EJERCICIOS CON CADENAS:

1. Implementa una función que recibe una frase y devuelve el número de apariciones de la letra 'a'.

2. Implementa una función que recibe una frase que contiene una lista de palabras separadas con blancos. La función devuelve el número de palabras de la lista que contienen más de 3 caracteres.

3. Escribir un programa que solicite dos cadenas que puede contener espacios, el programa deberá:
   - Decir qué cadena es mayor y cuál es menor alfabéticamente
   - Generar una nueva cadena que será la primera cadena leída, con las vocales convertidas a MAYÚSCULAS
   - Generar una nueva cadena que será la segunda cadena leída, con las consonantes convertidas a MAYÚSCULAS
   - Generar una nueva cadena concatenando las dos que ya fueron convertidas
   - Generar una nueva cadena que contenga la cadena concatenada invertida

   Al finalizar el programa, este deberá:
- Imprimir las dos cadenas originales
- Imprimir las dos cadenas con las conversiones
- Imprimir la cadena concatenada
- Imprimir la cadena concatenada invertida

Para escanear cadenas con espacios será necesario el uso de fgets en lugar de scanf.

Leyendo las restricciones vemos que vamos a comparar cadenas, para ello vamos a usar la función strcmp.

Después vemos que vamos a convertir algunas letras a mayúsculas, en el primer caso si son vocales y en el segundo si son consonantes.

Luego vamos a crear cadenas, así que necesitamos copiarlas usando `strcpy`. Para el caso de concatenar, será necesario el uso de `strcat`.

Finalmente, para invertir la cadena vamos a recorrerla cadena de fin a inicio e imprimir cada carácter, ya que el ejercicio no pide que almacenemos el resultado en una cadena distinta.

Al terminar el ejercicio vamos a imprimir todas las cadenas justo como fue solicitado.

Escanear cadenas

Comencemos a resolver las solicitudes. El primer requisito es escanear las cadenas con espacios:

```
char cadena1[MAXIMA_LONGITUD] = "";
char cadena2[MAXIMA_LONGITUD] = "";
printf("\nIngresa la cadena 1: ");
fgets(cadena1, MAXIMA_LONGITUD, stdin);
cadena1[strcspn(cadena1, "\r\n")] = 0;  //ver final de la nota
printf("\nIngresa la cadena 2: ");
fgets(cadena2, MAXIMA_LONGITUD, stdin);
cadena2[strcspn(cadena2, "\r\n")] = 0;
```

Decir cuál cadena es mayor

Ahora vamos a usar `strcmp` para saber cuál cadena es mayor alfabéticamente. Recordemos que `strcmp` regresa 0 si las cadenas son iguales, un número menor que 0 si la primer cadena es menor que la segunda, y un número mayor que 0 si la primer cadena es mayor que la segunda.

```
// Decir cuál es mayor y cuál es menor
int resultado = strcmp(cadena1, cadena2);
if (resultado >= 0)
{
   printf("Mayor: '%s'. Menor: '%s'\n", cadena1, cadena2);
}
else
{
   printf("Mayor: '%s'. Menor: '%s'\n", cadena2, cadena1);
}
```

Convirtiendo consonantes y vocales

Llega el turno del siguiente paso, el cual es convertir las vocales de la primera cadena en mayúsculas, y las consonantes de la segunda igualmente en mayúsculas.

Lo primero que tenemos que hacer es copiar las cadenas, después recorrerlas e ir modificando el carácter si es que es necesario. El código queda así:

```
// Cadena 1 con vocales mayúsculas
char cadena1VocalesMayusculas[MAXIMA_LONGITUD] = "";
strcpy(cadena1VocalesMayusculas, cadena1);
int i = 0;
for (i = 0; i < strlen(cadena1VocalesMayusculas); i++)
{
```

```
    char letraActual = cadena1VocalesMayusculas[i];
    if (esVocal(letraActual))
    {
        letraActual = toupper(letraActual);
    }
    cadena1VocalesMayusculas[i] = letraActual;
}
// Cadena 2 con consonantes mayúsculas
char cadena2ConsonantesMayusculas[MAXIMA_LONGITUD] = "";
strcpy(cadena2ConsonantesMayusculas, cadena2);
for (i = 0; i < strlen(cadena2ConsonantesMayusculas); i++)
{
    char letraActual = cadena2ConsonantesMayusculas[i];
    if (esConsonante(letraActual))
    {
        letraActual = toupper(letraActual);
    }
    cadena2ConsonantesMayusculas[i] = letraActual;
}
```

