Hoja 8

Importante: Los ficheros deben entregarse a través de web (por **Domjudge** y **Blackboard**). Para cada ejercicio se entregan **tres ficheros**:

Las funciones deben ir en un fichero de nombre:

<<nombre>><<apellidos>><<numE>>func.c

El programa principal (la interfaz) debe ir en un fichero de nombre:

<<nombre>><<apellidos>><<numE>>main.c

El fichero de cabecera (header) será de nombre:

<<nombre>><<apellidos>><<numE>>header.h

donde<<nombre>> es el nombre del alumno, <<apellidos>> los apellidos, y <<numE>> es el número del ejercicio.

La fecha de entrega: consultar la página de la actividad en blackboard

---

Nota 1: En los casos que el programa deba dar un error con ciertos datos de entrada, se deberá mostrar únicamente la palabra **ERROR** en el output.

Nota 2: Cuando se deban imprimir números reales (float) se utilizará el formato `"%.2f"`.

Nota 3: En los ejemplos de salida de cada ejercicio el texto subrayado se ha introducido por el usuario, el resto se ha escrito por el programa.

Ejercicio 35 (2 puntos):

Se pide implementar la función pedir_codigo que se utilizará en el siguiente ejercicio. La función debe pedir 4 números enteros al usuario y guardarlos en el array que recibe como un parámetro. Los números deben estar entre 0 y 9 (ambos incluidos) y sin repeticiones. En caso de que el número introducido esté fuera del rango [0,9] la función debe imprimir “INVALIDO NUM X!” donde x es el índice del número (empezando desde 1), y volver a pedir el número. Por ejemplo, si el segundo número introducido es 10 la función imprimirá “INVALIDO NUM 2!”. En caso de que el número introducido sea un número repetido la función debe imprimir “REPETIDO NUM X!” donde x es el índice del número (empezando desde 1), y volver a pedir el número. Por ejemplo, si el usuario introduce el valor 5 como el primer número y luego introduce otra vez 5 como el tercer número, la función imprimirá “REPETIDO NUM 3!”. Escribir un programa para probar la función. El programa define un array de cuatro enteros, llama a pedir_codigo para rellenar el array, y después imprime el array.

Ejemplo 1:

```
1
5
0
9
1 5 0 9
```

Ejemplo 2:

```
4
10
INVALIDO NUM 2!
3
1
2
4 3 1 2
```

Ejemplo 3:

```
1
3
2
3
REPETIDO NUM 4!
-1
INVALIDO NUM 4!
2
REPETIDO NUM 4!
5
1 3 2 5
```

Ejercicio 36 (3 puntos):

El objetivo de este ejercicio es poder jugar al Master Mind. La versión Master Mind de este ejercicio, llamada U-Master Mind, es una versión más simple de programar pero más difícil de jugar. El juego U-Master Mind se juega entre dos jugadores. El primer jugador debe elegir un código de cuatro números entre 0 y 9 y sin repeticiones y el segundo jugador debe averiguar este código. Cuando empieza el juego, el primer jugador elige cuatro números (sin números repetidos) que será el código que el jugador 2 debe adivinar. En cada intento el jugador 2 introduce un código posible (sin números repetidos) y el programa imprime una respuesta para dar pistas que le ayuden al jugador 2 a deducir el código.

La respuesta consiste en dos cantidades:

- Cantidad de aciertos: la cantidad de dígitos que están en el código secreto y en la misma posición.
- Cantidad de coincidencias: la cantidad de dígitos que están en el código secreto pero no en la misma posición.

Por ejemplo si el código secreto introducido por el jugador 1 es 3 6 1 4 y el jugador 2 propone 1 6 3 5, el programa responderá:

- Cantidad de aciertos: 1
- Cantidad de coincidencias: 2

puesto que el 6 está en el código secreto en la misma posición (el segundo número en el código) y los números 1 y 3 están en el código secreto pero no en las mismas posiciones.

El juego continuará pidiendo códigos de cuatro números hasta que el usuario adivine el código correcto **o hasta 12 intentos**. Al final el programa debe indicar si el jugador ha podido averiguar el código o no.

NOTA: Para almacenar los dígitos del código seleccionado por el jugador 1 y los que introduce el jugador 2 deberá utilizarse un array de enteros y llamar a la función pedir_codigo del ejercicio anterior.

En los ejemplos se ha escrito los cuatros números de cada jugador en una línea para tener más legibilidad.

Ejemplo 1:

```
3 6 1 4
1 6 3 5
a=1 c=2
3 6 4 2
a=2 c=1
3 6 1 4
a=4 c=0
WIN!
```

Ejemplo2:

```
1 2 3 4
0 1 2 3
a=0 c=3
9 1 2 3
a=0 c=3
8 1 2 3
a=0 c=3
7 1 2 3
a=0 c=3
6 1 2 3
a=0 c=3
5 1 2 3
a=0 c=3
4 1 2 3
a=0 c=4
3 1 2 4
a=1 c=3
2 1 3 4
a=2 c=2
1 4 3 2
a=2 c=2
1 3 4 2
a=1 c=3
1 3 2 4
a=2 c=2
GAME OVER!
```

Ejercicio 37:
Escribir las funciones **mezcla_uniforme**, **separar**, **cruce_simple** y el programa principal tal como se explica abajo.

