TP N°6: Arreglos

La siguiente guía cubre los contenidos vistos en las clases teóricas:
10. Arreglos unidimensionales y multidimensionales
11. Introducción a punteros
12. Cadenas de Caracteres

Antes de empezar con los ejercicios de esta guía se recomienda la resolución de la **Autoevaluación 3** disponible en Campus ITBA, para consolidar los contenidos vistos en la teórica.

A partir de esta práctica **modularice al máximo** los programas que se solicitan, separando especialmente las funciones de entrada y salida de datos de aquellas que sólo realizan procesamiento.
Cuando se soliciten solamente funciones, se deberá escribir además un programa simple que verifique el correcto funcionamiento de las mismas.

A cada ejercicio se le agregó una serie de asteriscos, donde la cantidad de los mismos indica el nivel de dificulatad, representando un asterisco un ejercicio "sencillo" y tres asteriscos un ejercicio de parcial.

Parte 1: Vectores y matrices

*Ejercicio 1 **

Detectar errores de compilación en el siguiente programa:

```
#define N 10
#define M 20

int
main (void)
{
      int dim = 7;
      int vectorA[ M * N];
      int vectorB[ -10 ];
      int vectorC[ 10.0 ];
      int vectorD[dim];
      vectorC[2.5] = 'a';
      vectorB[-1] = 5;
      vectorA['0'] = 10;
      vectorC[vectorA[48]] = 5.5;
      vectorA[1000] = 0;
      vectorA[M * N] = 10;
      return 0;
}
```

*Ejercicio 2 **

Reescribir el ejercicio 10 de la práctica 5 de forma tal que quede separado correctamente el front-end del back-end.

> Para la opción de simplificar una fracción, la obtención del numerador y denominador pertenece al front-end. Luego se invoca a una función de back-end que recibe esos parámetros y los modifica. ¿Cómo se puede hacer para que la función de back-end reciba ambos y los pueda modificar?

*Ejercicio 3 ***

Dado un arreglo lineal de números reales, cuyo indicador de fin de elementos es cero, escribir una función para obtener la mayor diferencia entre dos elementos consecutivos. (En no más de 5 líneas). Tener en cuenta que los números pueden ser negativos. En caso de que el vector tenga un solo elemento deberá retornar como diferencia el valor cero.

> A partir de ahora, cuando escriban sus propias funciones y consideren que los arreglos que recibas como parámetros de entrada no deben ser alterados en el cuerpo de la función, no se olviden de colocar el calificador ***const***.

*Ejercicio 4 ***

Escribir una función que reciba un vector desordenado de números enteros y su dimensión, y construya otro vector eliminando los valores repetidos. La función deberá retornar en su nombre la dimensión del nuevo vector (La función solicitada no debe superar las 10 líneas).

*Ejercicio 5 ****

Implementar la función anterior para vectores ordenados.

*Ejercicio 6 **

Dado un arreglo ordenado ascendentemente se pide escribir una función que reciba como parámetro de entrada/salida el arreglo y como parámetro de entrada su dimensión y que lo devuelva desordenado, simulando la mezcla de un mazo de cartas o de un bolillero (en no más de 10 líneas).

*Ejercicio 7 ***

Hacer una función que reciba dos parámetros de entrada representando arreglos de números enteros positivos, ordenados en forma ascendente y sin elementos repetidos. El último elemento de cada arreglo es -1. La función debe devolver en un tercer parámetro de salida un arreglo ordenado con la unión de los dos primeros, también terminado en -1.

Ejemplo: Dados v1 y v2 arreglos de enteros, se espera que la unión sea:

v1 = {1,2,3,-1} y v2 = {2,3,4,-1} → unión = {1,2,3,4,-1}.

v1 = {1,2,3,-1} y v2 = {1,2,3,-1} → unión = {1,2,3,-1}.

> Recorrer una sola vez cada arreglo.

*Ejercicio 8 **

Repetir el ejercicio anterior, teniendo en cuenta que los arreglos de entrada pueden tener elementos repetidos, pero el de salida NO debe tener repeticiones.

Ejercicio 9 **

Se desea calcular la ***desviación estándar*** de un arreglo de números enteros. Los números del arreglo toman valores entre 0 y 15 inclusive, por lo que para almacenar cada número se utilizarán solo 4 bits, pudiendo almacenar dos números en un solo byte.

Para representar dicho arreglo se utilizará un vector de caracteres, donde cada elemento del vector contendrá dos números (uno en los cuatro bits superiores y el otro en los cuatro bits inferiores). Escribir una función que reciba un arreglo como el mencionado anteriormente y la cantidad de números que contiene y retorne en su nombre la ***desviación estándar*** de los números recibidos.

