Problem A. Игра с многоугольником

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

Пусть среди образовавшихся после хода частей есть хотя бы одна часть, не являющаяся треугольником. Тогда в ней можно провести хотя бы одну диагональ, то есть на этом ходу игра не будет закончена. Таким образом, ответ — это количество непересекающихся диагоналей, необходимых для того, чтобы разбить многоугольник на треугольники.

Докажем индукцией по N, что ответ равен $N - 3$. Для $N = 3$ утверждение тривиально — треугольник уже есть, ничего проводить не надо. Пусть утверждение верно для всех $K < N$. Рассмотрим N-угольник. Первая проведённая диагональ делит его на два многоугольника. Так как проведённую диагональ пересекать нельзя, то все действия будут происходить либо в одном, либо в другом многоугольнике. Пусть в одном многоугольнике L вершин, тогда в другом будет $N + 2 - L$ вершин (так как вершины, соединённые диагональю, принадлежат обоим многоугольникам). Оба числа меньше, чем N, а раз так, то для них мы можем подсчитать ответ: $L - 3$ и $N - L - 1$. Одна диагональ уже проведена, таким образом, общее количество диагоналей будет равно $L - 3 + n - L - 1 + 1 = N - 3$. Утверждение доказано.

Таким образом, программа должна считывать N и выводить число $N - 3$.

Пример решения, реализованного на языке C:

```
#include <stdio.h>

main()
{
    int n;
    scanf("%d", &n);
    printf("%d\n", n - 3);
    return 0;
}
```

Problem B. Машина «Сетунь»

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 секунда
Memory limit:	512 мебибайт

Максимальным 14-тритовым числом будет число, во всех 14 разрядах которого будут единицы (действительно, если в каком-то разряде стоит 0 или -1, то при замене только этого разряда на единицу число увеличится, что противоречит свойству максимальности). То есть надо найти сумму $3^{13} + 3^{12} + 3^{11} + \ldots + 3 + 1$.

Это можно сделать как написав программу на компьютере, так и с помощью калькулятора, особенно, если заметить, что по формуле суммы геометрической прогрессии ответ равен $(3^{14} - 1)/2$, или, после всех вычислений, 2 391 484.

Problem C. Вставка скобок

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Самый простой подход к решению этой задачи: заметим, что если вставлять сразу перед закрывающей скобкой открывающую скобку того же типа, а сразу после открывающей скобки — закрывающую скобку того же типа (то есть каждую скобку во входной строке заменять парой из открывающей и закрывающей скобки этого же типа), то получившееся выражение очевидно будет правильным, а его длина будет ровно вдвое больше длины исходной последовательности.

Пример решения, реализованного на языке C:

```
#include <stdio.h>
#include <string.h>

int main()
{
 int i;
 char a[10010];
 scanf ("%s",a);
 for (i=0;i<strlen(a);i++)
  {
   if ( (a[i]=='{') || (a[i]=='}') ) printf ("{}");
   if ( (a[i]=='(') || (a[i]==')') ) printf ("()");
   if ( (a[i]=='[') || (a[i]==']') ) printf ("[]");
  }
 printf ("\n");
 return 0;
}
```

Problem D. Странный прибор

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 секунда
Memory limit:	512 мебибайт

Заметим, что в словах “кок” и “банан” все буквы, кроме одной, разбиваются на пары, а ответом служит номер в алфавите оставшейся буквы, а в слове “мама” все буквы разбиваются на пары и ответ равен 0.

Таким образом, действия идут с номерами букв в алфавите. Из известных побитовых операций таким свойством обладает “исключающее ИЛИ”, или XOR. Проверив этот вариант на остальных словах (например, на коротком слове “лёд”) убеждаемся, что ответ — это XOR номеров всех букв слова в алфавите.

Поэтому осталось закодировать слово “программирование” в соответствии с обнаруженной закономерностью.

Получаем число 13.

Problem E. Winux

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Представим список задач в виде окружности с N равномерно расположенными на ней точками, экземпляры задачи `solve` — красными точками, другие задачи — белыми. Ротация задач в этой модели соответствует вращению окружности на одно деление (например, по часовой стрелке).

Очевидно, что так как список различался, то, во-первых, существует как минимум одна белая точка, во-вторых, окружность сделала нецелое число оборотов (иначе бы список совпадал — при полном количестве оборотов все точки перейдут сами в себя). Очевидно, что «дробная часть» также переведёт красные точки друг в друга (так как полный оборот является, по сути, тождественной операцией, то количество полных оборотов можно не учитывать).

Покажем, что возможность повторения конфигурации красных точек при не целом количестве оборотов существует тогда и только тогда, когда N — составное.

Рассмотрим первую с начала вращения ситуацию, когда красные точки повторились; обозначим количество делений, на которое повернули окружность, за k. Докажем, что все ситуации, при которых красные точки повторяются, будут получены за nk делений, где n — целое число. Пусть это не так, тогда мы получили какую-то ситуацию при повороте на $n_1k + l$ делений, где $1 \leq l < k$. Но после каждых k поворотов красные точки встают на те же самые места, то есть получается, что после n_1k поворотов точки будут на том же самом месте, и получается, что после l поворотов красные точки окажутся снова на своих местах. Но так как $1 \leq l < k$, это противоречит выбору k как повороту на наименьшее количество делений, переводящему красные точки друг в друга. Противоречие.

А раз так, то в случае, если поворот на $k < N$ делений переводит красные точки друг в друга, то (так как поворот на N делений является полным поворотом и также переводит красные точки в красные точки), то N делится на k. При этом $k > 1$ (иначе бы все точки были красными — при повороте на 1 каждая точка переходит в соседнюю, и сохранение красных точек обозначало бы, что вся окружность красная, что противоречит доказанному ранее существованию белой точки), то есть N делится на меньшее его число, не являющееся единицей, и, следовательно, является составным.

В обратную сторону: если N является составным, оно делится на k ($1 < k < n$). Тогда берём каждую k-ю точку в качестве «красной». При повороте на k делений красные точки перейдут сами в себя, то есть пример построен.

Поэтому для того, чтобы повторение естественным образом было невозможно (то есть являлось бы признаком вмешательства), N должно быть простым. Так что задача сводится к тому, чтобы проверить, что введённое число N – простое.

Пример решения, реализованного на языке C:

```
#include <stdio.h>

int main()
{
 int i,N;
 scanf ("%d",&N);
 if (N==2)
  {
   printf ("Danger\n");
   return 0;
  }
  else if (N%2==0)
   {
    printf ("Safe\n");
    return 0;
   }
   else
    {
     for (i=3;i*i<=N;i++)
       if (N%i==0)
       {
        printf ("Safe\n");
        return 0;
       }
     printf ("Danger\n");
     return 0;
    }
}
```