Задача А. Трудная задача из ЕГЭ

Есть несколько подходов к решению этой задачи. Первый из них – *моделирование*. Будем брать поочередно каждую цифру, проверять, не превосходит ли она 5, делить ее на 2 при необходимости, и выводить результат, если получится нечетное число.

Другой подход – заранее рассмотреть все возможности:

Исходная цифра	Что выводить
0	
1	1
2	
3	3
4	
5	5
6	3
7	3
8	
9	

Из таблицы видно, что
а) при вводе 1 или 5 нужно вывести ту же цифру;
б) при вводе 3, 7 или 8 нужно вывести 3;
в) в остальных случаях ничего выводить не нужно.

Перейдем к реализации этих идей. Здесь также возможно несколько способов.
Первый способ: считываем строку.
Заметим, что входные данные можно считать как строку:

```
s:string;
readln(s);
```

а затем обработать отдельно каждый символ:

```
for i:=1 to 4 do
  обработать s[i]
```

При реализации первого подхода к решению придется преобразовать символ в число:

```
  t := strtoint(s[i]);
  if t>5 then
    t := t div 2;
  if t mod 2=1 then
    write(t);
```

```
При втором подходе можно работать сразу с символами:
  if s[i] in ['1','5'] then
    write(s[i])
  else
    if s[i] in ['3','6','7'] then
      write(3)
```

Второй способ: считываем число.

```
n:integer;
readln(n);
```

В этом случае нам придется выделить из него отдельные цифры. Чтобы получить соответствующую цифру, нужно разделить число на такую степень десятки, чтобы данная цифра оказалась последней, и затем взять остаток при делении на 10. Например,

```
(8372 div 1000) mod 10 = 8
(8372 div 100) mod 10  = 3
(8372 div 10) mod 10   = 7
(8372 div 1) mod 10    = 2
```

Конечно, операцию взятия остатка в первом примере и деление на 1 в последнем примере можно опустить, но мы собираемся написать единый цикл для выделения каждой цифры:

```
b := 1000;
for i:=1 to 4 do begin
  c := (n div b) mod 10;
  обработка c
  b := b div 10; {b = 1000, 100, 10, 1}
```

Обработка цифры выполняется по тому же принципу, что и в решениях первым способом.

Задача B. Разложение на простые множители

Это задача относится к разделу, называемому *комбинаторика.* Мы приведем два решения: одно – чисто математическое, другое – алгоритмическое. Для обоих решений потребуется предварительная подготовка – нам нужно будет разложить число на множители. В “математическом” решении нам будет важно, в какой степени входит каждый простой сомножитель. Для “алгоритмического” решения потребуется полное разложение.

Для разложения числа *N* на простые множители будем делить его сначала на 2 (столько раз, сколько оно делится без остатка), затем на 3, и т.д., пока не получим единицу:

```
d:=2; {число, на которое будем делить}
while n>1 do
  if n mod d = 0 then
    n:=n div d
  else
    d:=d+1;
```

Это общий алгоритм, который нужно будет модифицировать для каждого из решений.

Первый способ: математический.

Заметим, что если бы у числа *N* было *K* простых делителей, и каждый входил бы в разложение в первой степени, то количество возможных записей было бы равно 1*2*…**K* = *K*! Действительно, при записи делителей мы на первое место могли бы поставить любой из *K* делителей, на второе – любой из (*K* – 1) оставшихся, и т. д., на последнее место – единственный не использованный ранее делитель.
Пусть теперь один из делителей *p* входит в разложение дважды. Представим себе на минуту, что эти два делителя не равны и выпишем *K*! разложений. После этого мы обнаружим, что все разложения разбиваются на пары, в которых наши два делителя стоят на одних и тех же местах, но в обратном порядке. Например:

$$12 = 2_1*2_2*3 = 2_2*2_1*3 = 2_1*3*2_1 = 2_2*3*2_1 = 3*2_1*2_2 = 3*2_2*2_1.$$

Поэтому, чтобы получить количество различных записей, нужно разделить *K*! на 2. Если бы некоторый делитель входил в разложение в 3-й степени, то каждая запись имела бы еще 5 “двойников” – ведь переставить местами 3 одинаковых делителя можно 3! = 6 способами. И вообще, если делитель входит в степени *m*, то делить придется на *m*!
Если же несколько делителей входят в разложение в степени выше 1, то для каждого из них нужно будет разделить *K*! на соответствующий факториал.
Например, $72 = 2^3 * 3^2$. Общее количество делителей равно 5, а искомое количество записей равно 5! / (3! * 2!), поскольку один из делителей входит в разложение в степени 3, а другой – в степени 2.

