LABORATORIO 6
FUNCIONES

1 OBJETIVOS

Al finalizar esta actividad, serás capaz de:

- Utilizar funciones matemáticas predefinidas dentro de cualquier función (no sólo en la función main).
- Entender que de la misma forma en que dentro de un problema se pueden identificar y aislar subproblemas definiendo funciones que los resuelven, dentro de los subproblemas se pueden también identificar y aislar subproblemas más pequeños y se pueden definir funciones que los resuelven.
- Definir nuevas funciones que a su vez llaman a otras funciones (o utilizan otras funciones).

2 MOTIVACIÓN

Cuando un problema es dividido en subproblemas, surge a veces la necesidad o la conveniencia de dividir a su vez los subproblemas en subproblemas más pequeños y van surgiendo distintos niveles de profundidad. Pero puede que un mismo cálculo aparezca en distintos niveles y dentro de distintos subproblemas. Si se dispone de una función independiente que realiza ese cálculo, lo normal sería utilizarlo en cualquier sitio, sin importar el subnivel en el que se esté. Por tanto en vez de limitarse a utilizar funciones sólo en la función main, conviene utilizar funciones dentro de cualquier función.
Generalizando el uso de funciones, en el sentido de que desde cualquier función se puede llamar a otras funciones, se consigue estructurar aún más los programas y hacerlos más entendibles y claros.

2.1 Llamando a funciones desde otras funciones

- No hay límites a la hora de que una función pueda llamar a otras.
- Aunque unas funciones se llamen a otras, todas las funciones se definen de manera independiente: se pondrá el **prototipo** de todas las funciones antes de la definición de la función main y todas las **definiciones** de las funciones se colocarán seguidas después de la función main. El orden en el que se coloquen las funciones no importa.

2.2 Ejercicio ejemplo

2.2.1 Objetivo:

- Mostrar cómo utilizar funciones matemáticas en funciones que no son la función main.
- Mostrar cómo se definen y utilizan las funciones cuando unas llaman a otras.

2.2.2 Enunciado:

Programa que pide al usuario un número entero n mayor o igual que 1 y calcula por una parte los primeros n factoriales y sus raíces cuadradas y por otra parte los primeros n números primos. Al pedir n, se ha de repetir el proceso de petición hasta obener un número >= 1.

Funciones a utilizar:

- *pedir_numero_mayor*: función que, dado un número entero z, pide al usuario que teclee un número mayor que z. El proceso se ha de repetir hasta obtener un número que cumpla dicha condición.

- *primeros_factoriales_con_raices*: función que, dado un número entero z mayor o igual que uno, calcula y muestra por pantalla los factoriales de los números que van desde el 0 hasta el z – 1 y las raíces cuadradas de esos factoriales.

- *calcular_factorial*: función que, dado un número entero x mayor o igual que cero, calcula y devuelve el factorial de x.

- *primeros_primos*: función que, dado un número entero z mayor o igual que uno, calcula y muestra por pantalla los primeros z números primos.

