Hoja 4

Importante: Los ficheros deben entregarse a través de web (por **DomJudge** y **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

Advertencia: No está permitido el uso de la sentencia **for** ni de librerías externas (salvo stdio.h) en ninguno de los ejercicios de esta hoja.

Advertencia 2: 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: 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.

Cosas importantes a la hora de programación, para la claridad y la legibilidad del código:

- El programa debe estar bien comentado.
- Elegir un buen nombre para las variables.
- Sangrar adecuadamente las sentencias de control de flujo

17. (1 punto) Escribir un programa que pida un número entero positivo e imprima todos los números de 0 hasta el número introducido (utilizar **while**). Si el usuario introduce un número negativo el programa debe imprimir un mensaje de error y volver a pedir el número.

    Ejemplo:

    ```
    Un numero positivo:
    5
    0 1 2 3 4 5
    ```

    Ejemplo:

    ```
    Un numero positivo:
    0
    0
    ```

    Ejemplo:

    ```
    Un numero positivo:
    ```

```
-3
ERROR
Un numero positivo:
3
0 1 2 3
```

18. (1 punto) Escribir un programa que lea una serie de números enteros positivos hasta que se introduzca el número 0 y escriba:

    a. el producto de los números introducidos
    b. el total de números pares
    c. el total de números impares.

    El programa no debe aceptar números negativos. En caso de que el usuario introduzca un número negativo el programa dará un mensaje de error y continuará preguntando por números.

    Ejemplo:

```
Un numero positivo (para salir 0):
2
Un numero positivo (para salir 0):
3
Un numero positivo (para salir 0):
-2
ERROR
Un numero positivo (para salir 0):
5
Un numero positivo (para salir 0):
1
Un numero positivo (para salir 0):
0
30 1 3
```

19. (1 punto) Escribir un programa que lea dos números enteros y a continuación imprima un menú para elegir una acción (sumar, restar, multiplicar, dividir, o terminar) como se muestra abajo:

    S: Sumar
    R: Restar
    M: Multiplicar
    D: Dividir
    T: Terminar

    El programa espera al usuario para que elija una opción, y funciona como se indica abajo según la opción elegida:

- Si la opción es ‘S’ o ‘s’, el programa calculará la suma de los dos números, lo imprimirá, y volverá a mostrar el menú.
- Si la opción es ‘R’ o ‘r’, el programa calculará la resta de los dos números, lo imprimirá, y volverá a mostrar el menú.
- Si la opción es ‘M’ o ‘m’, el programa calculará la multiplicación de los dos números, lo imprimirá, y volverá a mostrar el menú.
- Si la opción es ‘D’ o ‘d’, el programa primero comprobará si el segundo número es cero o no. En el caso de que sí, el programa dará un mensaje de ERROR y volverá a mostrar el menú. En el caso contrario, el programa calculará la división del primer número entre el segundo, lo imprimirá con el formato “%.2f”, y volverá a mostrar el menú.
- Si la opción es ‘T’ o ‘t’, el programa imprimirá un mensaje de despedida “Adios.” y se terminará

Nota: Es obligatorio usar do-while en lugar de while. Se puede usar if-else o switch-case para manejar las opciones del menú.

Ejemplo:

```
El primer numero:
2
El segundo numero:
6
S: Sumar
R: Restar
M: Multiplicar
D: Dividir
T: Terminar
r
-4
S: Sumar
R: Restar
M: Multiplicar
D: Dividir
T: Terminar
s
8
S: Sumar
R: Restar
M: Multiplicar
D: Dividir
T: Terminar
D
0.33
S: Sumar
R: Restar
M: Multiplicar
D: Dividir
```

```
T: Terminar
w
ERROR
S: Sumar
R: Restar
M: Multiplicar
D: Dividir
T: Terminar
t
Adios.
```

20. (1,5 Puntos): Escribir un programa que muestre los primeros 10 elementos de una serie numérica en la que cada elemento es $a_{n+3} = a_n + a_{n+1} - a_{n+2}$. Para ello el programa debe leer los tres primeros números de la serie como enteros (a_1, a_2, a_3) y a continuación escribir los primeros 10 elementos incluyendo a_1, a_2 y a_3.

    Ejemplo:

```
Tres numeros:
1 2 9
1 2 9 -6 17 -14 25 -22 33 -30
```

21. **(1,5 Puntos):** Escribir un programa que lea números enteros y calcule sus divisores impares hasta que encuentre un número menor o igual que cero.

