EJERCICIOS DE ARRAYS

Problema 1: El Canizador

Implementa un programa que convierta un texto en castellano estándar a su equivalente "cani". El *lenguaje cani* se distingue básicamente por tres elementos característicos (simplificando):

- La sustitución de las "c" por "k": *kabeza*, *korte*, …
- La alternancia de minúsculas y mayúsculas: *uN RoLlO KoMo eStE*. Por ejemplo puedes hacer que los caracteres pares sean minúsculas y los impares mayúsculas
- La adición de una "h" al final de cada palabra acabada en vocal: *esoh*, *tioh*,… Esto no se aplica a las palabras terminadas en consonante.

```
Introduce un texto: como lo flipas tio
Versión cani: kOmOh Loh fLiPaS TiOh
```

El texto original se almacenará en un array de `char` y la versión "cani" en otro distinto.

Es importante que modularices tu código, de modo que la transformación del texto se haga en un módulo aparte que llamas desde el programa principal. Puedes implementar cada característica del "cani" en un módulo distinto y un módulo adicional que llame sucesivamente a los anteriores.

Para facilitar el proceso es aconsejable que te ocupes solo de una característica a la vez. Es decir, implementa primero solo la sustitución de las "c" por "k", comprueba que funciona y luego encárgate de las otras dos, una cada vez. Ten en cuenta que la parte más difícil de implementar es añadir las "h" al final de las palabras acabadas en vocal.

Para convertir una letra a mayúsculas puedes usar la función `toupper(letra)`, a la que se le pasa un `char` con la letra a cambiar, y que devuelve la misma letra en mayúsculas. Para convertir a minúsculas usa `tolower(letra)`. Para usar cualquiera de ellas necesitarás un `#include <cctype>` en tu programa.

Gracias a Aitor Medrano (que no es cani) por la idea original de este ejercicio

Problema 2: Comprobando anagramas

Un anagrama es una palabra o frase que se construye con las mismas letras que otra, por ejemplo frase y fresa. Nos han encargado un programa que se ocupe de comprobar si una frase es un anagrama de otra. Para facilitar el proceso no se hace distinción entre mayúsculas y minúsculas, no se usarán acentos y no se tendrán en cuenta los espacios.

Entrada

Dos frases para analizar, cada una tendrá como máximo 100 caracteres.

Salida

La palabra SI caso de que la primera frase sea un anagrama de la segunda y NO en caso contrario

Casos de prueba

Entrada	**Salida**
`Jose de San Martin` `*No te rindes jamás*`	SI
`Tom Marvolo Riddle` `I am Lord Voldemort`	SI
`Istmo de Panama` `Tío Sam me da pan`	SI
`Tom Marvolo Riddle` `I am Lord Volbemort`	NO

Problema 3: El juego del ahorcado

Implementar en C el juego del ahorcado. En el programa guardaremos una lista de posibles palabras (con todas sus letras en mayúsculas). Al comenzar la partida se seleccionará una palabra aleatoriamente, y se mostrará la palabra enmascarada con caracteres de subrayado ('_'). Por ejemplo, si la palabra seleccionada es "PROGRAMA" aparecerán 8 caracteres de subrayado:

```
_ _ _ _ _ _ _ _
```

Tras esto se solicitará al usuario que introduzca una letra que piense que pertenece a la palabra. Si la letra pertenece, ésta quedará desenmascarada:

```
Quedan 10 intentos. Introduzca una letra: R
_ R _ _ R _ _ _
```

El juego terminará cuando la palabra completa quede desenmascarada, o cuando se agote el número máximo de intentos.

Pistas:

- El tamaño máximo de las cadenas será de 20 caracteres, tendremos 5 posibles palabras a seleccionar en la lista, y el número máximo de intentos será 10.
- Podemos crear una lista de cadenas con todas las posibles palabras que puedan aparecer en el juego de la siguiente forma:

```
TPalabra palabras[NUM_PALABRAS] = { "PROGRAMA", "LENGUAJE",
"OBJETIVO", "PRACTICA","NUMERO" };
```

- Para seleccionar una palabra de la lista de forma aleatoria al iniciar el juego puedes generar números.

- Puedes crear una variable de tipo cadena que almacene la cadena enmascarada actual (se inicializará con tantos caracteres de subrayado como caracteres tenga la palabra seleccionada).
- Cuando el usuario introduzca un carácter deberemos comprobar que sea alfanumérico, y de no ser así pediremos un nuevo carácter.
- Si el carácter introducido está en minúsculas, deberemos transformarlo a mayúsculas.
- Puedes utilizar las siguientes funciones de C++ para el manejo de caracteres:

`bool isalpha(char )` devuelve true si el carácter es una letra minúscula ('a'..'z') o mayúscula ('A'..'Z'); devuelve false en otro caso

`bool islower(char )` devuelve true si el carácter es una letra minúscula ('a'..'z'); devuelve false en otro caso

`char toupper(char )` devuelve en mayúscula el carácter pasado como parámetro

- Se comprobarán las coincidencias del carácter introducido dentro de la palabra seleccionada, y se escribirá este carácter en las posiciones correspondientes de la cadena enmascarada.
- Tras cada jugada se puede comprobar con strcmp() si la palabra enmascarada coincide con la original. Si fuese así, el jugador habría ganado la partida.

