Hoja 9

Importante: Los ficheros deben entregarse a través de web (por **DomJudge** y **Blackboard)** excepto el fichero del ejercicio 38 que se entrega sólo por Blackboard. Para cada ejercicio se entrega un **fichero** con el nombre:

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

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

---

Instrucciones para generar números aleatorios:

Para obtener números aleatorios se usa las funciones void srand(unsigned int semilla) e int rand(void) de la biblioteca <stdlib.h>.

- Llamamos a la función **srand** sólo una vez al principio del programa para inicializar la **semilla** y llamamos a la función **rand** para producir un número aleatorio.
- La función **rand**() calcula una secuencia de números enteros pseudo-aleatorios en el intervalo de 0 a RAND_MAX (un número enorme, como de 2 mil millones) utilizando la **semilla** introducida por srand o 1 por defecto. Cada vez que llamamos a esta función nos devuelve un entero pseudo-aleatorio de la secuencia calculada.
- Si **srand** se llama con el mismo valor de **semilla**, la secuencia de números pseudo-aleatorios será la misma. Es decir si ejecutamos varias veces nuestro programa, se produce la misma secuencia de números aleatorios. También si **rand** se llama antes de que se haya hecho cualquier llamada a **srand**, la misma secuencia será generada como cuando srand fue llamada la primera vez con un valor semilla de 1.
- Para evitar este problema, se inicializa la semilla con el tiempo de la máquina (la fecha y hora del sistema). Como este valor cambia si ejecutamos el programa en distinto instante de tiempo, no se producirá la misma secuencia de números en cada ejecución del programa.
- Para obtener un número aleatorio en el rango [0,x], podemos llamar a la función **rand** utilizando la operación %: a=rand()%(x+1).
  Para obtener un número aleatorio en el rango [x,y], podemos llamar a la función utilizando la operación %: a=x+rand()%(y+1-x)
  Ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
   int A=1, B=6;
   int i, r;
```

```
    srand((unsigned)time(NULL)); // inicializar la semilla con el tiempo
    for (i=0; i<10; i++){
        r = A+(rand()%(B+1-A));
        printf("%d\n",r);
    }
    return 0;
}
```

Ejercicio 38 (1,5 punto): Escribir un programa que genere aleatoriamente 300 números entre 0 y 9 y lo guardé en un array de 300 enteros. Para generar los números aleatorios utilizar srand() y rand() de la biblioteca <stdlib.h>. Después el programa debe mostrar un histograma de la cantidad de cada número que existe en el array. El histograma tendrá una orientación horizontal, como el que se muestra en la figura. Cada línea de asteriscos (el carácter '*') representa la cantidad de cada número empezando desde 0 hasta 9 (es decir, el número de asteriscos en cada línea es la cantidad del número correspondiente en el array). Por ejemplo, el histograma de la figura muestra que el array contiene 24 veces el número 9 porque en la línea correspondiente al número 9 hay 24 asteriscos.

Nota: para que el programa genere números distintos en cada ejecución inicializar la semilla con el tiempo.

```
0|************************
1|*********************************
2|*******************************
3|***************************
4|************************************
5|******************************
6|**********************************
7|************************
8|***************************
9|************************
-------------------------------------------------
01234567890123456789012345678901234567890123456789

Press any key to continue . . .
```

Ejercicio 39 (2 puntos): Escribir una función llamada ordenar_cadenas que reciba las referencias a TRES punteros a cadenas y las ordena alfabéticamente de menor a mayor. Por ejemplo si tenemos tres punteros a, b, y c apuntando a las cadenas:

a="Maria"
b="Juan"
c="Noel"

Después de llamar a ordenar_cadenas(&a, &b, &c) el resultado será:

a="Juan"
b="Maria"
c="Noel"

Nota: utilizar la función strcmp de <string.h> para las comparaciones (diapositivas 7-8 del PPT 2.13) y la función swap_cadena codificada en los apuntes (diapositivas 11-14 del PPT 2.13). No se permite utilizar los algoritmos de ordenación de internet o libros.

Input:	Output:
Maria Juan Noel	Juan Maria Noel

Ejercicio 40 (2 puntos): Escribir un programa que pregunte al usuario tres números: **inicio** (un entero), **fin** (un entero), y **paso** (un float). El programa primero controla los casos de error de datos de entrada y si hay algún error imprime la palabra "ERROR" y vuelve a pedir los datos. Después de pedir los datos, el programa calcula el promedio de los números desde **inicio** hasta **fin** con pasos de **paso** e imprime el resultado. El programa debe llamar a cuatro funciones como se explican a continuación:

- Función "crear", que recibe los tres números introducidos por el usuario y utiliza malloc de la biblioteca <stdlib.h> para reservar memoria para un array de tamaño necesario de tipo float. La función debe calcular el tamaño necesario para reservar la memoria (la cantidad de números entre **inicio** y **fin** con pasos de **paso**). Por ejemplo, para reservar los números de 10 hasta 12 con pasos de 0.3 se necesita un array del tamaño 7.

- Función "inicializar", que recibe el array reservado en la función anterior y los valores inicio, fin y paso, y guarda en el array todos los números, por ejemplo de 10 hasta 12 con paso de 0.3, es decir, 10, 10.3, 10.6, …, 11.8.

- Función "promedio", que recibe el array y devuelve el promedio de los números guardados en el array.

- Función "liberar", que libera toda la memoria reservada para el array.

El programa main debe imprimir el resultado de la función promedio.

Nota: Está prohibido utilizar variables externas.

Input:	Output:
10 12 0.3	10.90

Input:	Output:
10 4 0.3	ERROR

Ejercicio 41 (2,5 puntos): Implementar el juego del ahorcado. Escribir un programa principal que sólo llama a la función ahorcado pasando la palabra a averiguar como un parámetro (esta función se utilizará después en el ejercicio 43). La función deberá ir pidiendo letras al usuario. En cada iteración, la función primero deberá indicar la longitud de la palabra y mostrar donde se encuentran las letras averiguadas correctamente (ver ejemplo) e indicar el número de vidas que le quedan al usuario. Luego pedirá una letra al usuario. Si la letra introducida está dentro de la palabra deberá dar el mensaje "Has acertado!" . En el caso de que la letra no esté dentro de la palabra la función dará el mensaje "Has fallado!" y el usuario perderá una vida.

El usuario tiene 5 vidas y sólo ganará el juego si consigue adivinar la palabra sin perder todas. En este caso la función dará el mensaje "Has ganado!". En caso de perder todas las vidas, la función dará el mensaje "Has perdido!". Al final, la función escribirá cual era la palabra a adivinar.

NOTA: La palabra a adivinar en este ejercicio es "chewbacca". Esta palabra deberá ser declarada como un constante en el código con #define. Pero el programa debe funcionar con cualquier otra palabra definida previamente.

Ejemplo del juego:

```
Adivinar: ---------
Vidas 5
Letra?
c
Has acertado!
Adivinar: c-----cc-
Vidas 5
Letra?
r
Has fallado!
Adivinar: c-----cc-
Vidas 4
Letra?
h
Has acertado!
Adivinar: ch----cc-
Vidas 4
Letra?
b
Has acertado!
Adivinar: ch--b-cc-
Vidas 4
Letra?
a
Has acertado!
Adivinar: ch--bacca
Vidas 4
Letra?
e
Has acertado!
Adivinar: che-bacca
Vidas 4
Letra?
w
Has ganado!
La palabra: chewbacca
```

Ejercicio opcional 42 (0,5 puntos Extra): Modificar el ejercicio anterior para que el usuario no pueda responder varias veces la misma letra. En caso de introducir una letra repetida el programa deberá volver a pedir una letra sin quitar vidas o dar mensaje de error. Ejemplo del juego:

```
Adivinar: ---------
Vidas 5
Letra?
c
Has acertado!
Adivinar: c-----cc-
Vidas 5
Letra?
c
Adivinar: c-----cc-
Vidas 5
Letra?
y
Has fallado!
Adivinar: c-----cc-
Vidas 4
Letra?
y
Adivinar: c-----cc-
Vidas 4
Letra?
a
Has acertado!
Adivinar: c----acca
Vidas 4
Letra?
```

…

Ejercicio 43 (2 puntos): Implementar una versión avanzada del juego del ahorcado. El programa principal, antes de llamar a la función ahorcado, debe pedir hasta 10 palabras al inicio del juego y almacenarlas en un array de cadenas (ver ejemplo).

El programa permite jugar varias partidas mientras que haya palabras para adivinar. En cada una de las partidas del juego se deberá elegir una de las palabras definidas inicialmente al azar y pasarla como un argumento a la función ahorcado implementado en el ejercicio 41. Después de terminar la función si todavía hay palabras sin adivinar, el programa preguntará al usuario si quiere jugar otra partida.

Ejemplo del juego:

```
Introducir una palabra:
coche
Quieres mas (S/N)?
S
Introducir una palabra:
ahora
Quieres mas (S/N)?
S
Introducir una palabra:
gato
Quieres mas (S/N)?
N
Adivinar: -----
Vidas 5
Letra?
a
Has fallado!
Adivinar: -----
Vidas 4
Letra?
h
Has acertado!
Adivinar: ---h-
Vidas 4
Letra?
o
Has acertado!
Adivinar: -o-h-
Vidas 4
Letra?
c
Has acertado!
Adivinar: coch-
Vidas 4
Letra?
e
Has ganado!
La palabra: coche
Otra Partida (S/N)?
S
Adivinar: ----
Vidas 5
Letra?
h
Has fallado!
Adivinar: ----
Vidas 4
Letra?
w
Has fallado!
Adivinar: ----
Vidas 3
Letra?
e
Has fallado!
Adivinar: ----
Vidas 2
Letra?
r
Has fallado!
Adivinar: ----
Vidas 1
Letra?
q
Has fallado!
Has perdido!
```

```
La palabra: gato
Otra Partida (S/N)?
S
Adivinar: -----
Vidas 5
Letra?
h
Has acertado!
Adivinar: -h---
Vidas 5
Letra?
a
Has acertado!
Adivinar: ah--a
Vidas 5
Letra?
o
Has acertado!
Adivinar: aho-a
Vidas 5
Letra?
r
Has ganado!
La palabra: ahora
```

NOTA 1: Para que funcione bien el código con DomJudge, utilizar scanf con %s para leer cada palabra. Después de cada scanf hay que poner un getchar() para limpiar el '\n' o espacio entre palabras.

NOTA 2: Cada palabra introducida se deberá almacenar de forma temporal en un array de caracteres y a continuación se reservará memoria mediante la función de malloc para almacenarla de forma permanente. Cada vez que el usuario averigüe una palabra, la memoria que ocupa la misma deberá liberarse mediante la función free. Por tanto, habrá que asegurase de que cuando se seleccione una nueva palabra de forma aleatoria, esta no esté liberada. Si todas las cadenas se han liberado, el programa debe terminar.

NOTA 3: Para poder probar este ejercicio en DomJudge es necesario que el programa siempre produzca la misma secuencia de números aleatorios. Por eso se pide inicializar la semilla a valor 10 para que funcione correctamente con el DomJudge.