    Ejemplo:

```
Numero:
2
1
Numero:
6
1 3
Numero:
27
1 3 9 27
Numero:
60
1 3 5 15
Numero:
36
1 3 9
Numero:
-4
```

22. (1,5 puntos) Escribir un programa que lea el nombre de una persona como una cadena y su edad (un número entero positivo) y escriba su número mágico. Para encontrar el número mágico el programa calcula el número que se obtiene al sumar el código ASCII de los caracteres almacenados en la cadena de nombre. A continuación, el programa lo multiplica por la edad de la persona y suma los dígitos del número resultante. El valor obtenido es el número mágico de la persona. Por ejemplo, si el nombre es "U-TAD" y la edad es 6, la suma de los caracteres por la edad será (85 + 45 + 84 + 65 + 68) × 6 = 2082 y el número mágico será 2 + 0 + 8 + 2 = 12. El programa debe imprimir el número mágico.

    Ejemplo:
    ```
    Edad:
    6
    Palabra:
    U-TAD
    12
    ```

    Ejemplo:
    ```
    Edad:
    -3
    ERROR
    Edad:
    1
    Palabra:
    a
    16
    ```

23. (2,5 puntos) El objetivo de este ejercicio es implementar **el juego de adivinar un número**. El juego se juega entre dos jugadores y hay dos fases claramente diferenciadas:

- El programa deberá pedir al primer jugador un número entero entre 1 y 1000. Este número será el número secreto que deberá adivinar el jugador 2. En caso de que el número introducido no esté entre 0 y 1000 (ambos incluidos) el programa dará un mensaje de ERROR y volverá a pedir al primer jugador un número secreto (utilizando un bucle do-while).
- A continuación, el programa pedirá al segundo jugador que lo adivine. Después de pedirle el número, el programa lo comparará con el número secreto, indicando si el número es mayor o menor que el número secreto. Usando un bucle do-while, el programa seguirá pidiendo al jugador 2 un número y dando pistas hasta que el usuario adivine el número secreto correctamente. El programa debe contar los intentos fallidos y al final debe imprimir el número de intentos.

Ejemplo:

```
Numero secreto:
350
Adivina:
500
Tu numero es superior.
Adivina:
250
Tu numero es inferior.
Adivina:
300
Tu numero es inferior.
Adivina:
400
Tu numero es superior.
Adivina:
350
Acertado.
Intentos: 5
```

Ejemplo:

```
Numero secreto:
10000
Error.
Numero secreto:
1000
Adivina:
500
Tu numero es inferior.
Adivina:
10000
Tu numero es superior.
Adivina:
1000
Acertado.
Intentos: 3
```

Ejercicio opcional 4a (**0,5 puntos** – este ejercicio no se entrega por DomJudge) Modificar el juego anterior para que sea un juego entre el ordenador y un jugador. El programa en lugar de pedir un número secreto al jugador 1, generará un número aleatorio entre 1 y 1000 y pedirá al usuario que lo adivine. El resto del programa será igual que el ejercicio anterior.

Ayuda: para generar números aleatorios ver las instrucciones abajo.

Ejemplo:

```
El numero secreto elegido por el ordenador.
```

```
Adivina:
500
Tu numero es inferior.
Adivina:
1000
Tu numero es superior.
Adivina:
750
Tu numero es inferior.
Adivina:
909
Acertado.
Intentos: 4
```

Ejercicio opcional 4b (0,5 puntos – este ejercicio no se entrega por DomJudge) Modificar el juego de la hoja 3 para jugar entre un usuario y el ordenador. La elección del ordenador se hace generando un número entero aleatorio entre 0 y 5 representado la elección del ordenador entre r, p, t, l, y s.
Ayuda: para generar números aleatorios ver las instrucciones abajo.

Ejemplo:

```
Eleccion jugador 1:
s
El ordenador ha elegido la opcion p.
Ganador el ordenador. El papel refuta a Spock.
```

Instrucciones para generar números aleatorios para los ejercicios de esta hoja y las siguientes hojas:

Para obtener números aleatorios se usan 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 = 0, r;
   srand((unsigned)time(NULL)); // inicializar la semilla con el tiempo
   while (i<10){ // generar 10 numeros aleatorios
          r = A+(rand()%(B+1-A));
          printf("%d\n",r);
          i++;
   }
   return 0;
}
```