Ejemplo: Si se define el siguiente arreglo:

```
unsigned char arreglo[] = { 0x37, 0xF2, 0x03, 0x4A, 0xFF };
```

Representa al arreglo de los elementos: 3, 7, 15, 2, 0, 3, 4, 10, 15, 15.

*Ejercicio 10 **

Los laboratorios de Propulsión por Reacción tienen la representación del cielo y sus estrellas, digitalizada en una matriz bidimensional de hasta 80 columnas por 20 filas. Cada elemento de la misma representa la cantidad de luz que hay en una zona del cielo con un rango de intensidad entre 0 y 20. En el lugar de coordenadas **(i,j)** del cielo se considera que hay una estrella si el elemento $\mathbf{A_{ij}}$ correspondiente cumple con la siguiente relación:

(A[i,j] + suma de las ocho intensidades circundantes) / 9 > 10

Escribir una función (en no más de 15 líneas) que reciba tres parámetros de entrada representando a una matriz de dichas características y sus dimensiones. Dicha función debe localizar gráficamente las estrellas en la pantalla representando las mismas con el carácter '*'. La función debe ignorar las aristas de la matriz.

Para completar la matriz no hace falta interactuar con el usuario, se puede utilizar números aleatorios.
Para una correcta separación de front y back, necesitás hacer una función que genere la matriz (back-end) y otra que grafique el "cielo" en salida estándar (front-end).

Ejercicio 11 ***

Escribir ***una función*** que ordene las filas de una matriz de cualquier tamaño, según el valor de una determinada columna. La función recibirá como parámetros la matriz, la cantidad de filas, la cantidad de columnas y el número de columna a tomar como clave de ordenación, teniendo en cuenta que la primera columna es la columna 1 (uno).

Para ordenar se puede usar el *bubble sort* (https://www.geeksforgeeks.org/bubble-sort/) o *selection sort* (https://www.geeksforgeeks.org/selection-sort/)

Ejemplo:
Si la matriz original fuese

1	2	3	4	5
6	7	8	9	10
3	5	8	2	1
8	1	3	6	7

Si se pide ordenar por la columna 1 :

1	2	3	4	5
3	5	8	2	1
6	7	8	9	10
8	1	3	6	7

Si se pide ordenar por la columna 3 :

1	2	3	4	5
8	1	3	6	7
3	5	8	2	1
6	7	8	9	10

A igualdad de valores, el orden de las filas coincidentes es indistinto

*Ejercicio 12 **

Escribir una función que cambie una matriz cuadrada por su traspuesta, recibiendo sólo los siguientes 2 parámetros:

- la matriz cuadrada
- un número entero positivo que indique la dimensión de la matriz

Dicha función debe hacer la conversión sobre la misma matriz recibida, sin usar vectores auxiliares.

La traspuesta de una matriz se obtiene cambiando cada elemento A_{ij} por el elemento A_{ji}.

*Ejercicio 13 ***

Escribir una función que realice el producto de dos matrices cuadradas y lo devuelva en una tercera. El algoritmo de la función que realiza el producto no debe tener más de dos ciclos ***for*** anidados explícitamente, pero sí puede utilizar funciones auxiliares que contengan ciclos (Ninguna de las funciones debe superar las 5 líneas).

*Ejercicio 14 *** (ex 18)*

Escribir la función `contiene` que reciba dos vectores de enteros y la dimensión de cada uno y retorne:

- 1 si todos los elementos del primer vector están en el segundo
- 2 si todos los elementos del segundo vector están en el primero
- 0 en caso contrario

Ejemplos:

Vector 1	**Vector 2**	**Respuesta**
{1, 6, 5, 3, 2}	{1, 2}	2
{1, 3, 2}	{3, 1, 2}	1 ó 2
{}	{5, 8, 54}	1
{1}	{2}	0
{1, 1, 5, 5, 9, 1}	{5, 9, 3, 1}	1
{6, 8}	{6, 6, 6}	2
{}	{}	1 ó 2

*Ejercicio 15 *** (ex 21)*

Una imagen bmp se almacena en un archivo como una matriz de píxeles.
Si la imagen es de escala de grises, cada píxel ocupa un solo byte. Así, si la imagen tiene 20x30 píxeles, ocupa 600 bytes donde un byte con el valor 0x00 corresponde al negro y un byte con el valor 0xFF corresponde al blanco. Cualquier valor entre 0x00 y 0xFF será una tonalidad de gris.
Realizar una función **`suavizar`** que, dada una imagen, la suavice aplicando un filtro de media de valor W, con W impar. Esto significa que cada píxel (i, j) de la nueva imagen se obtiene a partir de la media aritmética de los píxeles que se encuentran en la imagen original de la submatriz de tamaño WxW centrada en (i,j).

