TEMA 5
FUNCIONES

1 PROGRAMAS CON FUNCIONES

Hasta ahora en los programas sólo teníamos una función: la función *main* o la función principal. En esa función se indicaba todo lo que tenía que hacer el programa, pero a la hora de resolver un problema, en general es mejor y más sencillo dividirlo en subproblemas y resolver cada subproblema por su lado.
En este tema se abordará la división de problemas en subproblemas y la definición de funciones que resuelven esos subproblemas. Se definirá una función por cada subproblema.

1.1 Ventajas de utilizar funciones

- Cuando en un mismo programa hay que resolver un subproblema determinado varias veces, se definirá una función que resuelva dicho subproblema y se utilizará tantas veces como haga falta sin tener que indicar cada vez todas las intrucciones que resuelven el subprograma.

- Los programas son más claros. Si se divide el problema en subproblemas y se define una función para resolver cada subproblema, haciendo uso de las funciones definidas la función main quedará como si estuviera escrito casi en lenguaje natural.

- Es más fácil realizar modificaciones en los programas porque normalmente esas modificaciones afectarán a una función o a unas pocas funciones por lo que será más fácil controlar los cambios y además sin necesidad de retocar todo, sólo las funciones afectadas.

- Si un mismo subproblema aparece en distintos problemas, se podrá definir una función que resuelva ese subproblema y utilizarlo en distintos programas.

1.2 Nuevo formato de los programas

En los programas escritos hasta ahora sólo teníamos una función, la función main. Pero a partir de ahora escribiremos más de una función en los programas. La función main aparecerá siempre pero habrá también otras funciones. Por tanto, un programa puede ser visto como un conjunto de funciones.

1.3 Ejecución de programas

Tal como se ha dicho en el apartado anterior, a partir de ahora los programas constarán de varias funciones: la función main y otras funciones adicionales.
En el caso de los programas que sólo contienen la función main, el ordenador se limita a ejecutar las intrucciones que conforman esa función. Pero ¿qué ocurre en el caso de programas que contienen varias funciones? El ordenador sigue lo especificado en la función main y cuando desde la función main se llama a otra función, se ejecutará la otra función sustituyendo sus parámetros reales por los argumentos dados al realizar la llamada.

1.4 Conceptos relacionados con las funciones

Al trabajar con funciones hay que diferenciar tres conceptos:

- **Prototipo** de la función: los prototipos de las funciones utilizadas en un programa hay que colocarlos antes de la función main, justo después de los includes y las definiciones de las constantes (si los hay).
- **Definición** de la función: las definiciones de las funciones se colocan justo después de la definición de la función main.
- **Utilización** de la función: una función puede ser utilizada tanto en la función main como en cualquier otra función.

También es posible definir todas las otras funciones antes de la función main. En ese caso no es necesario poner los prototipos de las funciones.

1.5 Esquema general de las funciones

Al definir una función hay que tener en cuenta cuatro aspectos:

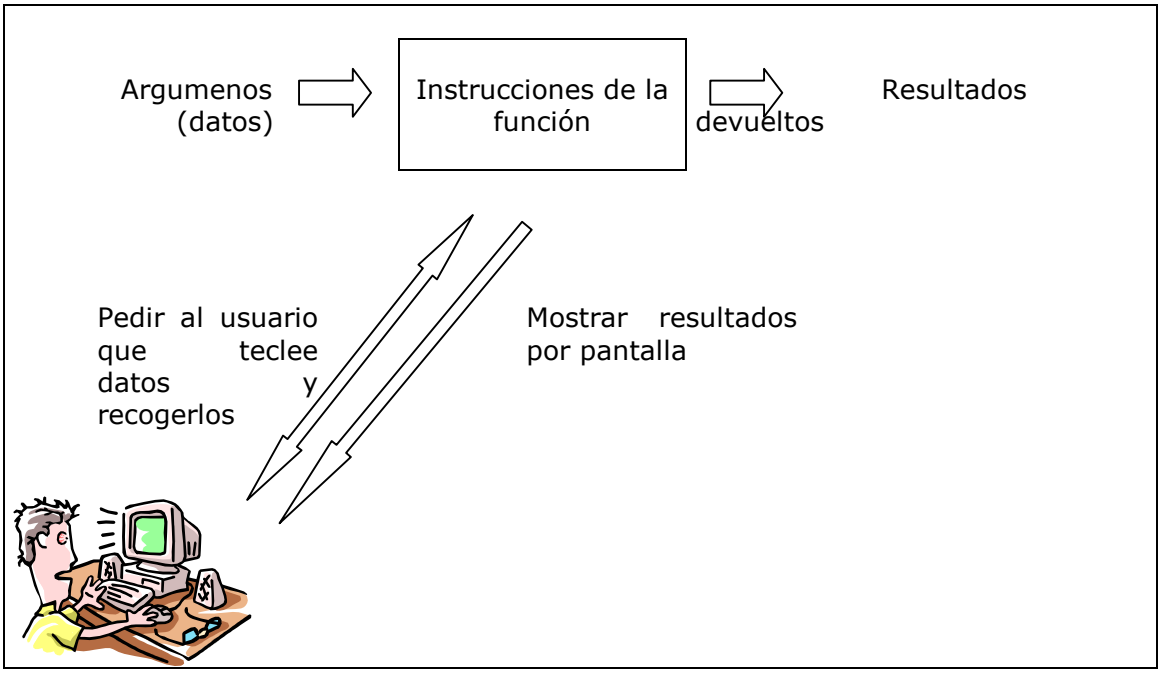
- Argumentos (o **datos**) de la función.
- **Resultados** de la función.
- **Si** la función **ha de pedir al usuario que teclee algo** o no.
- **Si** la función **ha de mostrar algo por pantalla**.

A partir de ahora en vez de utilizar los conceptos de "parámetro formal" y "parámetro real" se utilizará normalmente el concepto de "argumento" para englobar ambos conceptos:

- Los parámetros formalas son los nombres que se ponen entre paréntesis en el prototipo y en la definición de la función. Pueden ser datos o resultados.

- Los parámetros reales son los valores (variables, constantes, espresiones, ...) que se ponen en lugar de los párametros formales al llamar a la función.