Concatenar cadenas

Ya casi terminamos. Ahora vamos a hacer el siguiente paso del ejercicio y es concatenar las cadenas modificadas. Eso lo hacemos con `strcat` y queda así:

```
// Cadena concatenando las dos anteriores
// Nota: asegúrate de que la longitud alcance
char cadenaConcatenada[MAXIMA_LONGITUD] = "";
strcat(cadenaConcatenada, cadena1VocalesMayusculas);
strcat(cadenaConcatenada, " ");
strcat(cadenaConcatenada, cadena2ConsonantesMayusculas);
```

Imprimir cadena invertida

El último requisito es imprimir la cadena concatenada anteriormente, pero de manera **invertida**. Para ello hacemos lo siguiente:

```
printf("Cadena concatenada invertida: ");
for (i = strlen(cadenaConcatenada) - 1; i >= 0; i--)
{
    printf("%c", cadenaConcatenada[i]);
}
printf("\n");
```

Poniendo todo junto

Ahora veamos el código completo que solicita ambas cadenas al usuario e imprime todo lo solicitado.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAXIMA_LONGITUD 100
int esVocal(char letra)
{
    // Convertir para simplificar comparaciones
    char letraMinuscula = tolower(letra);
    char vocales[] = "aeiou";
    // Recorrer las vocales y comparar si la letra es una de ellas
    int i;
    for (i = 0; vocales[i]; i++)
    {
        if (letraMinuscula == vocales[i])
        {
            return 1;
        }
    }
    // Si terminamos de recorrer y no regresamos true dentro del for, entonces no es vocal
    return 0;
}
int esConsonante(char letra)
{
    return isalpha(letra) && !esVocal(letra);
}

int main()
{
    char cadena1[MAXIMA_LONGITUD] = "";
    char cadena2[MAXIMA_LONGITUD] = "";
    printf("\nIngresa la cadena 1: ");
    fgets(cadena1, MAXIMA_LONGITUD, stdin);
    cadena1[strcspn(cadena1, "\r\n")] = 0;
    printf("\nIngresa la cadena 2: ");
    fgets(cadena2, MAXIMA_LONGITUD, stdin);
    cadena2[strcspn(cadena2, "\r\n")] = 0;
    // Decir cuál es mayor y cuál es menor
    int resultado = strcmp(cadena1, cadena2);
    if (resultado >= 0)
    {
        printf("Mayor: '%s'. Menor: '%s'\n", cadena1, cadena2);
    }
    else
    {
        printf("Mayor: '%s'. Menor: '%s'\n", cadena2, cadena1);
    }
    // Cadena 1 con vocales mayúsculas
    char cadena1VocalesMayusculas[MAXIMA_LONGITUD] = "";
```

