LABORATORIO 5
FUNCIONES

1 OBJETIVOS

Al finalizar esta actividad, serás capaz de:

- Utilizar en la función main funciones matemáticas predefinidas.
- Entender que es posible aislar subtareas o subcálculos dentro de un programa y entender que es posible definir funciones nuevas que realizan esas subtareas o subcálculos.
- Definir nuevas funciones y utilizarlas dentro de la función main.

2 MOTIVACIÓN

Muchas veces un mismo subcálculo (relativamente) complejo es realizado varias veces dentro de un programa o aparece en distintos programas. El aislar ese subcálculo y definir una función que lo resuelve hace posible que no haya que escribir todas sus instrucciones cada vez que se que vaya a realizar el subcálculo, ya que bastará con escribir la función una vez y llamarla todas las veces que sea necesario.
El aislar subproblemas y resolverlos de manera independiente, definiendo funciones, posibilita estructurar los programas haciendo que éstos sean más claros y fáciles de seguir.

2.1 Las funciones en C

- Por una parte C dispone de funciones predefinidas. Por ejemplo, funciones matemáticas que sirven para calcular el valor absoluto, **fabs(x)**, la raíz cuadrada, **sqrt(x)** y la potencia, **pow (x, y)**. Para utilizar estas funciones es suficiente con añadir **#include<math.h>** en la cabecera del programa.
- Por otra parte el programador puede definir **funciones nuevas** y utilizarlas. Al definir una función hay que indicar cuál es el **tipo del resultado**, el **nombre de la función** y el **tipo y nombre de los argumentos**.
- **Si la función no devuelve ningún resultado y/o no tiene argumentos** habrá que poner la palabra **void** en el lugar correspondiente al tipo del resultado y/o a los tipos de los argumentos.
- Para utilizar funciones definidas por el programador, una opción es poner el **prototipo** de cada función antes de la definición de la función main, y poner las **definiciones** de las funciones todas seguidas después de la función main.
- Por tanto, **los programas constarán de varias funciones**. **La función main estará siempre** y además estarán las funciones que definamos.
- **El ordenador ejecuta siempre la función main** y si desde la función main se le llama a otra función entonces se ejecutarán las instrucciones correspondientes a esa función pero teniendo en cuenta los argumentos que se le han pasado en la llamada.
- **Una función puede realizar distintos tipos de tareas:**
- Realizar cáculos y devolver un resultado.
- Realizar cálculos y mostrar resultados por pantalla.
- Pedir datos al usuario.
- Mostrar resultados.
- Llamar a otras funciones para que éstas a su vez realicen subcálculos.

2.2 Ejercicio ejemplo

2.2.1 Objetivo:

- Mostrar cómo se utilizan las funciones matemáticas en la función main.
- Mostrar cómo se definen funciones nuevas y cómo se utilizan en la función main.

2.2.2 Enunciado:

Programa que pide al usuario dos números enteros mayores o iguales que 0 y calcula el factorial de cada uno de ellos, los muestra y, a continuación, si la raíz cuadrada del primer factorial partido por el segundo es mayor o igual que 1, calcula y muestra el primer factorial elevado al segundo y si no, el segundo elevado al primero. Al pedir los datos iniciales, se ha de repetir el proceso de petición hasta obener dos números que sean >= 0.

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.

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

- *mostrar_factorial*: función que, dados un número entero x (≥ 0) y su factorial f, muestra un mensaje por pantalla indicando que el factorial de x es f.

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

/* Datos: Este programa pedirá al usuario dos enteros n1 y
          n2 que sean >= 0. */

/* Resultados: Calculará y mostrará por pantalla los factoriales de n1
               y n2 y además si la raíz cuadrada del primer factorial
               partido por el segundo factorial es mayor o igual que
               1, calculará y mostrará el primero de esos factoriales
               elevado al segundo y en caso contrario, el segundo
               elevado al primero. */

/* Variables:
          - n1, n2: para guardar los dos enteros que dará el usuario.
          - f1, f2: para guardar los factoriales de n1 y n2. */

/* Prototipos de las funciones que se utilizarán en el programa. */

int pedir_numero_mayor(int z);
long calcular_factorial (int x);
void mostrar_factorial (int x, long f);

/* Función principal */

void main()
 { /* Inicio de la función main */
   int n1, n2;
   long f1, f2, p;

   n1 = pedir_numero_mayor(-1);/* se pide un número >= 0 */
   n2 = pedir_numero_mayor(-1);

   f1 = calcular_factorial (n1);
   f2 = calcular_factorial (n2);

   mostrar_factorial (n1, f1);
```

```
   mostrar_factorial (n2, f2);

   if (sqrt(f1/f2) >= 1)
     {
       p = pow(f1, f2);
       printf("\nEl primer factorial elevado al segundo es %ld", p);
     }
   else
     {
       p = pow(f2, f1);
       printf("\nEl segundo factorial elevado al primero es %ld", p);
     }

   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 */

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

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 mostrar_factorial (int x, long f)
 /* Esta función muestra un mensaje indicando que f es el
    factorial de x. */

 { /* Inicio de la función */

   printf("\nEl factorial de %d es %ld.", x, f);

 } /* 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 pow dentro de la función main.

