**Caesar**

[**tl;dr**](http://docs.cs50.net/problems/caesar/caesar.html#tl-dr)

Implemente un programa que encripte mensajes usando Caesar’s cipher, como abajo.

$ ./caesar 13

plaintext: HELLO

ciphertext: URYYB

[**Background**](http://docs.cs50.net/problems/caesar/caesar.html#background)

Supuestamente, Caesar (sí, ese Caesar) usó para "encriptar" (encubrir en una forma reversible) mensajes confidenciales corriendo cada letra del mismo un número determinado de lugares. Por ejemplo, el talvez escribía A como B, B como C, C como D, …​, y, dándole la vuelta alfabéticamente, Z como A. Y así, para decir HELLO a alguien, Caesar podría escribir IFMMP. Una vez habiéndo recibido tal mensaje del Caesar, el receptor tendría que "desencriptarlo" corriendo las letras en la dirección opuesta el mismo número de lugares.

La secretividad de este "cryptosistema" descansaba en que solamente Caesar y los recipientes conocían un secreto, el número de lugares que Caesar había corrido sus letras (e.g., 1). ¡Algo no particularmente seguro para los estándares modernos, pero, hey, si tú eres quizás el primero en el mundo en hacerlo, es muy seguro!

El texto no encriptado es generalmente llamado *plaintext*. El texto encriptado es generalmente llamado *ciphertext*. Y el secreto utilizado es llamado una *key o llave*.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tabla 1. Encrypting HELLO con una llave de 1 nos da IFMMP. | | | | | |
| **plaintext** | H | E | L | L | O |
| **+ key** | 1 | 1 | 1 | 1 | 1 |
| **= ciphertext** | I | F | M | M | P |

Más generalmente, el algoritmo Caesar’s (i.e., cipher) encripta mensajes "rotando" cada letra por *k* posiciones. Más formalmente, si *p* es un plaintext (i.e., un mensaje no encriptado), *pi* es el *iavo* caracter en *p*, y *k* es una llave secreta (i.e., un número entero no negativo), entonces cada letra, *ci*, en el ciphertext, *c*, es computado como

*ci*=(*pi*+*k*)mod26

en donde mod26 aquí significa el “residuo” de dividir por 26." Esta fórmula quizás haga que cipher se vea más complicado de lo que es, realmente es solo una forma concisa de expresar el algoritmo precisamente.

[**Especificación**](http://docs.cs50.net/problems/caesar/caesar.html#specification)

Diseñe e implemente un programa, caesar, que encripte mensajes utilizando Caesar’s cipher.

* Implemente su programa en un file llamado caesar.c en un directorio llamado caesar.
* Su programa debe aceptar un solo argumento de linea de commando, un entero no negativo. Vamos a llamarlo *k* por sanidad de la discusión.
* Si su programa es ejecutado sin ningún argumento de linea de comandos o con más de un argumento de linea de comandos, su programa deberá imprimir un mensaje de error de su elección (con printf) y retornar de main un valor de 1 (lo que tiende a significar un error) inmediatamente.
* Usted puede asumir que, si un usuario provee una línea de argumentos de línea de comandos, este será un entero no negativo (ej. 1). No se necesita comprobar que en efecto sea numerico.
* No asuma que *k* será menor o igual que 26. Su programa deberá trabajar con todos los valores integrales no negativos de *k* menores que 231 - 26. En otras palabras, usted no necesita preocuparse por si su programa eventualmente colapsa si el usuario escoje un valor para *k* que es muy grande o al menos muy grande para caber en un int. (Recuerde que un int puede desbordarse.) Pero, aún si *k* es mayor que 26, los caracteres alfabéticos en la entrada de su programa input deberán mantenerse como caracteres alfabéticos en la salida de su programa. Por ejemplo, si *k* es 27, A no debería convertirse en [ aún si [ está a 27 posiciones de A en ASCII, por [asciichart.com](http://www.asciichart.com/); A debería convertirse en B, ya que B está a 27 posiciones de A, al dar la vuelta de Z a A.
* Su programa deberá mostrar de salida texto plano: (sin una nueva linea) y luego solicitar al usuario una string de texto plano (usando get\_string).
* Su programa deberá mostar como salida ciphertext: (sin un salto de linea) seguido de el texto plano que correspoonda al ciphertext, con cada character alfabético en el texto plano "rotado" *k* posiciones; los caracteres no alfabéticos deberán mostrarse en la salida sin ningún cambio.
* Su programa deberá preserver las letras: letras mayúsculas, aunque rotadas , deberán permanecer como letras mayúsculas; las letras minúsculass , aunque rotadas deberán permanecer como letras minúsculas.
* Después de d poner como salida el ciphertext, se deberá incluir una nueva línea , su programa deberá finalzar entonces retornando 0 de main.

[**Walkthrough**](http://docs.cs50.net/problems/caesar/caesar.html#walkthrough)

[**Us**](http://docs.cs50.net/problems/caesar/caesar.html#usage)**o**

Su programa deberá comportarse como el ejemplo de abajo. Asumiendo que el texto subrayado es lo que el usuario ha introducido.