A continuación se muestra el esquema general de una función. En dicho esquema aparecen los datos y los resultados de la función y el intercambio de información que puede llevar a cabo con el usuario. Al ver los distintos tipos de funciones se adaptará el esquema a las distintas características:



1.6 Funciones predefinidas y funciones definidas por nosotros

Ya sabemos utilizar algunas funciones: printf, scanf, system, getch, …
Esas funciones están predefinidas en el lenguaje C y por tanto no hace falta que las definamos, las podemos utilizar directamente. Para utilizarlas tenemos que indicar mediante una instrucción **include**, en qué fichero están definidas. Esos ficheros tienen siempre la **extensión h** (stdio.h, conio.h, …).
En C hay muchas funciones predefinidas pero a continuación mencionaremos las que vamos a utilizar nosotros, indicando en cada caso en qué fichero de extensión h se encuentran:

1.6.1 Funciones matemáticas

Todas las funciones matemáticas que vamos a utilizar están en el fichero math.h y por consiguiente, siempre que queramos utilizar alguna de ellas, habrá que poner **#include < math.h >** en la cabecera del programa.
Para nosotros las funciones más relevantes serán fabs, sqrt y pow:

- **fabs**
  fabs(x): devuelve el valor absoluto de x, es decir, calcula |x|.

  Ejemplo:

```
int num = 372, cont = -40, a, b;
a = fabs(num);   /* se le asignará el valor 372 a la variable a */
b = fabs(cont);   /* se le asignará 40 a b */
```

- **sqrt**
  sqrt(x): devuelve la raíz cuadrada de x, es decir, calcula $\sqrt{x}$.

  Si x es negativo, se producirá un error. Tendremos que encargarnos de que no se le pase nunca un valor negativo.

  El resultado devuelto por la función sqrt será real y, por tanto, convendrá guardarlo en una variable de tipo float o double. Si se guarda en una variable de tipo int o long, se perderá la parte no entera.

  Ejemplo:

```
int a = 49, b, c;
float r = 23, s;
b = sqrt(a);   /* a b se le asignará el valor 7 */
c = sqrt(r);   /* a c se le asignará el valor 4 */
s = sqrt(r);   /* a s se le asignará el valor 4.795832 */
```

  Ejemplo:

  La expresión $\sqrt{\sqrt{\frac{341}{4}}+10}$ se representaría como sqrt(sqrt(341 / 4) + 10) en C.

  Ejemplo:

```
float r = 23, s, t;
s = sqrt((r * 3) + 5);   /* a s se le asignará 8.602325 */
t = sqrt(sqrt(341 / 4) + 10);   /* a t se le asignará 4.384010 */
```

Ejemplo:

```
float s, t;
s = sqrt(-200);   /* Error */
t = sqrt(sqrt(30 - 40);   /* Error */
```

Como no se puede calcular la raíz cuadrada para números negativos, esas dos asignaciones generarían un error.

Ejemplo:
Supongamos que queremos asignar a s la raíz cuadrada del valor absoluto de r:

$$\sqrt{|r|}$$

```
float r = -200, s;
s = sqrt(fabs(-200));   /* s tomará el valor 14.142136 */
```

Ejemplo:
Ahora se quiere asignar a s la raíz de r si r no es negativo y -1 en caso contrario:

Si r ≥ 0 asignar $\sqrt{r}$ a s y si no, asignar -1 a s.

```
float r, s;

if (r >= 0)
 {
   s = sqrt(r);
 }
else
 {
   s = -1;
 }
```

- **pow**
pow(x, y): devuelve x elevado a y, es decir, x^y.

En particular también sirve para calcular expresiones de la forma $\sqrt[n]{x^p}$ porque $\sqrt[n]{x^p} = x^{p/n}$.

Ejemplo:
Supongamos que queremos calcular las siguientes expresiones: 5^7, 23^{-7}, $(12*3)^{2-3}$, $\sqrt[4]{20.07}$, $\sqrt[6]{4^3}$.

En el lenguaje C habría que escribirlos de la siguiente forma:

```
long a;
      double r, s, t, v;
      a = pow(5, 7);
      r = pow(23, -7);
      s = pow((12 * 3), (2 - 3));
      t = pow(20.07, (1.0 / 4));
      v = pow(4, (3.0 / 6));
```

En este ejemplo se han utilizado variables de tipo long y double porque con la potencia se superan fácilmente los límites de los tipos int y float.

La función pow(x, y) genera un error en los siguientes casos:

- ✓ Cuando x = 0 e y ≤ 0.
- ✓ Cuando siendo x negativo, y no es entero.

Ejemplo:
A continuación se escribirá un programa que calcula la siguiente fórmula:

$$\sqrt[7]{\frac{(((2*x)/y)-y^4)^2}{(|x+y|)^5}}+x$$

Esa fórmula no es calculable (porque se produciría un error) en los siguientes casos:

- ✓ Cuando y vale 0.
- ✓ Cuando x + y es 0.

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

/* Datos: Este programa pedirá al usuario dos números reales. */

/* Resultados: Si la fórmula es calculable con los datos introducidos, se
               calculará y mostrará el resultado y si no es calculable se
               mostrará un mensaje indicándolo. */

/* Variables:
          - x, y: para guardar los dos valores que tecleará el usuario.
          - res: para guardar el resultado en caso de que la fórmula sea
                 calculable. */

void main()
{
  float x, y, res;

  printf("\nIntroduce dos números reales separados por una coma: ");
  printf("%f, %f", &x, &y);

  if ((y == 0)
   {
     printf("\nLa fórmula no es calculable porque el valor de y es 0.");
   }
  else if (x + y == 0)
   {
     printf("\nLa fórmula no es calculable porque el valor de x + y es 0.");
   }
  else
   {
     res = pow(pow(((2 * x) / y) - pow(y, 4), 2) / pow(fabs(x + y), 5),
               (1.0 / 7)) + x;
     printf("\nEl valor de la fórmula es %f.", res);
   }

  printf("\nPulsa una tecla para terminar.");
  getch(); /* El ordenador esperará hasta que el usuario pulse cualquier
              tecla. */
}
```

Además de las tres funciones matemáticas que se acaban de mencionar (fabs, sqrt eta pow), existen otras funciones matemáticas predefinidas que sirven, por ejemplo, para calcular distintos tipos de logaritmos, el seno, el coseno, etc. Pero no las vamos a utilizar:

- exp(x): sirve para calcular e^x.
- log(x): sirve para calcular el logaritmo neperiano de x, es decir, *ln x*.
- log10(x): calcula el logaritmo de x en base 10.
- sin(x), cos(x), tan(x): para calcular el seno, el coseno y la tangente.
- asin(x), acos(x), atan(x): para calcular el arcoseno, el arcocoseno y el arcotangente.
- sinh(x), cosh(x), tanh(x): para calcular el seno y el coseno hiperbólicos y la tangente hiperbólica.
- …

1.6.2 Funciones predefinidas relacionadas con caracteres

En este apartado se mencionarán algunas funciones predefinidas relacionadas con los caracteres. Para utilizar estas funciones habrá que añadir **#include < ctype.h >** en la cabecera del programa.

- **isupper**
  isupper(x): devuelve 1 si el valor de x (de tipo char) es una letra mayúscula y 0 en caso contrario.

  Ejemplo:

```
int ma;
char c;
c = 'A';
ma = isupper(c); /* se guardará el valor 1 en la variable ma. */
ma = isupper(c + 1); /* se guardará el valor 1 en la variable ma. */
ma = isupper('r'); /* se guardará el valor 0 en la variable ma. */
ma = isupper('&'); /* se guardará el valor 0 en la variable ma. */
```

- **islower**
  islower(x): devuelve 1 si el valor de x (de tipo char) es una letra minúscula y 0 en caso contrario.

- **isalpha**
  isalpha(x): devuelve 1 si el valor de x (de tipo char) es una letra y 0 en caso contrario.

- **isdigit**
  isdigit(x): devuelve 1 si el valor de x (de tipo char) es un dígito y 0 en caso contrario.

- **toupper**
  toupper(x): devolverá la mayúscula correspondiente al valor x (que será de tipo char). Si el valor
  x es una mayúscula o no es una letra, devuelve el mismo valor x.

  Ejemplo:

```
char car1, car2;
car1 = 'a';
car2 = toupper(car1); /* se guardará el valor 'A' en car2. */
car2 = toupper(car1 + 1); /* se guardará el valor 'B' en car2. */
car2 = toupper('R'); /* se guardará el valor 'R' en car2. */
```

```
car2 = toupper('&'); /* se guardará el valor '&' en car2. */
```

- **tolower**
  tolower(x): devolverá la minúscula correspondiente al valor x (que será de tipo char). Si el valor x es una minúscula o no es una letra, devuelve el mismo valor x.

1.6.3 Funciones predefinidas relacionadas con cadenas de caracteres

A continuación se mencionan dos funciones predefinidas relacionadas con las cadenas de caracteres. Para utilizar estas funciones es necesario añadir **#include < string.h >** en la cabecera del programa.

- **strlen**
  strlen(cad): indica cuántos caracteres hay en la cadena cad.
- **strcmp**
  strcmp(cad1, cad2): si las cadenas cad1 y cad2 son iguales, devolverá un 0 y si no devolverá un 1.

1.6.4 Funciones definidas por el programador

Además de utilizar las funciones predefinidas, el programador puede definir y utilizar nuevas funciones. En este tema se explicará cómo definir nuevas funciones y cómo utilizarlas.
Al definir nuevas funciones hay dos opciones:

- Escribir las funciones en un fichero con extensión h y mencionar dicho fichero mediante un comando *include* en la cabecera del programa.
- Escribir las funciones directamente en el fichero del programa. En este a caso, a su vez, hay dos posibilidades
  - Poner los prototipos de las funciones delante de la función main y colocar las definiciones detrás de la definición de main.
  - Colocar directamente las definiciones de las funciones delante de la definición de main. En este caso no hará falta dar los prototipos.

1.7 ¿Para qué sirve una función?

Utilizando funciones se consigue dividir un programa en subprogramas y resolver las distintas tareas cada una por su lado de manera independiente.
Una función puede llevar a cabo distintos tipos de tareas:

- Realizar cálculos a partir de los argumentos y devolver el resultado obtenido para que sea utilizado en otra parte (en la función main o en alguna otra función).
- Pedir al usuario que teclee algún dato, recogerlo y devolverlo para que sea utilizado en la función main o en alguna otra función.
- Mostrar por pantalla mensajes y resultados.
- Realizar cálculos a partir de los argumentos y mostrar los resultados por pantalla.
- Pedir al usuario que teclee algún dato o algunos datos, recogerlos, realizar cálculos a partir de esos datos y mostrar los resultados por pantalla.
- Una función también puede llamar a otras funciones para realizar subcálculos.

1.8 Instrucción RETURN

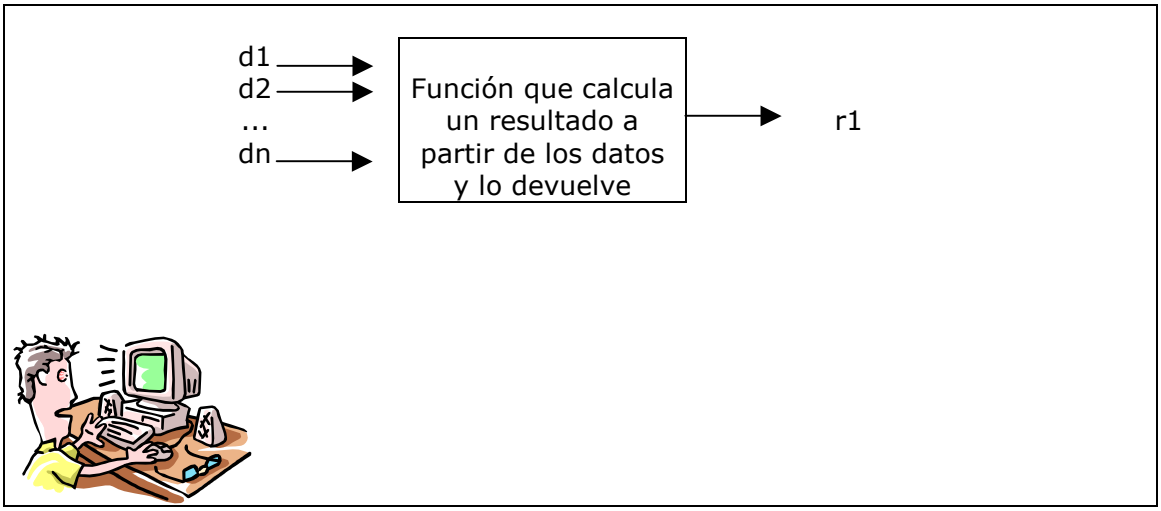
Tal como se ha indicado en el apartado anterior, las funciones pueden llevar a cabo distintos tipos de tareas. Cuando el objetivo de una función es calcular un resultado o pedir un dato por teclado y devolverlo para que sea utilizado en alguna otra función, habrá que utilizar la instrucción return para precisar cuál es el valor que devuelve la función.

Por medio de la instrucción return solo se pueden devolver valores númericos o caracteres simples, es decir, un número o un carácter, pero no se podrán devolver más de un carácter o más de un número.
En los siguientes apartados se mostrarán, por medio de ejemplos, funciones que llevan a cabo distintos tipo de tareas. No hay ninguna limitación a la hora de mezclar las características de los distintos tipos de funciones que se verán. Los tipos de funciones que se muestran son sólo los más usuales.

2 FUNCIONES QUE CALCULAN UN ÚNICO RESULTADO A PARTIR DE LOS ARGUMENTOS

2.1 Características y definición

Las funciones de este tipo reciben como argumentos uno o varios datos y calculan y devuelven un resultado.
El resultado devuelto por la función puede ser guardado en una variable o puede ser utilizado en una expresión o una comparación:
El siguiente gráfico muestra la interacción de este tipo de funciones con el entorno:



d1, d2, ..., dn son los datos pasados como argumentos y r1 es el resultado que devuelve la función.
Tal como se puede apreciar en el gráfico, las funciones de este tipo no piden al usuario que teclee nada y tampoco muestran nada por pantalla.
Todas las funciones matemáticas que se han mencionado anteriormente (fabs, sqrt, pow, sin, ...) son de este tipo: reciben un dato o más y devuelven un resultado.
A continuación se indican las características de los prototipos, definiciones y usos (llamadas) de este tipo de funciones:

- **Prototipo**: los protipos tienen el siguiente formato:

```
tipo_del_resultado nombre_de_la_función(tipos y nombres de los
```

- **Definición**: las definiciones se ajustan al siguiente esquema:

```
tipo_del_resultado nombre_de_la_función(tipos y nombres de los parámetros o datos)
  {
    instrucciones
    return(resultado);
  }
```

La instrucción **return** se utiliza para indicar cuál es el resultado que devuelve la función.

- **Uso (llamadas a función)**: el resultado devuelto por las funciones puede ser guardado en una variable o puede ser utilizado en una expresión o una comparación.

a) Para asignar a una variable el resultado devuelto por una función se escribirá lo siguiente:

```
variable = nombre_de_la_funcion
```

Ejemplo:

```
int n = 5, m;
m = pow (n + 1, 3); /* se le asignará 216 a m */
```

b) Para utilizar el resultado en una expresión, se pondrá la llamada directamente en la expresión:

```
nombre_de_la_funcion (nombres_de_los_argumentos)
```

Ejemplos:

```
int n = 5, m;
m = n + pow (n + 1, 3); /* se le asignará 223 a m */
```

```
int n = 5, m;
if (pow (n + 1, 3) < 300)
 {
   m = 0;
 }
```

2.2 Ejemplos

2.2.1 Función que calcula el factorial

A continuación se escribirá una función que sirve para calcular el factorial de un número entero y no negativo x (parámetro formal). La función devolverá un resultado de tipo long, que será el factorial de x:

- **Prototipo**:

```
long calcular_factorial (int
```

Tipo del resultado | Nombre de la función | Tipo y nombre del dato

- **Definición**:

En la definición de la función *calcular_factorial*, hay especificar en lenguaje C las instrucciones a llevar a cabo para calcular el factorial de un número entero x (≥ 0).
El factorial de un numero entero y no negativo x se calculará de la siguiente forma:

- Si x vale 0, el resultado es directamente 1.

- Si x > 0, el resultado se obtendrá multiplicando todos los números enteros comprendidos entre 1 y x.

```
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 */
```

Mediante la instrucción **return** se indica cuál es resultado de la función. El resultado es lo que quede en la variable f, no es ni x, ni num, ni ningún otro valor.

- **Uso (llamada a función)**:

A continuación se muestran algunos ejemplos de cómo se puede utilizar la función que se acaba de definir.

Ejemplo:
Supongamos que queremos asignar a la variable n (de tipo long) el factorial de 7 y a la variable p el factorial de m:

```
long n, p;
int m = 12;

n = calcular_factorial (7);
p = calcular_factorial (m);
```

Ejemplo:
Sopongamos que si el factorial de 5 es mayor que m, queremos asignar a n el factorial de m + 2, y si no queremos asignar a n el valor -1:

```
long n;
int m = 9;

if (calcular_factorial (5) > m)
 {
   n = calcular_factorial (m + 2);
 }
else
 {
   n = -1;
 }
```

Ejemplo:
A continuación se muestra como podríamos asignar a n el factorial del factorial de m + 3:

```
long n;
int m = 6;

n = calcular_factorial (calcular_factorial (m + 3));
```

2.2.2 Función que decide si un número es primo

A continuación se escribirá una función que, dado un número entero x mayor o igual que 1, devuelve un 1 si el número es primo y 0 en caso contrario:

- **Prototipo**:

```
int es_primo (int x);
```

Tipo del resultado | Nombre de la función | Tipo y nombre del dato

- **Definición**:
En la definición de la función *es_primo* hay que precisar los cálculos a realizar para decidir si un número x es primo o no. Para ello se contarán sus divisores. Si tiene justo 2 divisores, es primo y si tiene más o menos divisores que 2, no es primo. La función siempre devuelve 0 ó 1. Devuelve 0 si x no es primo y devuelve 1 si es primo:

```
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 */
```

- **Uso (llamada a función)**:

  A continuación se dan algunos ejemplos que muestran la manera de utilizar la función *es_primo*.

  Ejemplo:
  Supongamos que queremos decidir si el número 7 y el valor de n son primos y que queremos guardar las respuestas (0 ó 1) en las variables re1 y re2. Lo haríamos de la siguiente forma:

```
int n = 20, re1, re2;

re1 = es_primo (7);
re2 = es_primo (n);
```

  Ejemplo:
  Supongamos que si n + 3 es primo queremos guardar n * 2 en m y que si no queremos guardar n / 2:

```
int n = 23, m;

if (es_primo (n + 3) == 1)
 {
   m = n * 2;
 }
else
 {
   m = n / 2;
 }
```

2.2.3 Función que, dados dos números enteros y positivos (≥ 1) x e y, calcula x^y

A continuación se definirá una función que, dados dos números enteros y positivos x e y (tal que $x \geq 1$ e $y \geq 1$), calcula x^y. En realidad para realizar este cálculo ya se tiene la función predefinida pow, pero vamos a definirla nosotros. La función que vamos a definir no es tan general como pow porque pow sirve también para números reales pero nuestra función servirá sólo para números enteros.

- **Prototipo**:

```
int mi_pow (int x,
```

tipo del resultado — nombre de la función — tipos y nombres de los datos

- **Definición**:

  En la definición de la función *mi_pow* hay que indicar cómo se calcula x^y a partir de x e y: se multiplicará x y veces (x * x * x * …):

```
int mi_pow (int x, int y)

 /* Esta función, dados dos números x e y que son enteros y positivos
    (>= 1), calcula y devuelve el valor de x elevado a y. */

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

   /* se utilizará cont para contar cuántas veces se ha multiplicado x
   */

   elev = 1; /* para ir calculando el resultado x elevado a y */

   for (cont = 0; cont < y; cont = cont + 1)
    {
      elev = elev * x;
    }

   return (elev);
} /* Fin de la función */
```

- **Uso (llamada a función)**:

  A continuación se dan algunos ejemplos que muestran cómo utilizar la función *mi_pow*.

  Ejemplo:
  Supongamos que en la variable b se quiere guardar el valor 7^n. Escribiríamos lo siguiente:

```
int n, b;

b = mi_pow (7, n);
```

  Ejemplo:
  Supongamos que si n^5 es mayor que m, se quiere guardar n * 2 en p y que si no, se quiere guardar n / 2 en p. Escribiríamos lo siguiente:

```
int m, n, p;

if (mi_pow (n, 5) > m)
 {
   p = n * 2;
 }
else
 {
   p = n / 2;
 }
```

3 FUNCIONES QUE PIDEN AL USUARIO QUE TECLEE ALGO, RECOGEN LO TECLEADO Y LO DEVUELVEN

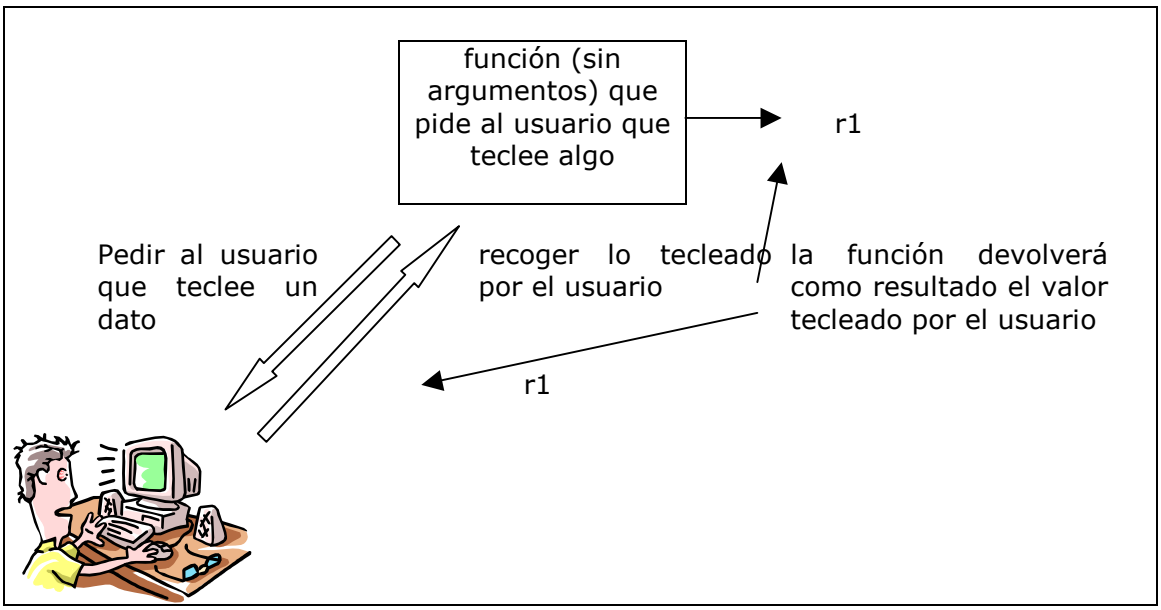
3.1 Características y definición

Las funciones que piden al usuario que teclee algo pueden tener o no tener argumentos, pero siempre piden al usuario que teclee un único valor y devuelven ese valor como resultado. En concreto, estas funciones pedirán al usuario que teclee un número o un carácter y cuando el usuario teclee lo solicitado, la función lo recogerá y lo devolverá como resultado utilizando la instrucción **return**. Tal como se verá en los ejemplos, a veces lo que

ha de tecleear el usuario tiene que cumplir alguna condición. En estos casos la función se encargará de repetir el proceso de petición hasta obtener un valor que cumpla la condición.

El número o carácter devuelto por la función se guardará normalmente en una variable.

a) El siguiente gráfico muestra la relación de las funciones de este tipo sin argumentos con el entorno:



r1 es el valor tecleado por el usuario y que la función devolverá para que sea utilizado en otra función.
A continuación se presentan las características de las funciones de este tipo:

- **Prototipo**:

```
tipo_del_resultado nombre_de_la_función
```

Como no hay argumentos, habrá que poner void entre los paréntesis.

- **Definición**: la definición tendrá el siguiente formato:

```
tipo_del_resultado nombre_de_la_función (void)
  {
    variables
    instrucciones
    return(resultado);
  }
```

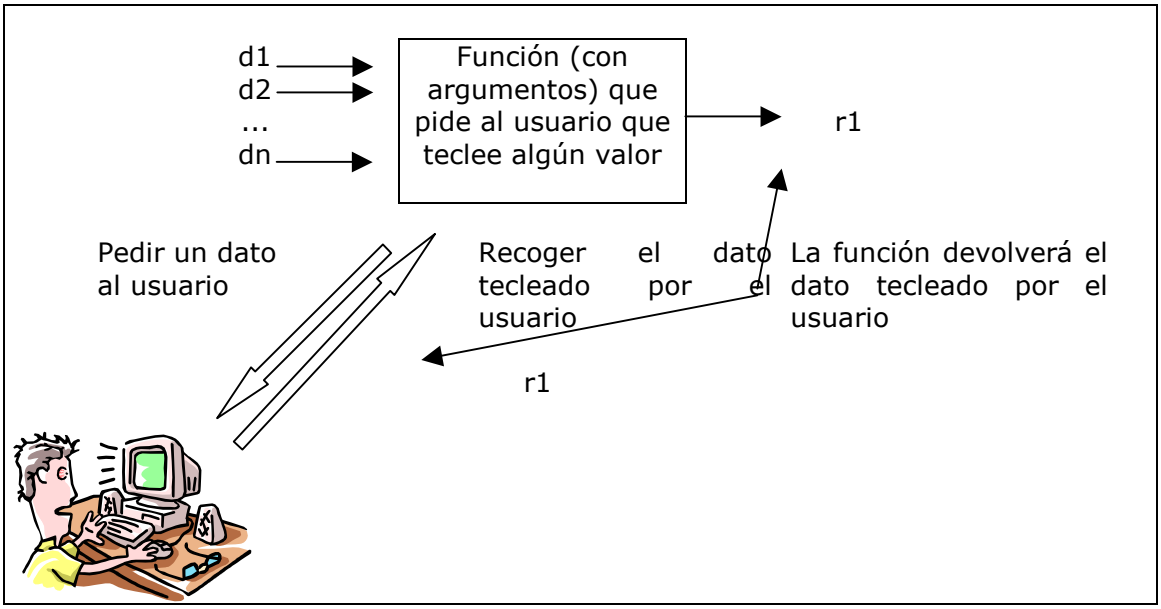
La instrucción **return** se utiliza para indicar cuál es el resultado que devuelve la función.

- **Uso (llamada a función)**: el resultado devuelto por una función de este tipo se guardará normalmente en una variable:

```
variable = nombre_de_la_función ( );
```

Como la función no tiene argumentos, al hacer la llamada no hay que poner nada entre los paréntesis.

b) El siguiente gráfico muestra la relación de las funciones que tienen argumentos y que piden al usuario que teclee un valor y el entorno:



d1, d2, ..., dn son los datos pasados como argumentos y r1 es el resultado que devuelve la función.
A continuación se muestran las características de las funciones de este estilo:

- **Prototipo**:

```
tipo_del_resultado nombre_de_la_función (tipos y nombres de los argumentos);
```

- **Definición**: la definición tiene el siguiente formato:

```
tipo_del_resultado nombre_de_la_función (tipos y nombres de los argumentos)
  {
    variables
    instrucciones
    return(resultado);
  }
```

La instrucción **return** se utiliza para indicar cuál es el resultado que devuelve la función.

- **Uso (llamada a función)**: el resultado devuelto por una función de este tipo se guardará normalmente en una variable:

```
variable = nombre_de_la_funcion (nombre de los argumentos);
```

3.2 Ejemplos

3.2.1 Función que pide un número entero

A continuación se definirá una función que pide un número entero al usuario. Cuando el usuario teclee un número entero, la función lo recogerá y lo devolverá como resultado:

- **Prototipo**:

```
int
pedir_un_numero_entero(void);
```

Tipo del resultado | Nombre de la función | Para indicar que no hay argumentos se pone void

- **Definición**:

En la definición de la función *pedir_un_numero_entero*, hay que indicar lo que hay que hacer para pedir y recoger un número entero:

```
int pedir_un_numero_entero(void)

 /* Esta función pedirá un número entero al usuario y cuando el usuario
    lo teclee, lo recogerá y lo devolverá como resultado. */

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

   printf("\nTeclea un número entero: ");
   scanf("%d", &num);

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

- **Uso (llamada a función)**:

A continuación se dan algunos ejemplos de cómo utilizar la función.

Ejemplo:
Supongamos que queremos pedir al usuario que nos dé un número entero. En vez de hacerlo utilizando directamente printf y scanf, podemos utilizar la función que acabamos de definir. En este ejemplo el número se guardará en la variable n:

```
int n;
n = pedir_un_numero_entero();
```

Ejemplo:
Supongamos que ahora queremos pedir dos números enteros al usuario. Una posibilidad sería hacerlo directamente por medio de printf y scanf, pero también podemos hacerlo utilizando la función que acabamos de definir. Pero como esa función pide un sólo número, habrá que llamarle dos veces:

```
int n, m;

n = pedir_un_numero_entero();
m = pedir_un_numero_entero();
```

Tal como se ha visto en estos ejemplos, como esta funcíon no tiene argumentos no hay que poner nada entre paréntesis.

3.2.2 Función que pide un número entero mayor que 0

A continuación se escribirá una función que pide al usuario que teclee un número entero ≥ 1. La función no tiene argumentos y el proceso de petición se repetirá hasta que el usuario

teclee un número que cumple esa condición. Cuando el usuario introduzca un número mayor que 0 la función lo devolverá como resultado.

- **Prototipo**:

  ```
  int     pedir_numero_positivo
  (void);
  ```

  Tipo del resultado | Nombre de la función | Se pone void para indicar que no tiene argumentos

- **Definición**:

  En la definición se indica lo que hay que hacer para pedir un número mayor que 0 repitiendo el proceso hasta obtener uno que lo sea:

```
int pedir_numero_positivo(void)

 /* Se pedirá al usuario un número >= 1 y se repetirá el proceso de
    petición hasta obtener uno que lo sea. La función devolverá ese
    último número. */

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

   do
   { printf("\nTeclea un numero entero y positivo (>= 1): ");
     scanf("%d", &num);
     if (num <= 0)
      {
        printf("\nEl numero tecleado no es adecuado.");
      }
   } while (num <= 0);

   return(num);
 } /* fin de la función */
```

- **Uso (llamada a función)**:
  A continuación se muestra, por medio de ejemplos, cómo utilizar la función que se acaba de definir.

  Ejemplo:
  Supongamos que queremos pedir al usuario un número entero y positivo para guardarlo en la variable n:

```
int n;
n = pedir_numero_positivo();
```

  Ejemplo:
  Si quisieramos pedirle al usuario dos números enteros y positivos, podríamos hacerlo de la siguiente forma:

```
int n, m;

n = pedir_numero_positivo();
m = pedir_numero_positivo();
```

  Como la función no tiene argumentos, al llamarle no hay que poner nada entre los paréntesis.

3.2.3 Función que pide un número entero mayor que z

A continuación se va a definir una función que recibe como argumento un número entero z y pide al usuario que teclee un número entero mayor que z. El proceso de petición se repetirá hasta que el usuario teclee un número que verifique dicha condición. Cuando el usuario teclee un número adecuado, la función devolverá ese número como resultado:

- **Prototipo**:

```
int pedir_numero_mayor (int
                        z);
```

Tipo del resultado — Nombre de la función — Tipo y nombre del dato

- **Definición**:

  En la definición se indicará qué hay que hacer para pedir al usuario un número que sea mayor que z:

```
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 */
```

- **Uso (llamada a función)**:
  A continuación se dan algunos ejemplos que muestran cómo se puede utilizar la función.

  Ejemplo:
  Supongamos que queremos pedirle al usuario un número mayor que 5. Para ello bastará con llamar a la función *pedir_numero_mayor* pasándole como argumento el valor 5:

```
int n;
n = pedir_numero_mayor(5);
```

  Ejemplo:
  Supongamos ahora que queremos pedir al usuario dos números enteros, uno de ellos ≥ 0 y el otro ≥ 1. Para ello habría que llamar a la función *pedir_numero_mayor* dos veces. En la primera llamada le pasaríamos como argumento el -1 (para conseguir un número mayor que -1, es decir, ≥ 0) y en la segunda el 0 (para conseguir un número mayor que 0, es decir, ≥ 1):

```
int m, n;

m = pedir_numero_mayor(-1);
n = pedir_numero_mayor(0);
```

Tal como se ha visto en estos ejemplos, al llamar a la función *pedir_numero_mayor* hay que poner un número entre los paréntesis. La función pedirá al usuario un número mayor que el que le hemos dado y repetirá el proceso de petición hasta obtener un número que cumpla dicha condición. Cuando finalmente obtenga un número que cumpla esa condición, lo devolverá como resultado.