- *es_primo*: función que, dado un número entero x (≥ 1), decide si es primo o no. Si x es primo, la función devolverá un 1 y si no devolverá un 0.

```
#include <stdio.h>
#include <math.h>

/* Datos: Este programa pedirá al usuario un entero n >= 1. */

/* Resultados: Calculará y mostrará por pantalla los primeros n
               factoriales y sus raíces cuadradas y los primeros n
               números primos. */

/* Variables:
          - n: para guardar el número entero que dará el usuario. */

/* Prototipos de las funciones que se utilizarán en el programa. */
int pedir_numero_mayor(int z);
void primeros_factoriales_con_raices (int z);
long calcular_factorial (int x);
void primeros_primos (int z);
int es_primo (int x);

/* Función principal */

void main()
 { /* Inicio de la función main */
   int n;

   n = pedir_numero_mayor(0);/* se pide un número >= 1 */

   primeros_factoriales_con_raices (n);
   primeros_primos (n);

   printf("\nPulsa una tecla para terminar.");
   getch(); /* El ordenador esperará hasta que pulsemos
               cualquier tecla. */
 } /* Fin de la función main */

                        /*********************/

/* Definición de las funciones utilizadas */
```

```
int pedir_numero_mayor(int z)

 /* Esta función pide al usuario un número entero mayor que z
    repitiendo el proceso hasta obtener uno que lo sea. Cuando obtenga
    un número que cumpla esa condición lo devolverá como resultado. */

 { /* Incio de la función */
   int num; /* para guardar el número tecleado por el usuario. */

   do
   { printf("\nIntroduce un numero entero mayor que %d: ", z);
     scanf("%d", &num);
     if (num <= z)
      {
        printf("\nEl numero introducido no es adecuado.");
      }
   } while (num <= z);

   return(num);
 } /* Fin de la función */

                          /*********************/

void primeros_factoriales_con_raices (int z)

 /* Esta función calculará y mostrará los primeros z factoriales y sus
    raíces cuadradas (z >= 1). */

 {
   int num, facto;
   float r;

   printf("\nLos primeros %d factoriales con sus raices cuadradas: ",
          z);

   for (num = 0; num < z; num = num + 1)
    {
      facto = calcular_factorial (num);
      r = sqrt(facto);
      printf("\nEl factorial de %d es %d y la raiz cuadrada de ese "
             "factorial es %f", num, facto, r);
    }
 }

                          /*********************/

long calcular_factorial (int x)

 /* Esta función, dado un número entero x >= 0, calcula y
    devuelve su factorial. */

 { /* Inicio de la función */
   long f;
   int num;

   if (x == 0)
    {
      f = 1;
    }
   else
```

```
    {
      f = 1;
      for (num = 1; num <= x; num = num + 1)
       {
         f = f * num;
       }
    }
   return(f);
} /* Fin de la función */

                        /********************/


void primeros_primos (int z)
 /* Esta función encuentra y muestra los primeros z números primos. */
 {
   int num, cont;

   num = 1; /* Se utilizará para ir pasando los números de uno en uno
               a partir de 1. En cada vuelta del while se analizará si
               el número que está en num es primo o no. */

   cont = 0; /* Esta variable indica en cada momento cuántos números
                primos se han encontrado hasta ese momento. */

   printf("\nLos primeros %d numeros primos son los siguientes: ", z);

   while (cont < z)
    {
      if (es_primo (num) == 1)
       {
         printf("%d  ");
         cont = cont + 1;
       }
      num = num + 1;
    }
 }

                        /********************/

int es_primo (int x)
 /* Esta función devuelve un 1 si x (que ha de ser >= 1) es primo y
    devuelve 0 si no es primo. */

 { /* Inicio de la función */
   int num, cont;

   cont = 0; /* para contar los divisores de x */
   /* num se utilizará para pasar de uno en uno los números que
      van de 1 a x */

   for (num = 1; num <= x; num = num + 1)
    {
      if (x % num == 0)
       {
         cont = cont + 1;
       }
    }

   if (cont == 2)
    {
```

```
      return (1);
    }
   else
    {
      return (0);
    }
} /* Fin de la función */

                        /********************/
```

3 EJERCICIOS

3.1 Ejercicio 1

3.1.1 Objetivo:

El objetivo del ejercicio 1 es utilizar la función matemática predefinida sqrt dentro de una función que no es la función main.

3.1.2 Ayuda para el enunciado 1:

- La raíz cuadrada se calcula utilizando la función matemática predefinida sqrt, es decir, sqrt(x) calcula $\sqrt{x}$. Para poder utilizar esa función hay que poner **#include <math.h>** en la cabecera del programa.
- La función sqrt se utilizará dentro de las funciones *calcular_raiz_positiva* y *calcular_raiz_negativa*.
- Las funciones *calcular_raiz_positiva* y *calcular_raiz_negativa* se utilizarán dentro de la función *calcular_raices*.
- Cuando la ecuación tenga una única raíz, para calcular esa raíz se puede utilizar *calcular_raiz_positiva* o *calcular_raiz_negativa*, es decir, cualquiera de las dos.