La función **`suavizar`** recibe como únicos parámetros la matriz y W. Si W es menor que 3 o no es impar, la función no altera la imagen.

El ancho y alto de la imagen son las constantes simbólicas ANCHO y ALTO respectivamente.

Ejemplo: Sea la imagen de tamaño 6x5

10	10	20	23	24
10	12	18	25	23
12	14	18	26	22
12	14	19	20	22
13	16	19	20	22
14	14	19	21	23

Si se invoca la función **`suavizar`** , para esa imagen, con W=3, se tendrá que calcular:

Píxel (0,0) como promedio de:

10	10
10	12

Píxel (0,1) como promedio de:

10	10	20
10	12	18

Píxel (1,1) como promedio de:

10	10	20
10	12	18
12	14	18

Etc.

Finalmente, debería quedar:

10	13	18	22	23
11	13	18	22	23
12	14	18	21	23
13	15	18	20	22
13	15	18	20	21
14	15	18	20	21

*Ejercicio 16 *** (ex 22)*

Escribir una función que reciba una matriz de **números enteros** y cuya cantidad de filas está dada por la constante simbólica FILS y la cantidad de columnas por la constante simbólica COLS.

Se asegura que ambas constantes son mayores o iguales a 2 (no hace falta validarlo)

La función debe retornar 1 si la matriz es "ascendente", -1 si es "descendente" y 0 si no es ascendente ni descendente. Se dice que una matriz es **ascendente** si recorriendola en forma ordenada por filas **cada elemento es mayor o igual al anterior**, En forma análoga se define una matriz descendente.
El recorrido comienza por el elemento [0][0], continúa con el [0][1] y luego de terminar la primera fila hace lo mismo con la segunda fila, tercer fila, etc.

Ejemplos:

```
#define FILS 3
#define COLS 4
```

```
int m1[][COLS] = {
      {-6,-5,2,6},
      {7,9,9,14},
      {21,32,45,46}
}; // m1 es ascendente

int m2[][COLS] = {
      {19,13,12,8},
      {7,7,5,-1},
      {-6,-10,-14,-16}
}; // m2 es descendente

int m3[][COLS] = {
      {-6,-5,-6,6},
      {7,9,10,14},
      {21,32,45,46}
} // m3 no es ascendente ni descendente

int m4[][COLS] = {
      {19,13,18,8},
      {7,5,2,-1},
      {-6,-10,-14}
}; // m4 no es ascendente ni descendente


int m5[][COLS] = {
      {1,1,1,1},
      {1,1,1,1},
      {1,1,1,1}
}; // m5 es ascendente y descendente. La función podría devolver 1 ó -1
```

*Ejercicio 17 ****

Escribir una función **`armaFilas`** que recibe una matriz de enteros de N filas y M columnas, y un vector de dimensión N.

La función debe armar los números que se forman con cada fila y guardarlos en el vector.

Si hubiera números negativos o de más de una cifra en una fila, no se completa el armado de ese número, pero se sigue con la próxima fila.

La función retorna en su nombre la dimensión final del vector de números.

Ejemplos:

Dada la siguiente matriz el vector queda {}

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

y la función retorna 0.

Dada la siguiente matriz, el vector resulta: {1325, 5018, 9672, 4541}

1	3	2	5
5	0	1	8
9	6	7	2
4	5	4	1

y la función retorna 4.

Dada la siguiente matriz, el vector resulta: {1325, 9672, 4541}

1	3	2	5
5	10	1	8
9	6	7	2
4	5	4	1

y la función retorna 3.