3.1.2 Ayuda para el enunciado 1:

Recordemos que la función matemática predefinida pow calcula la potencia, es decir, **pow(x, y**) calcula $\mathbf{x^y}$. Para poder utilizar esa función hay que poner **#include <math.h>** en la cabecera del programa.

3.1.3 Enunciado:

Escribir un programa que vaya pidiendo números enteros de dos en dos y calculando y mostrando el resultado que se obtiene al elevar el primero por el segundo en cada caso. El proceso terminará cuando alguno de los números introducidos sea menor o igual que 0.

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

```
Introduce dos enteros (>= 1) separados por una coma: 3, 2
3 elevado a 2 es 9
Introduce dos enteros (>= 1) separados por una coma: 2, 10
2 elevado a 10 es 1024
Introduce dos enteros (>= 1) separados por una coma: -5, 30
Pulsa una tecla para terminar.
```

3.2 Ejercicio 2

3.2.1 Objetivo:

El objetivo del ejercicio 2 es definir tres funciones y utilizarlas en la función main:

- La primera pedirá al usuario un dato teniendo en cuenta el argumento que se le pasa.
- La segunda realizará un cálculo a partir del argumento que se le pasa y devolverá un resultado para que sea utilizado en la función main.
- La tercera mostrará un mensaje por pantalla teniendo en cuenta los argumentos que se le pasarán.

3.2.2 Ayuda para el enunciado 2:

- Para localizar los primeros n números perfectos habrá que empezar a pasar los números de uno en uno a partir del 1 e ir comprobando en cada caso si es perfecto o no, llamando para ello a la función *es_perfecto*. Se continuará hasta haber encontrado n números perfectos.

- Para decidir si un número es perfecto habrá que partir desde 1 e ir pasando los números de uno en uno e ir calculando la suma de los divisores. Si al final la suma es igual al número dado, el número es perfecto y si no, no es perfecto.

3.2.3 Enunciado:

Escribir un programa que pida al usuario un número entero y positivo n (≥ 1) y calcule los primeros n **números perfectos**. A la hora de pedir el valor n habrá que repetir el proceso de petición hasta obtener un número que cumpla el requisito mencionado.

Un **número** entero n que sea mayor o igual que 1 es **perfecto** si la suma de sus divisores (sin tener en cuenta el propio n) suma n.

Ejemplos de números perfectos y no perfectos:
- 6 es perfecto porque 1 + 2 + 3 = 6
- 28 es perfecto porque 1 + 2 + 4 + 7 + 14 = 28
- 12 no es perfecto porque 1 + 2 + 3 + 4 + 6 $\neq$ 12

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

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

- *mostrar_perfecto*: función que, dados un número entero x (≥ 1) y el valor p (que ha de ser 1 ó 0), si p es 1 muestra un mensaje por pantalla indicando que x es perfecto y si p es 0, muestra un mensaje por pantalla indicando que x no es perfecto.

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

Pantalla

```
Introduce un numero entero >= 1: -4
El dato introducido no es correcto.
Introduce un numero entero >= 1: 2
6 es perfecto.
28 es perfecto.
Pulsa una tecla para terminar.
```

3.3 Ejercicio 3

3.3.1 Objetivo:

El objetivo del ejercicio 3 es volver a utilizar las mismas tres funciones que en el ejercicio anterior para resolver un problema distinto pero que comparte algunos subproblemas con el anterior problema. Esto sirve para ver que las funciones son muy útiles para aislar subprogramas y luego poder utilizarlos en distintos ejercicios sin necesidad de retocar las funciones. Cambiará la función main pero las otras tres funciones son las mismas:

3.3.2 Ayuda para el enunciado 3:

Para localizar números perfectos menores que n habrá que empezar a pasar los números de uno en uno a partir del 1 e ir comprobando en cada caso si es perfecto o no, llamando para ello a la función *es_perfecto*. Se continuará hasta llegar al número n.

3.3.3 Enunciado:

Escribir un programa que pida al usuario un número entero y positivo n (≥ 1) y calcule los números perfectos menores que n. A la hora de obtener el valor n habrá que repetir el proceso de petición hasta obtener un número que cumpla el requisito mencionado.

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

- *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.

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

- *mostrar_perfecto*: función que, dados un número entero x (≥ 1) y el valor p (que ha de ser 1 ó 0), si p es 1 muestra un mensaje por pantalla indicando que x es perfecto y si p es 0, muestra un mensaje por pantalla indicando que x no es perfecto.