3.1.3 Enunciado:

Escribir un programa que pida al usuario los tres coeficientes de una ecuación de segundo grado, calcule y muestre sus raíces (si las tiene) y repita el proceso hasta que el usuario introduzca como primer coeficiente (el de grado 2) el 0.

Una **ecuación de segundo grado** tiene la siguiente forma: $ax^2 + bx + c = 0$, donde a, b y c son los coeficientes y a es el coeficiente de grado 2.

La fórmula general para calcular las raíces es la siguiente: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Una ecuación de segundo grado puede no tener raíces, tener una única raíz o tener dos raíces:

- Si $b^2 - 4ac$ es menor que 0, no tiene raíz.
- Si $b^2 - 4ac$ es 0, tiene una única raíz.
- Si $b^2 - 4ac$ es mayor que 0, tiene dos raíces.

Ejemplos de ecuaciones de segundo grado:

- La ecuación $2x^2 + 1x + 2 = 0$ no tiene raíz.

- La ecuación $2x^2 + 4x + 2 = 0$ tiene una única raíz: -1
- La ecuación $2x^2 + 5x + 2 = 0$ tiene dos raíces: -0.5 y -2

Hay que definir y utilizar las siguientes **funciones**:

- *calcular_raices*: función que, dados los coeficientes a, b y c de una ecuación de segundo grado, procede de la siguiente forma:
- Si la ecuación no tiene raíces, muestra un mensaje indicándolo.
- Si la ecuación tiene una única raíz, la calcula y la muestra e indica que sólo hay una raíz.
- Si la ecuación tiene dos raíces, las calcula y las muestra.

- *calcular_raiz_positiva*: función que, dasdos los coeficientes a, b y c de una ecuación de segundo grado tal que $b^2 - 4ac \geq 0$, calcula y devuelve $\frac{-b+\sqrt{b^2-4ac}}{2a}$

- *calcular_raiz_negativa*: función que, dasdos los coeficientes a, b y c de una ecuación de segundo grado tal que $b^2 - 4ac \geq 0$, calcula y devuelve $\frac{-b-\sqrt{b^2-4ac}}{2a}$

Ejemplo de ejecución: (los datos tecleados por el usuario están en *cursiva y subrayado*)

Pantalla

```
Introduce los coeficientes de una ecuacion de segundo grado: 2, 1, 2
La ecuacion no tiene raices.
Introduce los coeficientes de una ecuacion de segundo grado: 2, 4, 2
La ecuacion tiene una unica raiz: -1
Introduce los coeficientes de una ecuacion de segundo grado: 2, 5, 2
La ecuacion tiene dos raices: -0.5 y -2
Introduce los coeficientes de una ecuacion de segundo grado: 0, 3, 5
Pulsa una tecla para terminar.
```

3.2 Ejercicio1

3.2.1 Objetivo:

El objetivo del ejercicio 2 es definir dos funciones y utilizar una de ellas dentro de la otra. Las dos calcularán un resultado y lo devolverán.

3.2.2 Ayuda para el enunciado 2:

- Para definir la función *mi_pow*, es decir, para indicar cómo se calcula x^y hay que tener en cuenta que $x^y = x * x * x * ...$ (multiplicando x y veces). Esta función sólo servirá para enteros positivos y por tanto no es tan general como la función predefinida pow que sirve incluso para los reales.

- La función *mi_pow* se utilizará dentro de la función *calcular_raiz*.

- Al definir la función *calcular_raiz*, para aproximar $\sqrt[y]{x}$, habrá que ir calculando 0^y, 1^y, 2^y, 3^y, etc. hasta obtener una potencia que es mayor que x. El resultado será el

número utilizado en el caso anterior al que es mayor x. Por ejemplo para calcular $\lfloor \sqrt[3]{85} \rfloor$ se calcularían 0^3, 1^3, 2^3, 3^3, 4^3 y 5^3. Como 5^3 es mayor que 85 el resultado es 4.