*Ejercicio 18 ****

Una fila es amiga de otra si todos los elementos de una fila están incluidos en otra. Por ejemplo:

$$F_1 = (1\ 2\ 2\ 3)$$
$$F_2 = (0\ 1\ 2\ 3)$$
$$F_3 = (1\ 2\ 3\ 4)$$

- F_1 es amiga de F_2 pues todos los elementos de F_1 están incluidos en F_2
- F_2 no es amiga de F_1 pues el número 0 presente en F_2 no está incluido en F_1
- F_1 es amiga de F_3 pues todos los elementos de F_1 están incluidos en F_3
- F_3 no es amiga de F_1 porque el número 4 presente en F_3 no está incluido en F_1

Una matriz es amiga de otra si todas las filas de la primera matriz son amigas de alguna fila de la segunda matriz. Es decir, siendo:

$$M_1 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 4 & 5 & 6 \\ 7 & 8 & 8 & 9 \end{pmatrix} \quad M_2 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ -3 & 7 & 8 & 9 \\ -1 & 3 & 4 & 7 \\ 4 & 5 & 6 & 8 \end{pmatrix} \quad M_3 = (2 \quad 3 \quad 3 \quad 7)$$

- M_1 es amiga de M_2 pues
  - la fila $(0\ 1\ 2\ 3)$ de M_1 es amiga de la fila $(0\ 1\ 2\ 3)$ de M_2
  - la fila $(4\ 4\ 5\ 6)$ de M_1 es amiga de la fila $(4\ 5\ 6\ 8)$ de M_2
  - la fila $(7\ 8\ 8\ 9)$ de M_1 es amiga de la fila $(-\ 3\ 7\ 8\ 9)$ de M_2
- M_2 no es amiga de M_1 pues la fila $(-\ 3\ 7\ 8\ 9)$ de M_2 no es amiga de ninguna de las filas de M_1
- M_1 no es amiga de M_3 pues la fila $(0\ 1\ 2\ 3)$ de M_1 no es amiga de ninguna de las filas de M_3
- M_3 no es amiga de M_1 pues la fila $(2\ 3\ 3\ 7)$ de M_3 no es amiga de ninguna de las filas de M_1

Se pide escribir la función **sonAmigas** que reciba los siguientes parámetros:

- **m1**: la primera matriz de enteros
- **fils1**: la cantidad de filas de la primera matriz de enteros
- **m2**: la segunda matriz de enteros
- **fils2**: la cantidad de filas de la segunda matriz de enteros

Nota 1: Se cuenta con la constante simbólica COLS que indica la cantidad de columnas de la primera y segunda matriz de enteros.
Nota 2: Todas las filas tienen sus elementos ordenados de manera no descendente (cada elemento es menor o igual al siguiente)

La función debe retornar:

- **1**: si la primera matriz es amiga de la segunda
- **2:** si la segunda matriz es amiga de la primera
- **0**: en cualquier otro caso

Si ambas son amigas, la función puede retornar 1 ó 2.

Ejemplo de invocación:

```
int
main(void)
{
  int m1[][COLS] = {{0,1,2,3}, {4,4,5,6}, {7,8,8,9}};
  int m2[][COLS] = {{0,1,2,3}, {-3,7,8,9}, {-1,3,4,7}, {4,5,6,8}};
  int m3[][COLS] = {{2,3,3,7}};
  printf("%d\n", sonAmigas(m1,3,m2,4)); // Imprime 1
  printf("%d\n", sonAmigas(m2,4,m1,3)); // Imprime 2
  printf("%d\n", sonAmigas(m1,3,m3,1)); // Imprime 0

  return 0;
}
```

Ejercicio 19 ***

Resolver el ejercicio anterior, pero teniendo en cuenta que cada fila no necesariamente está ordenada.

*Ejercicio 20 ****

Escribir una función que compruebe si una solución dada a un Sudoku es correcta sabiendo que:

- Un tablero sudoku se compone de una matriz de 9 filas por 9 columnas. A su vez el tablero se subdivide en 9 subcuadrados de 9 casillas cada uno (3x3) NO SUPERPUESTOS (un número en una posición NO puede pertenecer a dos subcuadrados).
- Se debe verificar que todas las casillas contienen un número comprendido entre el 1 y el 9. Si alguna casilla contiene un número menor a 1 o mayor a 9, es incorrecta la matriz.
- Por fila, no hay valores repetidos
- Por columna, no hay valores repetidos
- Por subcuadrado, no hay valores repetidos

Ejemplo:

Si se recibe la siguiente matriz, la función retornará 1 (uno)

3	8	1	9	7	6	5	4	2
2	4	7	5	3	8	1	9	6
5	6	9	2	1	4	8	7	3
6	7	4	8	5	2	3	1	9
1	3	5	7	4	9	6	2	8
9	2	8	1	6	3	7	5	4
4	1	2	6	8	5	9	3	7
7	9	6	3	2	1	4	8	5
8	5	3	4	9	7	2	6	1

Si se recibe la siguiente matriz, la función retornará 0 (cero), porque por ejemplo la segunda fila tiene dos veces el 3