$ ./caesar 1

plaintext: HELLO

ciphertext: IFMMP

$ ./caesar 13

plaintext: hello, world

ciphertext: uryyb, jbeyq

$ ./caesar 13

plaintext: be sure to drink your Ovaltine

ciphertext: or fher gb qevax lbhe Binygvar

$ ./caesar

Usage: ./caesar k

$ ./caesar 1 2 3 4 5

Usage: ./caesar k

[**Testing**](http://docs.cs50.net/problems/caesar/caesar.html#testing)

[**check50**](http://docs.cs50.net/problems/caesar/caesar.html#code-check50-code)

check50 cs50/2017/x/caesar

[**Staff’s Solution**](http://docs.cs50.net/problems/caesar/caesar.html#staff-s-solution)

~cs50/pset2/caesar

[**Consejos**](http://docs.cs50.net/problems/caesar/caesar.html#hints)

Este programa necesita aceptar un argumento de linea de comando, *k*, así que usted querrá declarar main con:

int main(int argc, string argv[])

Recuerde que argv es un "array" de strings. Usted puede pensar en un array como una fila de lockers de un gimnasio, dentro de los cuales hay ciertos valores (y quizás algunos calcetines). En este caso, dentro de cada uno de esos lockers hay un string. Para abrir (i.e., "index into") el primer locker, utilice una sintáxis como argv[0], ya que los arrays son "indexados a cero." Para abrir el próximo locker, utilice la sintáxis como argv[1]. Y así sucesivamente. Porsupuesto, si hay n lockers, usted deberá dejar de abrir lockers una vez que llegue a argv[n - 1], ya que argv[n] no existe! (Eso pertenece a otra cosa en cuyo caso usteed no debería abrirlo.)

Y así usted podría acceder *k* con código como

string k = argv[1];

asumiéndo que está realmente ahí! Recueder que argc es un int que iguala el número de strings que hay en argv, así que usted deberá revisar el valor de argc antes de abrir un locker que quizás no exista! Idealmente, argc será 2. porqué? bien, recuerde que todo dentro de argv[0], por defecto es, el nombre del programa, asi que argc siempre será al menos 1. Pero para este programa ustede desea que el usuario provea un argumento de línea de comando, k, en cuyo caso argc deberá ser 2. Por supuesto, si el usuario prove mas de un argumento de linea de comando en el prompt, argc podría ser mayor que 2, en cuyo caso, de nuevo, su programa deberá imprimir un error y devolver 1.

Ahora, solo porque el usuario introduce un entero en el prompt, eso no significa que su entrada será automáticamente almacenada en un int. Al contrario, será almacenada como un string que resultará verse como un int! Y por ello usted deberá convertir esa string en un int. Con suerte tendrán, una función, [atoi](https://reference.cs50.net/stdlib/atoi), que existe para ese propósito exactamente. He aquí como deberá usarla:

int k = atoi(argv[1]);

Note, que esta vez, hemos declarado k como un int asi que usted podrá hacer alguna aritmética con ello.

Ya que atoi es declarado en stdlib.h, usted querrá #include ese archive de cabecera en su propio código. (Tecnicamente, su código compilará sin él ahí, ya que notros también ya la hemos #include en cs50.h. Pero mejor no confiar en otra librería para #include archivos de cabecera que necesitará.)

Okay, así que una vez tenga almacenado k como un int, necesitará solicitar al usuario un texto plano. Es posible que get\_string e CS50’s pueda ayudarle con eso.

Una vez que tenga ambos k y un texto plano, p, es tiempo de encriptar el último con el primero. Recuerde que puede iterar sobre los caracteres de una string, imprimiendo cada una de ellas a la vez , con código como el siguiente:

for (int i = 0, n = strlen(p); i < n; i++)

{

printf("%c", p[i]);

}

En otras palabras, al igual que argv es un array de strings, también una string es un arreglo de chars. Por eso se puede usar corchetes para accesar caracteres individuales en una strings justo como sttrings individuales en argv. Elegante, eh? Por supuesto, imprimiéndo cada uno de los caracteres en una string uno a la vez no es exactamente criptografía. Bueno, quizás técnicamente si *k* es 0. Pero lo siguiente debería ayudarle a ayudar a Caesar a implementar su cipher!

Incidentalmente, necesitará #include otro archivo de cabecera para poder utilizer [strlen](https://reference.cs50.net/string/strlen).

Además de atoi, usted quizás encuentre algunas funciones muy útiles documentadas (en inglés) en [reference.cs50.net](https://reference.cs50.net/) bajo **ctype.h** y **stdlib.h**. Por ejemplo, isalpha resulte ser muy útil cuando se itera sobre caracteres de texto plano.

Y con respecto a dar la Vuelta de Z to A (o z a a), no se olvide de %, el operador modulo de C. Quizás también quiera revisar <http://asciitable.com/>, que revela los códigos ASCII para más que solo los caracteres alfabéticos , solo en caso de que resulte imprimiendo algunos caracteres accidentalmente.

[**FAQs**](http://docs.cs50.net/problems/caesar/caesar.html#faqs)

*Ninguna hasta ahora! Recargue esta página periódicamente para revisar si alguna surge!*