3.2.3 Enunciado:

Escribir un programa que pida al usuario dos números enteros m (≥ 1) y n (≥ 1), calcule $\lfloor \sqrt[n]{m} \rfloor$ (aproximación entera por defecto de la raíz n-ésima de m) y a continuación le pregunte al usuario si quiere calcular otra raíz. Si el usuario responde que sí ('s'), el programa ha de volver a pedir dos enteros m (≥ 1) y n (≥ 1) al usuario y calcular la raíz correspondiente. El proceso se repetirá hasta que el usuario responda que no ('n') quiere calcular más raíces. Cada vez que el usuario tenga que responder con una 's' o una 'n', no se admitirá ninguna otra respuesta, repitiendo la pregunta hasta obtener uno de esos dos caracteres. También al pedir m y n habrá que comprobar que son ≥ 1. **No se pueden utilizar las funciones matemáticas predefinidas**.

Ejemplos de aproximaciones enteras por defecto de raíces:

- $\lfloor \sqrt[3]{8} \rfloor = 2$
- $\lfloor \sqrt[3]{10} \rfloor = 2$
- $\lfloor \sqrt[3]{29} \rfloor = 3$

Hay que definir y utilizar las siguientes **funciones**:

- *calcular_raiz*: función que, dados dos números enteros x (≥ 1) e y (≥ 1), calcula y devuelve $\lfloor \sqrt[y]{x} \rfloor$.
- *mi_pow*: función que, dados dos números enteros x (≥ 1) e y (≥ 1), calcula (sin utilizar pow) y devuelve x^y.
- *preguntar_raiz*: función que pregunta al usuario si desea calcular otra raíz o no. El proceso se ha de repetir hasta obtener una 's' o una 'n'.

Ejemplo de ejecución: (los datos tecleados por el usuario están en *cursiva y subrayado*)

Pantalla

```
Introduce dos enteros >= 1: 5, -2
Los datos introducidos no son correctos.
Introduce dos enteros >= 1: 10, 3
La raiz de 10 con respecto a 3 es 2
Quieres calcular otra raiz? (s/n): q
El dato tecleado no es adecuado.
Quieres calcular otra raiz? (s/n): s
Introduce dos enteros >= 1: 29, 3
La raiz de 29 con respecto a 3 es 3
Quieres calcular otra raiz? (s/n): n
Pulsa una tecla para terminar.
```

3.3 Ejercicio 3

3.3.1 Objetivo:

El objetivo del ejercicio 3 es definir tres funciones y utilizar una de ellas dentro de otra. Así se ve que unas funciones serán llamadas desde main y otras serán llamadas desde

funciones que no son la función main. Además una de las funciones ya ha sido definida en otros ejercicios por lo que se puede reutilizar con la ventaja que ello supone.

3.3.2 Ayuda para el enunciado 3:

- Al definir la función *mostrar_triangulo_tartaglia* habrá que ir generando todos los pares posibles de filas y columnas desde la posición (0, 0) hasta la (x – 1, x – 1) y llamar a la función *calcular_combinatorio* con cada par generado.
- La función *calcular_combinatorio* utilizará la función *calcular_factorial*.
- Recordemos que al calcular el factorial de x si éste es 0, el resultado es 1 y en cualquier otro caso hay que obtener el producto de los números que van desde el 1 hasta el x.

3.3.3 Enunciado:

Escribir un programa que pida al usuario un número entero y positivo n (≥ 1), muestre por pantalla el triángulo de Tartáglia correspondiente, vuelva a pedir otro valor n, vuelva a mostrar el triángulo correspondiente y repita el proceso hasta que el usuario teclee un número negativo.

El **triángulo de Tartáglia** para un valor n = 5 tendría 5 filas y 5 columnas y sería el siguiente:

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

Si nos fijamos en las filas y las columnas tenemos lo siguiente:

	c = 0	c = 1	c = 2	c = 3	c = 4
f = 0	1				
f = 1	1	1			
f = 2	1	2	1		
f = 3	1	3	3	1	
f = 4	1	4	6	4	1

El elemento de la fila f y columna c se calcula con la siguiente fórmula (número combinatorio): $\binom{f}{c} = \frac{f!}{c!(f-c)!}$

Hay que definir y utilizar las siguientes **funciones**:

- *mostrar_triangulo_tartaglia*: función que, dado un número entero x (≥ 1), muestra por pantalla el triángulo de Tratáglia de x filas y x columnas.