3	8	1	9	7	6	5	4	2
2	3	7	5	3	8	1	9	6
5	6	9	2	1	4	8	7	3
6	7	4	8	7	2	3	1	9
1	3	5	7	4	9	6	2	8
9	2	8	1	6	3	7	5	4
4	1	2	6	8	5	3	3	7
7	9	6	3	2	1	4	8	5
8	5	3	4	9	7	2	6	1

Si se recibe la siguiente matriz, la función retornará 0 (cero), porque por ejemplo la submatriz superior izquierda contiene dos veces el 3

3	8	1	9	7	6	5	4	2
2	4	7	5	9	8	1	3	6
5	6	3	2	1	4	8	7	9
6	7	4	8	5	2	3	1	9
1	3	5	7	4	9	6	2	8
9	2	8	1	6	3	7	5	4
4	1	2	6	8	5	3	9	7
7	9	6	3	2	1	4	8	5
8	5	9	4	3	7	2	6	1

Parte 2: Punteros

*Ejercicio 21 **

Reescribir el ejercicio 10 de la práctica 5 de forma tal que quede separado correctamente el front-end del back-end.

Para la opción de simplificar una fracción, la obtención del numerador y denominador pertenece al front-end. Luego se invoca a una función de back-end que recibe esos parámetros y los modifica. ¿Cómo se puede hacer para que la función de back-end reciba el valor izquierdo y no el valor derecho?

*Ejercicio 22 ***

Escribir una función **`secuenciaAsc`** que reciba un vector de enteros (sin orden) y su dimensión y almacene en dos parámetros de salida:

- **`comienzo`**: Dónde comienza la secuencia de números ordenados en forma ascendente (cada elemento debe ser mayor al anterior) de mayor longitud
- **`longitud`**: Longitud de esa secuencia

En caso de haber más de una secuencia con la mayor longitud debe almacenar en **`comienzo`** donde comienza la primera de ellas.

Ejemplos:

- Para el vector `{1, 1, 3, 4, 4, 7, 10, 9, 11}` el comienzo es `1` y su longitud es `3`
- Para el vector vacío comienzo y longitud son `0 (cero)`
- Para los vectores `{3}`, `{2, 2, 2}` y `{3, 2, 1, 0, -1}` el comienzo es `0` y la longitud es `1`
- Para el vector `{1, 2, 3, 4, 5, 7, 10, 90, 111}` el comienzo es `0` y la longitud 9

Parte 3: Strings

Ejercicio 23 *

Escribir una función que reciba un string con el formato 'dd/mm/yyyy' que representa una fecha y devuelva en tres parámetros de salida el número de día, el número del mes y el año. En caso de que la fecha no sea correcta retorna el valor cero y no altera los parámetros recibidos, caso contrario retorna 1. (Ninguna función debe superar las 8 líneas).

Consejo #1

Para convertir una parte del string fecha a un entero podés usar la función de de la Standard System Library **int atoi(const char * s).** Un ejemplo de uso:

```
char * fecha = "07/09/1991";
int dia = atoi(fecha);       /*  dia = 7    */
int mes = atoi(fecha + 3);   /*  mes = 9    */
int anio = atoi(fecha + 6);  /* anio = 1991 */
```

Vas a ver en detalle ésta y más funciones relacionadas en la clase teórica **13. Biblioteca estándar (Segunda Parte)**

Consejo #2

Para validar si una fecha es correcta o no, necesitás saber la cantidad de días que tiene cada mes. Este número varía si el año es bisiesto o no. Se recomienda utilizar el arreglo **daytab** del ejemplo de la sección 5.7 del libro de consulta. Luego necesitás implementar una función o macro **int esBisiesto(unsigned int anio)**.

Ejercicio 24 **

Hacer una función que reciba como único parámetro de entrada/salida un string y elimine los espacios de más. Por ejemplo, si recibe **“Hola mundo ”**, deberá transformarlo en **“Hola mundo ”**

Ejercicio 25 **

Escribir una función que devuelva un subvector de un arreglo de caracteres, recibiendo sólo los siguientes parámetros:

- **arregloIn**: vector de entrada, *null terminated*.
- **arregloOut**: vector de salida *null terminated*.
- **desde, hasta**: valores enteros, representan las posiciones de valores a copiar.
- **dimMax**: dimensión máxima del vector de salida (incluyendo el cero).

Dicha función debe almacenar en arregloOut los elementos de arregloIn cuyos índices estén comprendidos entre **desde** y **hasta** inclusive y colocar el valor cero al final. La cantidad máxima de elementos a copiar está dada por el parámetro **dimMax**. En caso de que la cantidad de valores a copiar supere dimMax, se copiarán solamente dimMax - 1 valores.