Превращение данной формулы в программу мы оставляем в качестве самостоятельного упражнения.

Второй способ: переборный.

Вместо того, чтобы вычислять количество записей, можно попытаться перебрать и пересчитать их все (благо, их не очень много – оцените наибольший возможный ответ в задаче самостоятельно!). Программа получится менее эффективной и более сложной, поэтому мы не будем приводить здесь подробное описание.

Задача C. У моего папы больше денег

В этой задаче требуется всего лишь вывести одно любое число, которое больше введенного. Но первое приходящее на ум решение

```
read(N);
write(N+1);
```

работать не будет: вводимое число может содержать до 100 знаков и не поместится ни в один из стандартных типов данных.

Впрочем, на языке python, который поддерживает длинные числа, такое решение можно записать даже в одну строчку:
`print(int(input())+1)`

Следующая приходящая на ум идея – напечатать число, в 10 раз большее данного. Для этого достаточно лишь приписать к вводимому числу в конце 0. А считать это число можно в строковую переменную:

```
var
  s : string;
begin
  read(s);
  write(s,0);
end.
```

Но оказывается, что есть решение еще проще (в некотором смысле)! Поскольку все числа в условии ограничены (состоят не более чем из 100 цифр), то можно независимо от вводимого числа вывести, например число 10…0 (100 нулей), используя один оператор write:

```
write('10000000000000000000000000000000000000000…0000000000');
```

или цикл `for` (тогда удобнее выводить число, состоящее из 101 единицы):

```
for i:=1 to 101 do
  write(1);
```

На языке python такие решения пишутся еще короче:
`print(10**100)`
или
`print('1'*101)`

Обратите внимание: в условии есть ограничение на число первого игрока, но нет ограничений на число второго, поэтому все наши решения корректные.

Задача D. Деление с остатком

Первый способ: перебор.

Возьмем какое-нибудь число, которое меньше, чем максимально возможное число в ответе, например, 48 000 (при делении его на 2, затем на 3 и на 4 получится 2 000, а, согласно ограничениям в условии, K не превосходит 1 000). Будем перебирать все возможные ответы (от 1 до 48 000) по возрастанию и выводить подходящие:

```
for i:=1 to 48 000 do
  if i div 2 div 3 div 4 = K then
    write(i,' ');
```

Это решение верное, но не самое эффективное.

Второй способ: вычисляем ответ.

Ясно, что Вася мог изначально взять одно из нескольких подряд идущих натуральных чисел. Поэтому достаточно определить минимальное и максимальное такое число.

Начнем с минимального. Ясно, что самое маленькое число должно каждый раз делиться без остатка, поэтому это число 2*3*4*K=24K.

Вместо того, чтобы искать максимальное число, найдем следующее за ним число, то есть минимальное число, которое в результате проделанных операций даст уже ответ K+1. Аналогично предыдущему рассуждению это число 24(K+1). Поэтому Вася мог взять любое число от 24K до 24(K+1)-1 = 24K+23. Осталось написать программу, печатающую эти числа через пробел:

```
for i:=24*K to 24*K+23 do
  write(i,' ');
```

Задача E. Считалочка

Составим план решения.

1) Подсчитаем количество слов в считалочке.
2) Подсчитаем количество людей.
3) Выясним номер человека, которому выпадет водить.
4) Найдем в списке имен нужное по номеру.

Теперь перейдем к реализации этого плана.

1) Заметим, что количество слов на 1 больше, чем количество пробелов. Поэтому достаточно пройти по строке и подсчитать количество пробелов:

```
spaces:=0;
for i:=1 to length(s1) do
  if s1[i]=' ' then
    inc(spaces);
```

2) Аналогично подсчитаем количество слов *N* во второй строке:
3) Заметим, что номер человека, на котором заканчивается считалочка, есть остаток от деления количества слов в считалочке на количество людей (если остаток равен 0, то номер человека равен *N*).

```
number := (spaces + 1) mod N;
if number = 0 then
  number := N;
```

Заметим, что если пронумеровать людей с нуля, то формулу можно записать проще:

```
number := spaces mod N;
```

Далее мы будем предполагать, что нумерация именно такая.