Observación: Las funciones mezcla_uniforme y cruce_simple son versiones cambiadas/simplificadas de los operadores cruce uniforme (también llamado recombinación uniforme) y cruce en un punto de los algoritmos evolutivos, un método de optimización inspirado por evolución biológica (se vera en las asignaturas Algoritmos y Inteligencia Artificial).

- **mezcla_uniforme (1.5 puntos)**: recibe dos cadenas (cadena1 y cadena2), mezcla las cadenas alternando las letras y lo guarda en otra cadena (cadena3) que recibe como parámetro (ver ejemplos de la Figura 1). Se supone que la cadena del primer parámetro no es más larga que la del segundo parámetro. Como se ve en el segundo ejemplo de la Figura 1, en caso de que la primera cadena sea más corta que la segunda, la función continúa con la secuencia restante de la segunda cadena.
  La función devuelve la posición ind en la cadena resultante donde la alternación de letras termina. Este índice se utilizará en la función *separar* para separar las letras de una cadena y guardarlas en dos cadenas.

Primero y segundo parámetro de *mezcla_uniforme*	**Resultado de *mezcla_uniforme***
Cadena1 = “abc” Cadena2 = “xyz”	Cadena3 = “axbycz” ind = 6
Cadena1 = “abcd” Cadena2 = “uvwxyz”	Cadena3 = “aubvcwdxyz” ind = 8

Figura 1

- **separar (1.5 puntos)**: esta función deshace lo que hizo la función mezcla_uniforme. La función recibe una cadena, cadena1, y un entero ind y descompone la cadena y lo guarda en otras dos cadenas, cadena2 y cadena3 que recibe como parámetros (ver ejemplos de la Figura 2). La función usa el entero ind para saber en qué posición termina la alternación de las letras.

Primer parámetro de *Separar*	**Resultado de *Separar***
Cadena1 = “axbycz” ind = 6	Cadena2 = “abc” Cadena3 = “xyz”
Cadena1 = “abcdefghijk” ind = 6	Cadena2 = “ace” Cadena3 = “bdfghijk”
Cadena1 = “abcdefghijk” ind = 0	Cadena2 = “” Cadena3 = “abcdefghijk”

Figura 2

- **cruce_simple (1.5 puntos)**: recibe dos cadenas, cadena1 y cadena2, y los recombina y guarda en otras dos cadenas, cadena3 y cadena4, que también recibe como parámetros. Se supone que la cadena del primer parámetro no es más larga que la del segundo parámetro. Para recombinar las cadenas la función primero debe seleccionar un punto de cruce. El punto se selecciona dividiendo la longitud de la primera cadena entre dos (strlen(cadena1)/2). A continuación la función recombina las dos cadenas cortándolas en el punto de cruce y guarda el resultado de la recombinación en la tercera y cuarta cadena como se muestra en la Figura 3. Se recomienda usar la función **strncpy**.

Parámetros de entrada de *cruce_simple*	**Resultado de *cruce_simple***
Cadena1 = "abc" Cadena2 = "xyz"	Punto_cruce = 1 Cadena3 = "ayz" Cadena4 = "xbc"
Cadena1 = "abcd" Cadena2 = "uvwxyz"	Punto_cruce = 2 Cadena3 = "abwxyz" Cadena4 = "uvcd"

Figura 3

Programa principal (0,5 puntos): Para probar las funciones se pide escribir un programa principal que pida dos cadenas de máximo 20 letras y un carácter como opción y llame a las funciones según la opción elegida como se explica a continuación:

- Si la opción es 'M' el programa llamará a la función **mezcla_uniforme** pasando la cadena más corta como el primer parámetro y la cadena más larga como el segundo. En caso de que las cadenas sean de la misma longitud, la primera cadena se pasará a la función como el primer parámetro. A continuación, el programa principal imprimirá el tercer parámetro de la función y el número entero devuelto por la función.

  Después, el programa llamará a la función **separar** pasando el tercer parámetro de la función anterior y el ind como parámetros para volver a crear las cadenas originales. A continuación el programa imprimirá el resultado de la separación y terminará el programa.
- Si la opción es 'C' el programa llamará a la función **cruze_simple** pasando la cadena más corta como el primer parámetro. En caso de que las cadenas sean del mismo tamaño la primera cadena se pasará a la función como el primer parámetro. A continuación, el programa imprimirá el tercer y cuarto parámetro de la función y terminará.
- Si la opción elegida no es ninguna de las anteriores el programa da un mensaje de error y termina.

Ojo: Tener en cuenta que aunque cada cadena de entrada es de 20 caracteres máximo, las cadenas generadas por las funciones pueden ser más largas.

Ejemplo 1:

```
abc
xyz
M
axbycz 6
abc xyz
```

Ejemplo 2:

```
abcdef
xyz
M
xaybzcdef 6
xyz abcdef
```

Ejemplo 3 (la segunda frase vacía):

```
abc

M
abc 0
 abc
```

Ejemplo 4:

```
abcdef
xyc
C
xbcdef ayc
```

Ejemplo 5:

```
abc
xyz
S
ERROR
```