Si alguno de los parámetros es erróneo o la posición **desde** esté fuera de los límites del arreglo de entrada, la función debe retornar el valor cero y no alterar el arreglo de salida, caso contrario debe retornar la cantidad de elementos copiados (sin contar el cero final).

*Ejercicio 26 ***

a) Escribir la función **insertaDesde**, que recibe dos strings (null terminated) y un carácter. Al primer string se le inserta el segundo a partir de la primera aparición del carácter indicado. Si el carácter no aparece en el primer string, el mismo no debe ser alterado.

Ejemplo 1:

```
char str1[20] = "manuel";
char str2[] = "javi";
insertaDesde(str1, str2,'n');
printf("%s\n",str1); // muestra majavi
```

Ejemplo 2:

```
char str1[20] = "manuel";
insertaDesde(str1, "javi" ,'l');
printf("%s\n",str1); // muestra manuejavi
```

Ejemplo 3:

```
char str1[20] = "manuel";
char str2[] = "javi";
insertaDesde(str1, str2,'j');
 printf("%s\n",str1); // muestra manuel (sin modificaciones porque 'j' no aparece en la primer palabra
```

b) Escribir un ejemplo de invocación que asegure que durante la ejecución de la función anterior se produzca un "segmentation fault" (el programa aborta por una operación no permitida).

*Ejercicio 27 ***

Escribir la función **`analize`** que reciba un string de nombre **`text`** (se espera que sea muy extenso) y un vector de chars de nombre **`chars`** que tiene al menos 256 posiciones reservadas pero no inicializadas (contienen basura). La función debe dejar en **`chars`** los distintos caracteres que aparecen en **`text`**, ordenados ascendentemente por valor ASCII y sin repetir. El vector **`chars`** debe quedar *null terminated*.
El siguiente programa

```c
#define CHARS_DIM 256

int main(void) {
 char chars[CHARS_DIM];
 analize("El zorro mete la cola, Pero no la pata!", chars);
 printf("%s\n", chars);
 return 0;
}
```

imprimirá lo siguiente (sin las comillas)

```
" !,EPacelmnoprtz"
```

*Ejercicio 28 ****

Escribir la función **`elimVocales`** que reciba un string `s` y una matriz de 5 columnas y `n` filas, donde cada columna representa una vocal y `n` es un parámetro de la función.

La función deberá devolver el string `s` sin las vocales (ya sean mayúsculas o minúsculas), y la matriz con las ubicaciones de las vocales eliminadas. Cada columna de la matriz debe "cerrarse" con un -1.

La función tiene que controlar el espacio disponible en la matriz para seguir guardando ubicaciones. Si no hay espacio suficiente, devuelve ERROR, aunque haya quedado modificado el string.

Ejemplo: Se invoca la función con `s` = "las buenas ideas escasean, si"

<table>
<tr><td>Invocada con 6 filas</td><td>Invocada con 5 filas</td></tr>
<tr><td>El resultado es EXITO. <code>s</code> = "ls bns ds scsn, s"
M:
<table>
<tr><td>1</td><td>6</td><td>11</td><td>-1</td><td>5</td></tr>
<tr><td>8</td><td>13</td><td>28</td><td></td><td>-1</td></tr>
<tr><td>14</td><td>17</td><td>-1</td><td></td><td></td></tr>
<tr><td>20</td><td>22</td><td></td><td></td><td></td></tr>
<tr><td>23</td><td>-1</td><td></td><td></td><td></td></tr>
<tr><td>-1</td><td></td><td></td><td></td><td></td></tr>
</table></td><td>El resultado es ERROR. <code>s</code> = "ls bns ds scsan, s" (la posición de la quinta 'a' ya no se puede guardar)
M:
<table>
<tr><td>1</td><td>6</td><td>11</td><td>-1</td><td>5</td></tr>
<tr><td>8</td><td>13</td><td>28</td><td></td><td>-1</td></tr>
<tr><td>14</td><td>17</td><td>-1</td><td></td><td></td></tr>
<tr><td>20</td><td>22</td><td></td><td></td><td></td></tr>
<tr><td>-1</td><td>-1</td><td></td><td></td><td></td></tr>
</table></td></tr>
</table>

Nota: ERROR y EXITO son constantes simbólicas previamente definidas

*Ejercicio 29 ***

Escribir la función `deleteChars` que recibe dos strings y elimine en ambos los caracteres que sean iguales y estén en la misma posición. Ambos strings pueden tener longitudes diferentes.