```
    strcpy(cadena1VocalesMayusculas, cadena1);
    int i = 0;
    for (i = 0; i < strlen(cadena1VocalesMayusculas); i++)
    {
        char letraActual = cadena1VocalesMayusculas[i];
        if (esVocal(letraActual))
        {
            letraActual = toupper(letraActual);
        }
        cadena1VocalesMayusculas[i] = letraActual;
    }
    // Cadena 2 con consonantes mayúsculas
    char cadena2ConsonantesMayusculas[MAXIMA_LONGITUD] = "";
    strcpy(cadena2ConsonantesMayusculas, cadena2);
    for (i = 0; i < strlen(cadena2ConsonantesMayusculas); i++)
    {
        char letraActual = cadena2ConsonantesMayusculas[i];
        if (esConsonante(letraActual))
        {
            letraActual = toupper(letraActual);
        }
        cadena2ConsonantesMayusculas[i] = letraActual;
    }
    // Cadena concatenando las dos anteriores
    // Nota: asegúrate de que la longitud alcance
    char cadenaConcatenada[MAXIMA_LONGITUD] = "";
    strcat(cadenaConcatenada, cadena1VocalesMayusculas);
    strcat(cadenaConcatenada, " ");
    strcat(cadenaConcatenada, cadena2ConsonantesMayusculas);
    // Resultados
    printf("Cadena 1: '%s'\n", cadena1);
    printf("Cadena 2: '%s'\n", cadena2);
    printf("Cadena 1 con vocales mayúsculas: '%s'\n", cadena1VocalesMayusculas);
    printf("Cadena 2 con consonantes mayúsculas: '%s'\n", cadena2ConsonantesMayusculas);
    printf("Cadena concatenada: '%s'\n", cadenaConcatenada);
    printf("Cadena concatenada invertida: '");
    for (i = strlen(cadenaConcatenada) - 1; i >= 0; i--)
    {
        printf("%c", cadenaConcatenada[i]);
    }
    printf("'\n");
    return 0;
}
```

Usar fgets en lugar de scanf en C

Cuando aprendemos algoritmos en C, la forma de escanear variables por teclado es a través de **scanf**. Si bien este método funciona, existe el peligro de un **desbordamiento de búfer**. fgets **escanea una variable pero únicamente hasta donde le digamos**; es decir, nosotros le decimos cuánto debe leer (el tamaño). Scanf en cambio no hace eso, lee todo dentro de la posición en memoria.

La sintaxis es:

```
fgets(cadena, longitud, stream);
```

En donde la **cadena** es la variable en donde almacenaremos lo leído.

La **longitud** es cuántos caracteres leeremos (aquí es en donde se previene el desbordamiento) y el **stream** es un stream de tipo **FILE**; recordemos que **stdin** es un stream así que viene perfecto para leer datos que el usuario introduzca.

```
#include <stdio.h>
#define LONGITUD 20
int main(){
   char nombre[LONGITUD];
   printf("Dime tu nombre: ");
   fgets(nombre, LONGITUD, stdin);
   printf("Hola, %s", nombre);
}
```

Prevenir desbordamiento de búfer con fgets en C

Un ejemplo usando `fgets` en lugar de `scanf`:

```
/* Ejemplo simple de prevención

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

// Por si decidimos cambiar la longitud después
#define LONGITUD 4

int main() {
  char cadena[LONGITUD];
  int autenticado = 0; // 1 es que sí, 0 que no
```

```
    printf("Ingresa la contraseña:\n");
    fgets(cadena, LONGITUD, stdin); // ----- Aquí está la prevención del desbordamiento
                cadena[strcspn(cadena, "\r\n")] = 0;
    if (strcmp(cadena, "123") == 0) {
      autenticado = 1;
    }
    if (autenticado)
      printf("Bienvenido al programa");
    else
      printf("Acceso denegado");

    printf("\nAl final de todo, el valor que tiene 'autenticado' es: %d",
           autenticado);
}
```

Lo que cambia es que ahora le indicamos a fgets **que lea hasta 4 caracteres** (sólo usamos 3 pero necesitamos dejar uno extra) y lo demás no sé en dónde lo ponga, pero eso me asegura de que no sobrescriba a otras variables ni corrompa la memoria.

De esta manera prevenimos el desbordamiento de búfer en C, al menos cuando leemos variables del usuario.

Por cierto, te estarás preguntando **por qué declaramos la cadena para que almacene 4,** si sólo nos importan 3, y esto es porque **al final `fgets` le agrega un salto de línea**.

Limpiar cadena escaneada con fgets en C

Muy bien, ya hemos escaneado y prevenido un desbordamiento de búfer, pero falta **limpiar el último carácter**. Como dije, esta función devuelve todo lo leído, incluso el salto de línea.

Para quitarlo y limpiar nuestra cadena (en caso de que lo requieras) hacemos esto:

```
cadena[strcspn(cadena, "\r\n")] = 0;
```

En donde la variable *cadena* es tu búfer. Lo que hacemos es **buscar el índice de la primera aparición del salto de línea y remplazarlo con el carácter nulo**.