 do
  if (a[i] mod 3=0) or (a[i+1] mod 3=0) then
    k := k + 1;
writeln(k);
```

50) Эта задача аналогична задаче 49. Нужно только уметь определять последнюю десятичной записи цифру числа как остаток от деления этого числа на 10 и не выйти за границы массива, то есть закончить цикла на значение **i=N-1**:

```
k := 0;
for i := 1 to N-1 do
  if (a[i] mod 10 = 5) or (a[i+1] mod 10 = 5) then
    k := k + 1;
writeln(k);
```

51) Если шестнадцатеричная запись числа заканчивается на F, это означает, что остаток от деления этого числа на 16 равен 15. Перед циклом перебора элементов массива нужно записать в переменную **k**, которая будет хранить максимальное подходящее значение, число 0. Если ни одного подходящего числа мы не найдем, в этой переменной останется 0 и он будет выведен в качестве результата. Заметьте, что по условию все числа положительные и нуля среди них быть не может.

```
k := 0;
for i := 1 to N do
  if (a[i] mod 16 = 15) and (a[i] > k) then
    k := a[i];
writeln(k);
```

52) Эта задача полностью аналогична предыдущей, за исключением выбора начального значения для переменной **k**. Можно, например, использовать тот факт, что все значения элементов массива не превышают 10000. Сначала записываем в переменную **k** значение 10001 (или любое значение, большее 10000). Если это значение не изменится после работы цикла (не нашли ни одного подходящего числа), выводим 0.

```
k := 10001;
for i := 1 to N do
  if (a[i] mod 8 = 7) and (a[i] < k) then
    k := a[i];
if k = 10001 then
  writeln(0)
else
```

```
  writeln(k);
```

Возможен другой вариант – сначала присвоить переменной **k** значение 0 и усложнить условие: будем изменять **k**, если значение подходящего элемента массива меньше **k** или **k** = 0. В этом случае не требуется ветвление при выводе результата, так как в переменной **k** останется 0, если ни одного подходящего значения не нашли.

```
k := 0;
for i := 1 to N do
  if (a[i] mod 8 = 7) and
     ((a[i] < k) or (k = 0)) then
    k := a[i];
writeln(k);
```

53) Если восьмеричная запись числа содержит три цифры, то число находится в диапазоне $[8^2; 8^3-1]$, то есть [64; 511]. В остальном решение аналогично решению задачи 51:

```
k := 0;
for i := 1 to N do
  if (64 <= a[i]) and (a[i] < 512) and (a[i] > k) then
    k := a[i];
writeln(k);
```

54) Если шестнадцатеричная запись числа содержит две цифры, то число находится в диапазоне $[16^1; 16^2-1]$, то есть [16; 255]. В остальном решение аналогично решению задачи 52:

```
k := 10001;
for i := 1 to N do
  if (16 <= a[i]) and (a[i] <= 255) and (a[i] < k) then
    k := a[i];
if k > 10000 then
  writeln(0)
else
  writeln(k);
```

55) Если восьмеричная запись числа содержит не менее трёх цифр, то число больше или равно $8^2 = 64$. Если в восьмеричной системе запись оканчивается на 5, то остаток от деления этого числа на 8 равен 5. В остальном решение аналогично решению задачи 51:

```
k := 0;
for i := 1 to N do
  if (64 <= a[i]) and (a[i] mod 8 = 5) and (a[i] > k) then
    k := a[i];
writeln(k);
```

56) Если шестнадцатеричная запись числа содержит не менее трёх цифр, то число больше или равно $16^2 = 256$. Если шестнадцатеричная запись заканчивается на букву С, то остаток от деления числа на 16 равен 12. В остальном решение аналогично решению задачи 52:

```
k := 10001;
for i := 1 to N do
  if (256 <= a[i]) and (a[i] mod 16 = 12) and (a[i] < k) then
    k := a[i];
if k > 10000 then
  writeln(0)
else
  writeln(k);
```

57) Если восьмеричная запись числа содержит ровно две цифры, то число находится в диапазоне $[8^1; 8^2-1]$, то есть [8; 63]. В этом случае первая цифра находится как результат

деления нацело на 8, а вторая цифра – как остаток от деления на 8. Приведём полное решение:

```
k := 0;
for i := 1 to N do
  if (8 <= a[i]) and (a[i] <= 63) and
     (a[i] div 8 < a[i] mod 8) then
    k := k + 1;
writeln(k);
```

58) Если шестнадцатеричная запись числа содержит две цифры, то число находится в диапазоне $[16^1; 16^2-1]$, то есть [16; 255]. В этом случае первая цифра находится как результат деления нацело на 16, а вторая цифра – как остаток от деления на 16. В остальном решение аналогично решению задачи 54:

```
k := 10001;
for i := 1 to N do
  if (16 <= a[i]) and (a[i] < 256) and
     (a[i] div 16 > a[i] mod 16) and
     (a[i] < k) then
    k := a[i];
if k > 10000 then
  writeln(0)
else
  writeln(k);
```

59) Эту задачу уже не решить с помощью одного цикла, потому что, взяв очередной элемент массива **a[i]**, нужно проверить, не встречалось ли оно раньше. Это можно сделать, например, так:

```
j := 0;
while a[j] <> a[i] do
  j := j + 1;
```

Если такого элемента раньше не было, то после окончания цикла мы получим **j=i** (остановимся на этом же элементе), и нужно увеличивать счетчик. А такое же число раньше встречалось, цикл остановится при **j<i** и счётчик не увеличиваем.

Вот полное решение задачи:

```
count := 0;
for i := 0 to N - 1 do begin
  j := 0;
  while a[j] <> a[i] do
    j := j + 1;
  if j = i then
    count := count + 1
end;
writeln(count);
```

60) Решение этой задачи аналогично решению задачи 58:

```
k := 10001;
for i := 1 to N do
  if (16 <= a[i]) and (a[i] < 256) and
     (a[i] mod 16 >= 10) and (a[i] < k) then
    k := a[i];
if k > 10000 then
  writeln(0)
else
  writeln(k);
```

61) Особенность этой задачи в том, что в решении нужно использовать двухпроходный алгоритм. В результате первого прохода по массиву (первого цикла) мы должны определить, сколько там есть нужных элементов, а затем при втором проходе (во втором цикле) заменяем все эти элементы на их количество и сразу выводим их на экран. Вывод на экран можно оформить и отдельно, как третий цикл.

Вот стандартное решение на языке Паскаль в два прохода:

```
k := 0;
for i:=1 to N do
  if (a[i] mod 3 = 0)
      and (a[i] mod 10 = 1) then
    k := k+1;
for i:=1 to N do begin
  if (a[i] mod 3 = 0)
      and (a[i] mod 10 = 1) then a[i] := k;
  writeln(a[i])
end;
```

Решение на языке Паскаль в три прохода:

```
k := 0;
for i:=1 to N do
  if (a[i] mod 3 = 0)
      and (a[i] mod 10 = 1) then
    k := k+1;
for i:=1 to N do
  if (a[i] mod 3 = 0)
      and (a[i] mod 10 = 1) then a[i] := k;
for i:=1 to N do
  writeln(a[i])
end;
```

Обратите внимание, что по условию задачи **обязательно изменить массив**. То есть, если вы заменяете элементы только при выводе, оценка будет снижена на 1 балл. Вот это неправильное решение:

```
k := 0;
for i:=1 to N do
  if (a[i] mod 3 = 0)
      and (a[i] mod 10 = 1) then
    k := k+1;
for i:=1 to N do begin
  if (a[i] mod 3 = 0)             // это неверно!
      and (a[i] mod 10 = 1) then // массив не изменяется!
    writeln(k)
  else
    writeln(a[i])
end;
```

Решение на языке Python:

```
k = len( [s for s in a
            if s % 3 == 0 and s % 10 == 1] )
for i in range(n):
  if a[i] % 3 == 0 and a[i] % 10 == 1:
    a[i] = k
  print(a[i])
```

В первой строке составляется новый массив из всех нужных элементов и в переменной k сохраняется его длина.

62) Очевидно, что на первом проходе нужно определить минимальный элемент из тех, которые больше, чем 50. Поскольку по условию все элементы не больше 1000, в качестве начального значения для переменной, где будет храниться минимальное значение, можно взять 1001 – все элементы в массиве должны быть меньше него. Дальше применяем стандартный алгоритм поиска минимального элемента:

```
k := 1001;
for i:=1 to N do
  if (a[i] > 50) and (a[i] < k) then
    k := a[i];
```

Чтобы минимальный из элементов, больших 50, стал равен 50, нужно вычесть из него значение (**k** – 50), поэтому сразу уменьшаем **k** на 50. После этого во втором цикле меняем нужные элементы массива и сразу выводим их на экран.

Вот полная программа:

```
k := 1001;
for i:=1 to N do
  if (a[i] > 50) and (a[i] < k) then
    k := a[i];
k := k - 50;
for i:=1 to N do begin
  if a[i] > 50 then
    a[i] := a[i] - k;
  writeln(a[i])
end;
```

Важно убедиться, что программа будет работать и в том случае, когда в массиве нет ни одного элемента больше 50. В этом случае значение **k** останется равным 1001, но во втором цикле оно не будет использовано.

Решение на языке **Python**:

```
k = [s for s in a if s > 50]
if k:
  k = min(k) - 50
for i in range(n):
  if a[i] > 50:
    a[i] -= k
  print(a[i])
```

Обратите внимание, что следующий вариант решения не получит полный балл:

```
k = min( [s for s in a if s > 50] ) - 50  // ошибка!
for i in range(n):
  if a[i] > 50:
    a[i] -= k
  print(a[i])
```

Дело в том, что если в массиве нет элементов, больших 50, работа функции **min** завершится аварийно. В версиях Python, начиная с 3.4, можно указать дополнительный аргумент с именем **default** (значение по умолчанию):

```
k = min( [s for s in a if s > 50], default=0 )
```

При этом в случае пустого массива в переменную **k** будет записано значение 0.

63) Эта задача полностью аналогична предыдущей, только вместо минимума нужно искать максимум среди элементов, которые больше, чем 80. Поскольку по условию все элементы положительны, в качестве начального значения для переменной, где будет храниться максимальное значение, можно взять 0 – все элементы в массиве должны быть больше него.

Вот полная программа на Паскале:

```
k := 0;
for i:=1 to N do
  if (a[i] > 80) and (a[i] > k) then
    k := a[i];
k := k - 80;
for i:=1 to N do begin
  if a[i] > 80 then
    a[i] := a[i] - k;
  writeln(a[i])
end;
```

Решение на языке **Python**:

```
k = [s for s in a if s > 80]
if k:
  k = max(k) - 80
for i in range(n):
  if a[i] > 80:
    a[i] -= k
  print(a[i])
```

64) Эта задача полностью аналогична задаче 62.

Вот полная программа на Паскале:

```
k := 30001;
for i:=1 to N do
  if (a[i] mod 2 = 0) and (a[i] < k) then
    k := a[i];
k := k - 20;
for i:=1 to N do begin
  if a[i] mod 2 = 0 then
    a[i] := a[i] - k;
  writeln(a[i])
end;
```

Решение на языке **Python**:

```
k = [s for s in a if s % 2 == 0]
if k:
  k = min(k) - 20
for i in range(n):
  if a[i] % 2 == 0:
    a[i] -= k
  print(a[i])
```

65) Эта задача полностью аналогична задаче 63.

Вот полная программа на Паскале:

```
k := 0;
for i:=1 to N do
  if (a[i] mod 2 = 1) and (a[i] > k) then
    k := a[i];
k := k - 1;
for i:=1 to N do begin
  if a[i] mod 2 = 1 then
    a[i] := a[i] - k;
  writeln(a[i])
end;
```

Решение на языке **Python**:


```
k = [s for s in a if s % 2 == 1]
if k:
  k = max(k) - 1
for i in range(n):
  if a[i] % 2 == 1:
    a[i] -= k
  print(a[i])
```

66) Эта задача сводится к тому, чтобы найти максимум из тех элементов, которые меньше 40, и затем найти максимальный дополнительный множитель, разделив 10000 на этот максимум (нацело).

Решение на **Паскале**:

```
k := 0;
for i:=1 to N do
  if (a[i] < 40) and (a[i] > k) then
    k := a[i];
k := 10000 div k;
for i:=1 to N do begin
  if a[i] < 40 then
    a[i] := a[i] * k;
  writeln(a[i])
end;
```

Решение на языке **Python**:

```
k = [s for s in a if s < 40]
if k:
  k = 10000 // max(k)
for i in range(n):
  if a[i] < 40:
    a[i] *= k
  print(a[i])
```

67) Сначала нужно найти сумму элементов, которые меньше 200 и делятся на 5. Затем при втором проходе по массиву заменяем все эти элементы на эту сумму.

Решение на **Паскале**:

```
s := 0;
for i:=1 to N do
  if (a[i] < 200) and (a[i] mod 5 = 0) then
    s := s + a[i];
for i:=1 to N do begin
  if (a[i] < 200) and (a[i] mod 5 = 0) then
    a[i] := s;
  writeln(a[i])
```

Решение на языке **Python**:

```
s = sum( [k for k in a
            if k < 200 and k % 5 == 0] )
for i in range(n):
  if a[i] < 200 and a[i] % 5 == 0:
    a[i] = s
  print(a[i])
```

68) Сглаживание выполняется очень просто: для всех элементов, кроме крайних, нужно применить формулу

```
a[i] = (a[i-1] + a[i] + a[i+1]) div 3;
```

Для крайних элементов находим среднее арифметическое двух значений, их нужно обрабатывать отдельно:

```
a[1] = (a[1] + a[2]) div 2;
a[N] = (a[N-1] + a[N]) div 2;
```

Но вот такое решение неверно:

```
a[1] := (a[1] + a[2]) div 2;
writeln(a[1]);
for i:=2 to N-1 do begin
  a[i] := (a[i-1] + a[i] + a[i+1]) div 3; // ошибка!
  writeln(a[i]);                          // a[i-1] изменено!
end;
a[N] := (a[N-1] + a[N]) div 2;  // ошибка! a[N-1] изменено!
writeln(a[N]);
```

Дело в том, что когда мы вычисляем новое значение **a[2]**, мы берём уже измененное значение **a[1]**, а нам нужно брать исходное значение этого элемента, до изменений. Поэтому предыдущий элемент нужно где-то запоминать. Кроме того, когда мы записываем новое значение **a[i]**, его старое значение тоже нужно где-то запомнить. для этих целей будем использовать две переменных:

- **k** – значение предыдущего элемента, a[i-1]
- **j** – значение элемента **a[i]**, которое мы сейчас будем изменять (оно нужно для следующего шага цикла, когда будем определять новое значения **a[i-1]**).

Сначала обработаем **a[1]**, предварительно запомнив его значение в переменной **j**, и сразу выводим на экран:

```
j := a[1];
a[1] := (a[1] + a[2]) div 2;
writeln(a[1]);
```

Обрабатываем в цикле все элементы, кроме первого и последнего:

```
for i:=2 to N-1 do begin
  k := j;        // сохранили предыдущий элемент
  j := a[i];     // запомнили текущий элемент
  a[i] := (k + j + a[i+1]) div 3;
  writeln(a[i]);
end;
```

Теперь остался последний элемент:

```
a[N] := (j + a[N]) div 2;
writeln(a[N]);
```

Полное решение на **Паскале**:

```
j := a[1];
a[1] := (a[1] + a[2]) div 2;
writeln(a[1]);
for i:=2 to N-1 do begin
  k := j;
  j := a[i];
  a[i] := (k + j + a[i+1]) div 3;
  writeln(a[i]);
end;
a[N] := (j + a[N]) div 2;
writeln(a[N]);
```

Решение на языке **Python**:

```
j = a[0];
a[0] = (a[0] + a[1]) // 2
print(a[0])
for i in range(1,n-1):
  k = j
  j = a[i]
  a[i] = (k + j + a[i+1]) // 3
  print(a[i])
a[n-1] = (j + a[n-1]) // 2
print(a[n-1])
```


69) Здесь нужно построить цикл, в котором переменная изменяется с шагом 4. В Паскале придётся использовать цикл с условием (**while**). Переменная **i** обозначает номер первого элемента в четвёрке. Находим среднее арифметическое четвёрки и сохраняем в переменной **j**. Затем во вложенном цикле записываем это значение во все элементы четвёрки и сразу выводим на экран.

Полное решение на **Паскале**:

```
i := 1;
while i < N do begin
  j := (a[i] + a[i+1] + a[i+2] + a[i+3]) div 4;
  for k:=0 to 3 do begin
    a[i+k] := j;
    writeln(a[i+k]);
  end;
  i := i + 4; // переход к следующей четвёрке
end;
```

Решение на языке **Python**:

```
for i in range(0,n,4):
  j = (a[i] + a[i+1] + a[i+2] + a[i+3]) // 4
  for k in range(4):
    a[i+k] = j
    print(a[i+k])
```

70) Сначала находим среднее арифметическое, оно может быть вещественным числом, для его вычисления предназначена переменная **s**. Затем в цикле проверяем условие для каждого элемента массива: если модуль разности этого элемента и **s** больше, чем половина **s**, заменяем элемент на 0.

Полное решение на **Паскале**:

```
s := 0;
k := 0;
for i:=1 to n do
  if a[i] mod 2 = 0 then begin
    s := s + a[i];  // сумма чётных
    k := k + 1;     // количество чётных
  end;
s := s / k;         // среднее арифметическое чётных
for i:=1 to n do begin
  if abs(a[i]-s) > s/2 then
    a[i] := 0;
  writeln(a[i]);
end;
```

Решение на языке **Python**:

```
k = [s for s in a if s % 2 == 0]
```

```
s = sum(k) / len(k)
for i in range(n):
  if abs(a[i]-s) > s/2:
    a[i] = 0
  print(a[i])
```

71) Как следует из условия. нужно сначала найти максимальные из чётных и нечётных элементов. Для этого будем использовать переменные **k** и **s**.

Тут есть одна сложность – как искать максимум не из всех элементов, а только из тех, которые удовлетворяют какому-то условию. Стандартный приём поиска максимума с записью начального значения, равного первому элементу массива, не сработает, потому что этот первый элемент может не удовлетворять условию.

Но выход есть – сначала записать в переменную **k** (в ней будет максимальный элемент) какое-то значение, которое меньше всех возможных. Например, если мы ищём максимум из положительных чисел в диапазоне от 0 до 10000, запишем туда -1:

```
k := -1;
for i:=1 to N do
  if (a[i] mod 2 = 0) and (a[i] > k) then
    k := a[i];
```

Максимальный из нечётных ищется аналогично:

```
s := -1;
for i:=1 to N do
  if (a[i] mod 2 <> 0) and (a[i] > s) then
    s := a[i];
```

Конечно, два цикла можно объединить в один.

После вычисления **k** и **s** сравниваем их и во втором цикле выполняем обнуление чётных или нечётных элементов.

Полное решение на **Паскале**:

```
k := -1;
s := -1;
for i:=1 to N do begin
   if (a[i] mod 2 = 0) and (a[i] > k) then
       k := a[i];
   if (a[i] mod 2 <> 0) and (a[i] > s) then
       s := a[i];
end;
if k < s then
  for i:=1 to N do begin
    if a[i] mod 2 = 0 then
      a[i] := 0;
    writeln(a[i])
  end
else
  for i:=1 to N do begin
    if a[i] mod 2 = 1 then
      A[i] := 0;
    writeln(a[i]);
  end;
```

В конце программы не совсем красиво то, что в обеих ветвях условного оператора записаны два цикла, которые выполняют похожие действия. Можно исправить этот недочёт, например, так:

```
k := -1;
```



```
s := -1;
for i:=1 to N do begin
   if (a[i] mod 2 = 0) and (a[i] > k) then
       k := a[i];
   if (a[i] mod 2 <> 0) and (a[i] > s) then
       s := a[i];
end;
if k < s then
     j := 0
else j := 1;
for i:=1 to N do begin
  if a[i] mod 2 = j then
    a[i] := 0;
  writeln(a[i]);
end;
```

Однако следует помнить, что в этой задаче не требуется находить оптимальное решение, и оптимизация никак не будет оценена. Главное – написать правильно работающую программу. И лучше, если она будет понятной.

72) Задача в целом аналогична предыдущей задаче. В разборе задачи 71 можно посмотреть, как искать минимальный и максимальный из элементов, удовлетворяющих некоторому условию. Отметим, что достаточно найти только количество чётных элементов **j**, тогда количество нечётных определяется как **N–j**.

Полное решение на **Паскале**:

```
j := 0;       { счётчик чётных элементов }
k := -1;      { максимальный чётный элемент }
s := 10001;   { минимальный нечётный элемент }
for i:=1 to N do begin
   if a[i] mod 2 = 0 then begin
     j := j + 1;
     if a[i] > k then
       k := a[i];
   end
   else
     if a[i] < s then
       s := a[i];
end;
if j < N-j then begin
  for i:=1 to N do          { замена чётных элементов на k }
    if a[i] mod 2 = 0 then
      a[i] := k;
end
else
  for i:=1 to N do begin { замена нечётных элементов на s }
    if a[i] mod 2 = 1 then
      a[i] := s;
  end;
```

И оптимизированный вариант:

```
j := 0;
k := -1;
s := 10001;
for i:=1 to N do begin
   if a[i] mod 2 = 0 then begin
```

```
     j := j + 1;
     if a[i] > k then
       k := a[i];
   end
   else
     if a[i] < s then
       s := a[i];
end;
if j < N-j then
  j := 0
else begin
  j := 1;
  k := s;
end;
for i:=1 to N do begin
  if a[i] mod 2 = j then
    a[i] := k;
  writeln(a[i])
end
```

73) Эта задача решается так же, как и две предыдущих. Но есть одна особенность: при первом проходе нам нужно найти

- количество элементов, заканчивающихся на 3,
- минимальный элемент, заканчивающийся на 3;
- количество элементов, заканчивающихся на 5;
- максимальный элемент, заканчивающийся на 5.

Для этого нужно 4 переменных, а у нас в распоряжении – только 3. Но выход есть. нас ведь интересуют не отдельные количества двух групп элементов, а то, каких больше, то есть разность этих значений. Поэтому будем использовать переменную-счётчик `j` так: если нашли элемент, который заканчивается на 3, увеличиваем `j`; если нашли элемент, который заканчивается на 5, уменьшаем `j`.

Полное решение на **Паскале**:

```
j := 0;     { разность размеров двух групп }
k := 10001; { минимальный элемент, заканчивающийся на 3 }
s := -1;    { максимальный элемент, заканчивающийся на 5 }
for i:=1 to N do begin
   if a[i] mod 10 = 3 then begin
     j := j + 1;
     if a[i] < k then
       k := A[i];
   end;
   if A[i] mod 10 = 5 then begin
     j := j - 1;
     if A[i] > s then
       s := A[i];
   end;
end;
if j < 0 then begin
  for i:=1 to N do begin
    if A[i] mod 10 = 3 then
      A[i] := k;
    writeln(a[i]);
  end;
```



```
end
else
  for i:=1 to N do begin
    if A[i] mod 10 = 5 then
      A[i] := s;
    writeln(a[i]);
  end;
```

И оптимизированный вариант:

```
j := 0;
k := 10001;
s := -1;
for i:=1 to N do begin
   if A[i] mod 10 = 3 then begin
     j := j + 1;
     if A[i] < k then
       k := A[i];
   end;
   if A[i] mod 10 = 5 then begin
     j := j - 1;
     if A[i] > s then
       s := A[i];
   end;
end;
if j < 0 then
  j := 3
else begin
  j := 5;
  k := s
end;
for i:=1 to N do begin
  if A[i] mod 10 = j then
    A[i] := k;
  writeln(a[i])
end;
```