Ejemplo:

Si recibe "Hola, soy un string" y "Yo soy otro string" deben quedar como "Hla, sy un string" y "Y soy tro string" respectivamente
Si recibe "Tengo algunos chars" y "", no deben cambiar
Si recibe "ABCDE" y "abcde", no deben cambiar
Si recibe dos string iguales, ambos deben quedar vacíos

*Ejercicio 30 **

Dado el siguiente fragmento de código, indicar cuáles de las siguientes afirmaciones son verdaderas.

```
typedef enum {MON=0, TUE, WEN, THU, FRI, SAT, SUN} TDay;
typedef TDay TVec[10];
typedef TVec TMat[5];

TMat m1;
int m2[10][5];
int m3[5][10];
```

a) no compila porque falta un nombre luego de la palabra reservada enum
b) TDay es una variable de tipo entero
c) TVec es un vector de 10 enteros, que deberían valer entre 0 y 6 inclusive
d) m1 y m2 son del mismo tipo
e) m1 y m3 son del mismo tipo
f) m2 y m3 son del mismo tipo
g) m1 es una "matriz" cuyos elementos solamente pueden tomar los valores definidos en el enum TDay (entre 0 y 6 inclusive)
h) m1 es una "matriz" cuyos elementos solamente deberían tomar los valores definidos en el enum TDay (entre 0 y 6 inclusive)
i) El compilador no reserva espacio de memoria para TVec

*Ejercicio 31 **

Resolver los siguientes ejercicios del libro de texto: **5-4 y 5-5**

Exercise 5-4. Write the function `strend(s,t)`, which returns 1 if the string `t` occurs at the end of the string `s`, and zero otherwise.

Exercise 5-5. Write versions of the library functions `strncpy`, `strncat`, and `strncmp`, which operate on at most the first `n` characters of their argument strings. For example, `strncpy(s,t,n)` copies at most `n` characters of `t` to `s`. Full descriptions are in Appendix B.

Ejercicio 32 (optativo)

Se pretende expresar números reales positivos en notación exponencial normalizada con un único dígito para la parte entera y cantidad variable de dígitos para la mantisa.

Para esto se pide escribir la función entera ***NotacionExp*** que recibe dos parámetros:

- una cadena de caracteres de entrada conteniendo el número original.
- una cadena de caracteres de salida donde dejará el número normalizado, eliminado los ceros no significativos.

En caso de error en la entrada, la función debe devolver el valor cero y dejar en la respuesta el string vacío, caso contrario debe retornar el valor uno.

El número original puede no tener parte fraccionaria, esto implica que "123.0" es válido y "123" también; y en ambos casos la respuesta será "1.23E+02".

Ejemplos:

Texto Original	***Respuesta***
"1"	"1E+00"
"0012"	"1.2E+01"
"145.720"	"1.4572E+02"
"134597"	"1.34597E+05"
"0.34"	"3.4E-01"
"0.00200"	"2E-03"
"987654321011"	"9.87654321011E+11"

*Ejercicio 33 ****

Existe un método para encriptar mensajes que consiste en mezclar las letras de un alfabeto y reemplazar cada letra del mensaje por la nueva letra que ocupa su lugar en el alfabeto.

Se pide escribir tres funciones:

- **crearAlfabeto**: recibe como parámetro de salida un vector de caracteres y lo completa con el alfabeto mezclado (letras 'A' a 'Z'). El alfabeto mezclado debe armarse utilizando un ciclo ***for.***
- **codificar**: que recibe dos parámetros de entrada que representan un mensaje a codificar y un alfabeto, y un parámetro de salida en el cual se devuelve el mensaje codificado.
- **decodificar**: recibe dos parámetros de entrada que representan un mensaje a decodificar y un alfabeto, y un parámetro de salida en el cual se devuelve el mensaje decodificado.

Ejemplo:

Alfabeto original = { A, B, C , D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z }
Alfabeto mezclado = { C, I, N, Q, U, X, A, E, H, O, R, T, Z, B, G, K, M, Y, D, L, S, V, F, J, P, W }

Si se codifica el mensaje **"Abeja Reina"** se obtiene el mensaje **"CIUOC YUHBC".**
Si se decodifica el mensaje **"CYAUBLHC"** se obtiene el mensaje **"ARGENTINA"**.