4) Осталось найти нужное имя (оно будет следовать сразу после пробела с номером `number`):

```
i := 1; k := 0;
while k<number do begin
  if s2[i] = ' ' then
    inc(k);
  inc(i);
end;
inc(i);
```

и вывести его:

```
while (i<=length(s2)) and (a[i]<>' ') do
  write(s2[i]);
  inc(i);
```

А вот полное решение этой задачи на языке python:

```
spaces, people = int(input().count(' ')), input().split()
print (people[spaces % len(people)])
```

Задача F. Кольцевая

Заметим, что если бы автомобили двигались по прямой, то они удалялись бы друг от друга со скоростью v_1+v_2. За время T они оказались бы на расстоянии $(v_1+v_2)T$. Поскольку они двигаются по кругу длины L, то чтобы найти расстояние между ними (по одной из дуг окружности), достаточно взять остаток $r = ((v_1+v_2)T) \bmod L$. Тогда длина второй дуги окружности между ними будет равна $L - r$. Осталось вывести минимальное из чисел r и $L - r$.

Задача G. Хорошие стихи

Рассмотрим две строки s и t и научимся определять, сколько последних символов в них совпадают. Будем одновременно идти по этим строкам с конца до тех пор, пока одна из них не закончится или пока мы не встретим несовпадающие символы:

```
i := 0;
while (i < length(s)) and (i < length(t))
                and (s[length(s) - i] = t[length(t) - i]) do
  i := i+1;
```

После выполнения этого фрагмента программы переменная i будет содержать количество равных символов в конце строк s и t. Эту процедуру нужно проделать для первой и третьей строк стихотворения, затем для второй и четвертой, и из двух полученных чисел выбрать большее.

Задача H. Прыжки с трамплина

Решение задачи состоит из двух этапов. Сначала нужно найти минимальную и максимальную оценки и подсчитать сумму без них, а затем вывести результат в указанном формате.

Начнем с цикла, который будет искать минимум, максимум и подсчитывать суммы всех оценок.

```
max := 1; {номер максимальной оценки}
min := 1; {номер минимальной оценки}
sum := a[1];

for i := 2 to 5 do begin
  if a[i] < a[min] then
    min := i;
  if a[i] >= a[max] then
    max := i;
  sum := sum + a[i];

end;
```

Обратите внимание на знаки < и >= при поиске максимума и минимума. Нам нужно найти самый левый минимум, поэтому минимумы, равные предыдущему, нас не интересуют. Что касается максимума, то здесь нам нужно самое правое число, поэтому если мы встречаем равное число, мы обновляем максимум.

После того, как искомые оценки найдены, осталось напечатать оценки:

```
for i := 1 to 5 do
  if (i = min) or (i = max) then
    write('(', a[i], ')', ' ')
  else
    write(a[i], ' ');
```

и сумму:

```
write('= ', sum - a[min] - a[max]);
```

Задача I. Магический квадрат

В этой задаче требуется аккуратно реализовать описанный в условии алгоритм. Будем по очереди расставлять числа от 1 до N^2 в клетки с координатами (x,y) двумерного массива a (изначально по условию x:=(n+1) div 2; y:=1; массив заполнен нулями).

```
for i:=1 to N*N do
  a[x][y]:=i;
  {Затем запомним эти координаты}
  oldx:=x;
  oldy:=y;
  {и вычислим координаты следующей клетки}
  y:=y-1;
  x:=x+1;
  {Проверим, не вышли ли мы за пределы квадрата:}
  if x=n+1 then x:=1;
  if y=0 then y:=n;
  {После этого проверим, не были ли мы в этой клетке ранее:}
  if a[x][y]<>0 then
     {(спускаемся на одну клетку вниз от начальной: здесь нам и пригодились
запомненные координаты) }
     x:=oldx;
     y:=oldy+1;
     {и снова проверяем, не вышли ли мы за границы квадрата в этом случае:}
     if y=n+1 then y:=1;
```