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

Pantalla

```
Introduce un numero entero >= 1: -4
El dato introducido no es correcto.
Introduce un numero entero >= 1: 40
6 es perfecto.
28 es perfecto.
Pulsa una tecla para terminar.
```

3.4 Ejercicio 4

3.4.1 Objetivo:

El objetivo del ejercicio 4 es definir dos funciones nuevas y utilizarlas en la función main.

3.4.2 Ayuda para el enunciado 4:

Para decidir si un número es medio o no, primero se ha de calcular la suma de sus predecesores (a partir de 1) y a continuación se ha de ir probando con las sumas de sus sucesores contiguos, hasta encontrar una suma que coincide con la de los predecesores o hasta obtener una suma que es mayor que la suma de los predecesores. Si se da este último caso, se sabe que ninguna otra suma coincidirá con la suma de los predecesores (porque las sumas de los sucesores son cada vez mayores).

3.4.3 Enunciado:

Escribir un programa que muestre al usuario el primer **número medio** y a continuación le pregunte si quiere el siguiente. Si el usuario responde que sí ('s'), el programa ha de calcular y mostrar el siguiente número medio y ha de volver a preguntar al usuario si quiere el siguiente. El proceso se repetirá hasta que el usuario responda que no ('n'). 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.

Un **número** entero n que sea mayor o igual que 1 es **medio** si la suma de sus predecesores (1 + 2 + 3 + … + n − 1) se puede obtener también sumando algunos números contiguos que le siguen:

Ejemplos de números medios y no medios:

- 6 es medio porque 1 + 2 + 3 + 4 + 5 = 15 y 7 + 8 = 15. Es decir, la suma de los predecesores de 6 es obtenible sumando algunos sucesores contiguos de 6.
- 35 es medio porque 1 + 2 +...+ 34 = 595 y 36 + 37 +…+ 49 = 595 Es decir, la suma de los predecesores de 35 es obtenible sumando algunos sucesores contiguos de 35.
- 7 no es medio porque 1 + 2 + 3 + 4 + 5 + 6 = 21 y no se puede obtener el valor 21 sumando números contiguos a partir de 8: $8 \neq 21$, $8 + 9 \neq 21$, $8 + 9 + 10 \neq 21$, etc. Además 8 + 9 + 10 es mayor que 21 y por tanto ya se sabe que ninguna otra suma superior dará 21.

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

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

- *preguntar_otro_medio*: función que pregunta al usuario si desea otro número medio 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*)

```
El primer numero medio es: 6
Quieres el siguiente numero medio?(s/n): q
El dato introducido no es adecuado.
Quieres el siguiente numero medio?(s/n): s
El siguiente numero medio es: 35
Quieres el siguiente numero medio?(s/n): n
Pulsa una tecla para terminar.
```

Pantalla

3.5 Ejercicio 5

3.5.1 Objetivo:

El objetivo del ejercicio 5 es definir una función nueva y volver a utilizar una función que ya se ha definido con anterioridad. Esto sirve para ver que las funciones son muy útiles para aislar subprogramas y luego poder utilizarlos en distintos ejercicios sin necesidad de retocar las funciones.

3.5.2 Ayuda para el enunciado 5:

- En la función main para obtener los primeros n términos de la serie de Fibonacci habrá que ir pasando los números de uno en uno a partir del 0 y calcular el Fibonacci de cada uno de ellos llamando a la función *calcular_fibonacci*.

- Al definir la función *calcular_fibonacci*, como para obtener el Fibonacci de un número es necesario sumar los Fibonaccis de los dos anteriores, para calcular el Fibonacci de x hay que partir de los Fibonaccis de 0 y 1 (que se consideran básicos) e ir calculando los Fibonaccis de todos los números hasta llegar a tener el Fibonacci de x. Por tanto, aunque al final esta función sólo devolverá el Fibonacci de x, en el proceso se habrán tenido que calcular los Fibonnacis de todos los números menores que x.

3.5.3 Enunciado:

Escribir un programa que pida al usuario un número entero y positivo n (≥ 1) y calcule los primeros n términos de la **serie de Fibonacci**. Como el primer término de la serie es el Fibonacci de 0, el último Fibonacci a mostrar será el de n – 1. A la hora de pedir el valor n habrá que repetir el proceso hasta obtener un número que cumpla la condición requerida.

El **Fibonacci** de un número entero (≥ 0) 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**:

- *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.

- *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
El dato introducido no es correcto.
Introduce un numero entero >= 1: 4
Fibonacci(0) = 0
Fibonacci(1) = 1
Fibonacci(2) = 1
Fibonacci(3) = 2
Pulsa una tecla para terminar.
```