- *calcular_combinatorio*: función que, dados dos números enteros x e y (tal que x ≥ 0, y ≥ 0, x ≥ y), calcula y devuelve $\binom{x}{y}$.

- *calcular_factorial*: función que, dado un número entero x mayor o igual que cero, calcula y devuelve el factorial de x.

Ejemplo de ejecución: (los datos tecleados por el usuario están en *cursiva y subrayado*)

Pantalla

```
Introduce un numero entero >= 1: 2
Triangulo de Tartaglia correspondiente:
1
1 1
1 2 1
Introduce un numero entero >= 1: 4
Triangulo de Tartaglia correspondiente:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
Introduce un numero entero >= 1: -9
Pulsa una tecla para terminar.
```

3.4 Ejercicio 4

3.4.1 Objetivo:

El objetivo del ejercicio 4 es definir y utilizar dos funciones nuevas y utilizar una de ellas dentro de otra, aprendiendo a estructurar programas.

3.4.2 Ayuda para el enunciado 4:

- Al definir la función *mostrar_logaritmos* con respecto a los parámetros x e y, habrá que pasar las bases de los logaritmos de uno en uno a partir de 2 y hasta llegar a x. Para cada base habrá que llamar a la función *calcular_logaritmo* pasándole como argumentos la base e y.

- Al definir la función *calcular_logaritmo* con respecto a los parámetros x e y, habrá que ir multiplicando x por sí mismo (x * x * x * ...) hasta obtener un valor mayor que y. El número de veces que se haya multiplicado menos 1 será el resultado que ha de devolver la función.

3.4.3 Enunciado:

Escribir un programa que pida al usuario dos números enteros m y n (tal que m ≥ 1, n ≥ 1 y n ≥ m) calcule y muestre los elementos $\lfloor \log_2 n \rfloor$, $\lfloor \log_3 n \rfloor$, $\lfloor \log_4 n \rfloor$, ..., $\lfloor \log_m n \rfloor$ (aproximaciones enteras por defecto de los correspondientes logaritmos) y la suma de todos ellos y a continuación le pregunte al usuario si quiere repetir el cálculo para otros dos valores. Si el usuario responde que sí ('s'), el programa ha de volver a pedir dos enteros m y n al usuario y calcular los logaritmos y la suma correspondientes. El proceso se repetirá hasta que el usuario responda que no ('n') quiere calcular más logaritmos. Cada vez que el usuario tenga que responder con una 's' o una 'n', no se admitirá ninguna otra respuesta, repitiendo la pregunta hasta obtener uno de esos dos caracteres. También al pedir m y n habrá que comprobar que son ≥ 1 y que n ≥ m. **No se pueden utilizar las funciones matemáticas predefinidas**.

Para calcular la **aproximación entera por defecto de un logaritmo**, $\lfloor \log_m n \rfloor$, hay que calcular cuántas veces es necesario multiplicar m por sí mismo (m * m * m * …) para obtener un valor mayor que n. Ese número de veces menos 1 será el resultdo:

Ejemplos de aproximaciones enteras por defecto de logaritmos:

- $\lfloor \log_2 8 \rfloor = 3$ porque para obtener un valor mayor que 8 hay que multiplicar 2 por sí mismo 4 veces (2 * 2 * 2 * 2) y por tanto el resultado es 4 – 1, es decir, 3.
- $\lfloor \log_2 12 \rfloor = 3$ porque para obtener un valor mayor que 12 hay que multiplicar 2 por sí mismo 4 veces (2 * 2 * 2 * 2) y por tanto el resultado es 4 – 1, es decir, 3.
- $\lfloor \log_4 20 \rfloor = 2$ porque para obtener un valor mayor que 20 hay que multiplicar 4 por sí mismo 3 veces (4 * 4 * 4) y por tanto el resultado es 3 – 1, es decir, 2.