Ejemplo de ejecución:

```
_ _ _ _ _ _ _ _
Quedan 10 intentos. Introduzca una letra: R
_ R _ _ R _ _ _
Quedan 10 intentos. Introduzca una letra: E
_ R _ _ R _ _ _
Quedan 9 intentos. Introduzca una letra: a
_ R _ _ R A _ A
Quedan 9 intentos. Introduzca una letra: 2
Error: Debe introducir una letra
Quedan 9 intentos. Introduzca una letra: p
P R _ _ R A _ A
Quedan 9 intentos. Introduzca una letra: O
P R O _ R A _ A
Quedan 9 intentos. Introduzca una letra: m
P R O _ R A M A
Quedan 9 intentos. Introduzca una letra: G
P R O G R A M A
¡Enhorabuena! Ha acertado la palabra
```

Problema 4: Escapa del laberinto

Implementa un juego en el que el jugador se mueve en un mapa bidimensional formado por una cuadrícula de habitaciones (mazmorras). Cada una de ellas puede contener un

monstruo, un tesoro, la salida, o bien nada. El objetivo es escapar por la salida sin ser devorado por ningún monstruo. Las reglas son las siguientes:

- El jugador siempre partirá de la esquina superior izquierda (fila 0, columna 0)
- La casilla con la salida de las mazmorras se fijará de forma aleatoria, pero no puede ser la (0,0)
- El tamaño del mundo (filas y columnas) lo elige el jugador al principio de la partida
- Los monstruos y tesoros se repartirán de forma aleatoria. En cada casilla debe haber un 10% de probabilidades de que haya un monstruo y un 20% de que haya un tesoro.
- En el modo de juego normal, el mundo no será visible para el jugador. Únicamente se le informará del contenido de la habitación actual cuando se mueva a ella. Pero para poder probar y depurar el juego debería haber también un modo en el que el mapa sea visible.
- Los movimientos se introducirán como letras: n-norte, s-sur, e-este, o-oeste. Hay que controlar los movimientos "imposibles" cuando estamos en el borde del mapa (por ejemplo no se puede ir al norte estando en la primera fila, o al este estando en la última columna)
- Cuando se llega a una habitación con un monstruo, este devora al jugador y el juego termina inmediatamente.
- Cuando se llega a un tesoro, el jugador se lo lleva, incrementando el número de tesoros que tiene. La habitación pasará a estar vacía.
- Si el jugador alcanza la salida, el juego acaba, imprimiendo el número de tesoros que ha conseguido.

```
¿Cuántas filas quieres? 3
¿Cuántas columnas quieres? 3
¿Quieres que el mundo sea visible? (s/n) s
(J=Jugador, M=Monstruo, T=Tesoro, S=Salida)
 J  .  S
 M  .  .
 .  T  M
Introduce el movimiento (n/s/e/o): n
¡Ouch! ¡Te has dado contra una pared!. Prueba otra dirección
Introduce el movimiento (n/s/e/o): s
Un monstruo te devora. Todo ha terminado
```

(otra ejecución)

```
¿Cuántas filas quieres? 3
¿Cuántas columnas quieres? 3
```

```
 J  T  .
 S  .  .
 .  T  .
Introduce el movimiento (n/s/e/o): e
¡¡Encuentras un tesoro!!
 .  J  .
 S  .  .
 .  T  .
Introduce el movimiento (n/s/e/o): s
Estás en una mazmorra fría y oscura...
 .  .  .
 S  J  .
 .  T  .
Introduce el movimiento (n/s/e/o): o
 .  .  .
 J  .  .
 .  T  .
¡Has salido con vida de las mazmorras!. Te llevas 1 tesoros!
```

Pistas de implementación:

Puedes **representar el mundo** con un array bidimensional de enteros, codificando los posibles contenidos de una habitación con diferentes valores. Por ejemplo 0 podría significar vacío, 1 un monstruo, 2 un tesoro…. También puedes usar un array de enum (lo que internamente viene a ser lo mismo)

Rellenar con monstruos/tesoros: debes ir recorriendo todas las filas y columnas. Para ver si la casilla actual debería contener un monstruo puedes generar un número al azar entre 0 y 100 (o sea, con la instrucción rand()%100) y comprobar si es menor que 10. Esto solo pasará el 10% de las veces. Si no, aplica el mismo procedimiento (pero con el 20%) para ver si debe contener un tesoro y finalmente si no pasa nada de esto debe estar vacía.