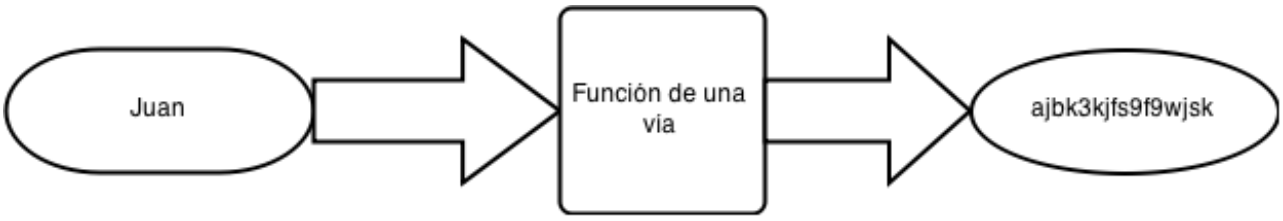
Sales

En la ciencia de la computación, las funciones *una-vía* son aquellas que son fáciles de computar para cada posible entrada, pero a la vez es bien difícil invertir la imagen de cualquier entrada aleatoria. En contextos aplicados, los términos "fácil" y "difícil" se interpretan usualmente de forma relativa a alguna entidad computacional específica; típicamente "usando pocos recursos computacionales para los usuarios legítimos" y "excesivamente cargado para algún agente malicioso". Fíjate que esto tiene mucho sentido para la criptografía. Hoy en día, la data que debe ser confidencial, se desplaza por la Internet de forma encriptada. Para que esto sea factible, debe ser relativamente rápido poder encriptar algún pedazo de información, pero también a la vez increíblemente difícil para algún usuario malicioso.



Una función hash es una función *una-vía* (fíjate en la figura). Una función hash recibiría la cadena de caracteres "Juan" y produciría "ajbk3kjfs9f9wjsk". Esta última cadena puede navegar por la Internet libremente, y algún usuario malicioso no tendría idea de qué se está hablando.

Hay muchos algoritmos de "hashing" conocidos, con implementaciones gratuitamente disponibles en distintos lenguajes de programación. Ejemplos de estos tenemos: MD5, SHA-1, SHA-2 y SHA-3. Estos algoritmos, similar a la figura de arriba, reciben una cadena de caracteres y, con relativa facilidad, devuelven otra cadena "hashada". Si tu amigo en su computadora, por ejemplo, escribe md5("Juan"), y tú, en la tuya, escribes md5("Juan"), ambos conseguirían la misma cadena de caracteres como resultado.

¿Te ha llegado por la cabeza que los usuarios maliciosos son genuinamente curiosos? Lo son. Es por ello que los hash no son normalmente mecanismos extremadamente seguros para almacenar información confidencial. Un usuario malicioso pudiera, teóricamente, tener un enorme diccionario de los hash de cada posible cadena de caracteres. Así este **si** pudiera saber de qué estás hablando: sólo tiene que buscar cada hash en su diccionario teórico. Que gran problema, ¡no hay privacidad!

Sin embargo, hay mecanismos que ayudan a prevenir que tales cosas sucedan. Uno de ellos es el uso de Sales Criptográficas. Una sal es una data aleatoria que se utiliza como entrada adicional a las funciones *una-vía* para prevenir este tipo de ataques basados en diccionarios. El uso es bastante sencillo. En vez de llamar md5("Juan"), lo que haremos es concatenar alguna otra cadena de caracteres aleatoria a "Juan". Por ejemplo, "abc992" + "Juan" = "abc992Juan". Entonces, cuando vayamos a llamar a la función md5(), lo hacemos con esta nueva cadena. ¡Ahora si que los usuarios maliciosos pasarán trabajo!

Un uso similar de las sales es en los sistemas de archivos en las nubes. Imagina que Juan y tú suben ambos dos archivos distintos, pero con el mismo nombre, a una misma carpeta. ¿Qué sucedería? Dependiendo del sistema de archivos, la operación que se ejecute de último pudiera bien fallar, o renombrar el último archivo, concatenándole un feo "(1)" en algún lugar.

Una solución más elegante es añadirle una sal a cualquier archivo que se suba a una carpeta. Por ejemplo, imaginemos que se suban estos tres archivos al mismo tiempo: "A.txt", "B.txt", "A.txt". Para evitar el feo renombrado, o el vergonzoso fallo, un sistema de archivos inteligente produciría lo siguiente como resultado: "92839842_A.txt", "9282482942_B.txt", "92828111_A.txt".

El sistema ODI (Ortografía Digital Inteligente) utiliza la siguiente nomenclatura para nombres salados:

1. Todas las sales serán **exactamente** de 20 caracteres.
2. Las sales estarán formadas **exclusivamente** por letras y dígitos.
3. El nombre de un archivo salado se establecerá tras concatenar tres cadenas: una sal debidamente formada, un guión bajo ('_') y el nombre original del archivo.

Juan tiene una lista de nombres de archivos que extrajo de una carpeta. En esta carpeta residen tantos archivos salados como archivos no salados. Escribe un programa que imprima el nombre original de cada uno de los archivos.

Entrada

La entrada de este problema cuenta de múltiples líneas. En la primera línea habrá un único número entero N ($1 \leq N \leq 10$), la cantidad de archivos presentes al momento en que Juan revisó la carpeta. Siguen N líneas. La i-ésima línea contendrá el i-ésimo nombre de archivo. Se garantiza que ningún archivo tendrá espacios en su nombre. El alfabeto de los nombres de archivos es el siguiente: letras mayúsculas y minúsculas, dígitos, puntos y guiones bajos ('_').

Salida

Imprime los nombres originales de cada uno de estos archivos, uno por cada línea, en el mismo orden en que se leyeron de la entrada.

Ejemplos

Ejemplo #1

Entrada
4 jfjr959493ajsks999s9_A.txt sieeke4444abcksl4krk_A.cpp a8b95b8k4ra9f4ka92kf_A.cs A.java
Salida
A.txt A.cpp A.cs A.java