Ejercicio 34 (Ejercicio extra)

El Juego de la Vida de Conway (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life) se desarrolla sobre un tablero rectangular, donde cada casilla puede tener una célula muerta (representada con un 0) o una célula viva (representada por un uno). En cada turno una célula puede seguir en su mismo estado, puede morir o puede revivir.
Las reglas para determinar cómo va cambiando el tablero (la matriz) son las siguientes

- Si una célula está muerta y rodeada por exactamente 3 células vivas (entre sus 8 celdas vecinas exactamente 3 tienen el valor 1) entonces la célula revive
- Una célula viva con 2 ó 3 células vecinas vivas sigue viva, si hay menos muere por soledad, si hay más muere por sobrepoblación.

El juego continúa hasta que todas las células mueren o se llega a un estado estable (no hay cambios). Para este ejercicio se pide escribir la función `nextStep` que reciba una matriz de chars de `FILS` filas por `COLS` columnas que representa el estado actual del juego y deje en esa misma matriz el estado siguiente. Además la función debe retornar 1 (uno) si hubo cambios y 0 (cero) si no hubo cambios.
Se considera que por fuera de la matriz no puede haber vida

Ejemplo de evolución en una matriz de 4x4

0	1	0	0
0	0	1	0
1	1	1	0
0	0	0	0

0	0	0	0
1	0	1	0
0	1	1	0
0	1	0	0

0	0	0	0
0	0	1	0
1	0	1	0
0	1	1	0

En el siguiente ejemplo no hay cambios

0	0	0	0
0	1	1	0
0	1	1	0
0	0	0	0

En el siguiente se produce un bucle infinito

0	0	1	0
0	0	1	0
0	0	1	0
0	0	0	0

0	0	0	0
0	1	1	1
0	0	0	0
0	0	0	0

0	0	1	0
0	0	1	0
0	0	1	0
0	0	0	0

Pueden encontrar una versión del juego online en https://playgameoflife.com

Ejercicio 35 (Ejercicio extra)

Si bien pareciera ser algo ya perimido por usos y costumbres, es importante recordar que en cada palabra que utilizamos hay una sílaba tónica, es decir, aquella que pronunciamos con una mayor intensidad (también llamada la sílaba acentuada). Las palabras pueden clasificarse, según la posición de la sílaba tónica, de la siguiente forma:

- Una palabra es aguda cuando la sílaba tónica es la última sílaba.
- Una palabra es grave cuando la sílaba tónica es la anteúltima sílaba (la segunda contando de atrás para adelante).
- Una palabra es esdrújula si la sílaba tónica es la antepenúltima sílaba (la tercera contando de atrás hacia adelante).
- Una palabra es sobresdrújula si la sílaba tónica es la cuarta contando de atrás hacia adelante.

Las siguientes reglas son algunas de las que nos permiten identificar cuando una palabra lleva tilde (también llamada acento ortográfico):

- Las palabras esdrújulas y sobreesdrújulas siempre llevan tilde
- Las palabras agudas llevan tilde si terminan en n, s o vocal.
- Las palabras graves llevan tilde si NO terminan en n, ni en s, ni en vocal.

Por ejemplo, la palabra corazón lleva tilde porque es aguda (su sílaba tónica es la última) y termina en n. Pero la palabra corazones no lleva tilde, porque es grave y termina en s.

Algunos ejemplos más:

- Palabras agudas: café - rubí - menú - algún - jamás - según - cantidad - papel - reloj
- Palabras graves: árbol - cárcel - ángel - problema - adulto - martes - zapato - examen (*) - imagen (*)
- Palabras esdrújulas: Sudáfrica - música - miércoles - sílaba - máquina - gramática - esdrújula
- Palabras sobreesdrújulas: cuéntamelo - devuélveselo - éticamente - fácilmente

Hay algunas reglas más para identificar si una palabra lleva tilde o no, pero para este ejercicio solo vamos a considerar las que fueron descritas arriba.

Escribir una función que reciba una frase, donde cada palabra se encuentra separada en sílabas mediante un guión medio “-” y se indica con una barra vertical “|” cuál es la sílaba tónica de la palabra; y retorne la cantidad de palabras que deberían llevar tilde siguiendo las reglas mencionadas anteriormente.

Ejemplos

Frase recibida	Retorna
“ |El ca-|fe |del me-|nu |tie-ne |mal |gus-to“	2
“Da-|mian com-|pro |es-te |ar-bol”	3
""	0
“A-pro-|ba-mos |el e-|xa-men”	0

Se garantiza que la frase recibida está correctamente separada en sílabas utilizando “-” y que todas las palabras contienen un “|” indicando la sílaba tónica.

(*) es muy común verlas mal escritas con tilde