Hay que definir y utilizar las siguientes **funciones**:

- *mostrar_logaritmos*: función que, dados dos números enteros x e y (tal que x ≥ 1, y ≥ 1 e y ≥ x), calcula y muestra $\lfloor \log_2 y \rfloor$, $\lfloor \log_3 y \rfloor$, $\lfloor \log_4 y \rfloor$, ..., $\lfloor \log_x y \rfloor$ y la suma de todos ellos.

- *calcular_logaritmo*: función que, dados dos números enteros x e y (tal que x ≥ 1, y ≥ 1 e y ≥ x), calcula y devuelve $\lfloor \log_x y \rfloor$.

- *preguntar_logaritmo*: función que pregunta al usuario si desea calcular otra secuencia de logaritmos o no. El proceso de recoger la respuesta se ha de repetir hasta obtener una 's' o una 'n'.

Ejemplo de ejecución:(los datos tecleados por el usuario están en *cursiva y subrayado*)

Pantalla

```
Introduce dos enteros >= 1: 5, -2
Los datos introducidos no son correctos.
Introduce dos enteros >= 1: 4, 20
log(2)20 = 4  log(3)20 = 2  log(4)20 = 2
La suma es 8.
Quieres calcular otros logaritmos? (s/n): q
El dato tecleado no es adecuado.
Quieres calcular otros logaritmos? (s/n): s
Introduce dos enteros >= 1: 3, 50
log(2)50 = 5  log(3)50 = 3
La suma es 8.
Quieres calcular otros logaritmos? (s/n): n
Pulsa una tecla para terminar.
```

3.5 Ejercicio 5

3.5.1 Objetivo:

El objetivo del ejercicio 5 es definir una función nueva y reutilizar otra definida con anterioridad utilizándola además dentro de la primera.

3.5.2 Ayuda para el enunciado 5:

- En la función main se pedirá un número n y se le pasará a la función *decidir_fibonacci* y se mostrará un mensaje teniendo en cuenta lo que devuelva esa

función. A continuación desde main se pedirá otro número, se le pasará a esa función etc hasta que el usuario dé un número negativo.

- En la función *decidir_fibonacci*, se irán calculando los Fibonaccis de 0, de 1, de 2, etc hasta dar con un número cuyo Fibonacci sea x o hasta encontrar un número cuyo Fibonacci sea mayor que x. En el primer caso existe un número cuyo Fibonacci es x, en el segundo caso no. Para calcular los Fibonaccis habrá que llamar a la función *calcular_fibonacci*.

3.5.3 Enunciado:

Escribir un programa que pida al usuario un número entero y positivo n (≥ 0) decida si existe algún número tal que su Fibonacci sea n y repita el mismo proceso hasta que el usuario introduzca un número negativo.

El **Fibonacci** de un número entero (≥ 1) se define de la siguiente forma:

Fibonacci(0) = 0
Fibonacci(1) = 1
Fibonacci(x) = Fibonacci(x – 2) + Fibonacci(x – 1)

Ejemplos de Fibonaccis:

x	0	1	2	3	4	5	6	7	8	9	10	11	…
Fibonacci(x)	0	1	1	2	3	5	8	13	21	34	55	89	…

Hay que definir y utilizar las siguientes **funciones**:

- *decidir_fibonacci*: función que, dado un número entero x (≥ 0), decide si x es Fibonacci de algún número. En caso de que no lo sea, ha de devolver un -1 y en caso de que sí sea Fibonacci de algún número ha de devolver ese número.

- *calcular_fibonacci*: función que, dado un número entero x (≥ 0), calcula y devuelve el Fibonacci de x.

Ejemplo de ejecución:(los datos tecleados por el usuario están en *cursiva y subrayado*)

Pantalla

```
Introduce un numero entero >= 1: 4
4 no es Fibonacci de ningún número.
Introduce un numero entero >= 1: 8
8 es el Fibonacci de 6.
Introduce un numero entero >= 1: -5
Pulsa una tecla para terminar.
```