3.2.4 Función que pide un número entero del intervalo [x, y]

A continuación definiremos una función que, dados dos números enteros x e y ($x \leq y$), pide al usuario que teclee un número entero del intervalo [x, y]. El proceso de petición se repetirá hasta obtener un número que cumpla la condición requerida. Cuando se obtenga un número que cumple la condición, se devolverá ese número como resultado:

- **Prototipo**:

```
int pedir_numero_intervalo (int
                x, int y);
```

Tipo del resultado | Nombre de la función | Tipos y nombres de los datos

- **Definición**:

En la definición se indica lo que hay que hacer para pedir al usuario un número de un intervalo definido por los valoes x e y:

```
int pedir_número_intervalo (int x, int y)

 /* Esta función pedirá al usuario un número del intervalo [x, y] y
    repetirá el proceso de petición hasta obtener un número que esté en
    ese intervalo. Cuando se obtenga un número del intervalo, la función
    devolverá ese número como resultado. Se presupone que x será menor o
    igual que y. */

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

   do
    {
      printf("\nTeclea un numero entero del intervalo [%d, %d]: ",
             x, y);
      scanf("%d", &num);
      if ((num < x) || (num > y))
       {
         printf("\nEl numero tecleado no es adecuado.");
       }
    } while ((num < x) || (num > y));

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

- **Uso (llamada a función)**:

A continuación se muestran algunos ejemplos de cómo utilizar la función que acabamos de definir.

Ejemplo:

Supongamos que queremos pedir al usuario un número entero del intervalo [3, 12]. Para ello sería suficiente llamar a la función *pedir_numero_intervalo* pasándole como argumentos los valores 3 y 12:

```
int n;
n = pedir_numero_intervalo(3, 12);
```

Ejemplo:

Supongamos ahora que queremos pedir al usuario que introduzca dos números enteros del intervalo [0, 10]. Para ello podríamos utilizar la función *pedir_numero_intervalo*, pero como esa función pide un número cada vez que se le llama, habría que llamarle dos veces. En cada llamada le pasaríamos los valores 0 y 10 como argumentos:

```
int m, n;

m = pedir_numero_intervalo(0, 10);
n = pedir_numero_intervalo(0, 10);
```

Ejemplo:

A continuación se indican las llamadas que habría que hacer a la función *pedir_numero_intervalo* en caso de que quisieramos pedir al usuario un número entero del intervalo [-20, 15] y otro del intervalo [30, 60]:

```
int m, n;

m = pedir_numero_intervalo(-20, 15);
n = pedir_numero_intervalo(30, 60);
```

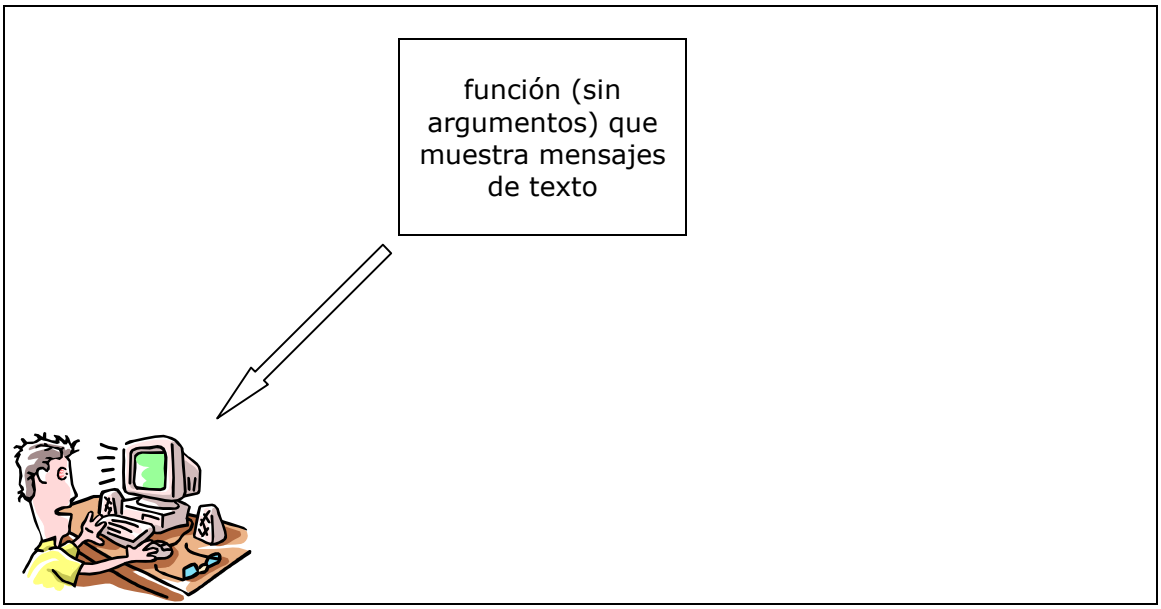
Tal como se ha visto en estos ejemplos, al llamar a la función *pedir_numero_intervalo* hay que poner entre paréntesis dos números enteros. La función pedirá al usuario que teclee un número entero del intervalo definido por esos dos valores, repitiendo el proceso hasta obtener un número del intervalo. Ese número será lo que devuelva la función como resultado.

4 FUNCIONES QUE MUESTRAN POR PANTALLA MENSAJES Y LOS VALORES DE LOS ARGUMENTOS DADOS

4.1 Características y definición

Las funciones de este tipo pueden tener o no tener argumentos. Las que no tienen argumentos sirven para mostrar por pantalla mensajes de texto. Las que tienen argumentos sirven para mostrar el valor de esos argumentos.

a) El siguiente gráfico muestra la interacción entre las funciones de este tipo que no tienen argumentos y el entorno:



- **Prototipo**:

```
void nombre_de_la_funcion
```

Como estas funciones no devuelven ningún resultado, en el lugar correspondiente al tipo del resultado se pone void y como tampoco tienen argumentos, también se pone void entre los paréntesis.

- **Definición**: la definición de una función de este tipo suele tener el siguiente formato:

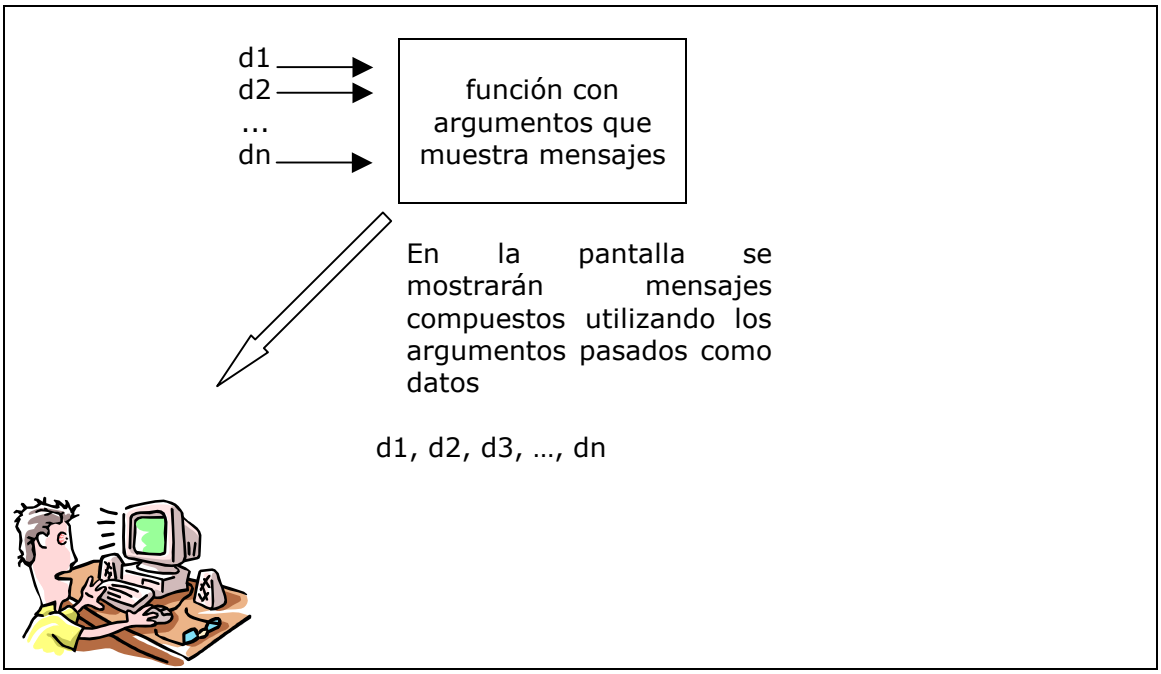
```
void nombre_de_la_funcion (void)
  {
    variables
    instrucciones
  }
```

- **Uso (llamada a función)**: el formato de una llama a una función de este tipo es el siguiente:

```
nombre_de_la_funcion
          ();
```

Como no devuelven ningún resultado, en la izquierda no se pone ninguna variable y como tampoco reciben datos (no tienen argumentos), los paréntesis van vacíos.

b) El siguiente gráfico muestra la interacción entre las funciones de este tipo que tienen argumentos y el entorno:



d1, d2, ..., dn son los argumentos de la función, es decir, los datos que se le pasan a la función.

- **Prototipo**:

```
void nombre_de_la_funcion (tipos y nombres de los argumentos);
```

Como estas funciones no devuelven ningún resultado, se pone void en el sitio correspondiente al tipo del resultado.

- **Definición**: la definición de una función de este tipo suele tener el siguiente formato:

```
void nombre_de_la_funcion (tipos y nombres de los argumentos)
  {
    variables
    instrucciones
  }
```

- **Uso (llamada a función)**: el formato de una llama a una función de este tipo es el siguiente:

```
nombre_de_la_funcion (nombres de los
```

Como no devuelven ningún resultado no hay que poner ninguna variable en la izquierda.

4.2 Ejemplos

4.2.1 Función que indica que la secuencia de números tecleada por el usuario es vacía

A continuación se definirá una función que muestra un mensaje indicando que la secuencia de números introducida por el usuario es vacía.
Esta función no tiene argumentos y no devuelve ningún resultado. Es un ejemplo del caso a):

- **Prototipo**:

  void secuencia_vacia (void);

  Por no devolver resultado | Nombre de la función | Por no tener datos

- **Definición**:

```
void secuencia_vacia (void)

 /* Esta función muestra un mensaje por pantalla indicando que la
    secuencia de números introducida por el usuario es vacía. */

 { /* Inicio de la función */

   printf("\nLa secuencia de numeros es vacia.");

 } /* Fin de la función */
```

- **Uso (llamada a función)**:

  La manera de utilizar esta función sería mediante una llamada de la siguiente forma:

  secuencia_vacia ();

4.2.2 Función que muestra por pantalla el número x

A continuación se escribirá una función que, dado un número entero x como argumento, lo muestra por pantalla. Es un ejemplo del caso b):

- **Prototipo**:

  void mostrar_numero (int x);

  Por no tener resultado | Nombre de la función | Tipo y nombre del dato o argumento

- **Definición**:

```
void mostrar_numero (int x)

 /* Esta función mostrará el número x por pantalla. */

 { /* Inicio de la función */

   printf("\n%d", x);
```

```
} /* Fin de la función */
```

- **Uso (llamada a función)**:

  A continuación se presentan algunos ejemplos que muestran cómo utilizar la función que acabamos de definir.

  Ejemplo:

  Supongamos que queremos mostrar en pantalla los números 5, -68 y 40. Utlizando la función *mostrar_numero* podríamos hacerlo de la siguiente forma:

```
mostrar_numero(5);
mostrar_numero(-68);
mostrar_numero(40);
```

  Ejemplo:

  Si quisieramos mostrar por pantalla los valores de n y m + 1, utilizando la función *mostrar_numero* podríamos hacerlo de la siguiente forma:

```
int m = 10, n = 20;

mostrar_numero(n);
mostrar_numero(m + 1);
```

  Tal como se ha visto en estos ejemplos, al llamar a la función *mostrar_numero* tenemos que ponerle entre paréntesis un número o una expresión de tipo entero y la función lo mostrará por pantalla pero no devuelve ningún resultado.

4.2.3 Función que muestra el factorial de x

A continuación se escribirá una función que, dados un número x y su factorial f, muestra por pantalla un mensaje indicando que el factorial de x es f. Es un ejemplo del caso b):

- **Prototipo**:

```
void mostrar_factorial (int x,
long f);
```

  ↑ Por no devolver resultado — ↑ Nombre de la función — ↑ Tipos y nombres de los datos o argumentos

- **Definición**:

```
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 */
```

- **Uso (llamada a función)**:

  A continuación se muestran algunos ejemplos para ver cómo se utilizaría la función que acabamos de definir.

  Ejemplo:
  Supongamos que queremos mostrar un mensaje que diga que 120 es el factorial de 5. Utilizando la función *mostrar_factorial* lo haríamos de la siguiente forma:

```
mostrar_factorial(5, 120);
```

  Ejemplo:
  Ahora calcularemos y mostraremos el factorial de 8:

```
long fact;

fact = calcular_factorial(8);
mostrar_factorial(8, fact);
```

  Ejemplo:
  Para calcular y mostrar los factoriales de n y m + 3 podríamos hacer lo siguiente:

```
int n = 10, m = 6;
long fact1, fact2;

fact1 = calcular_factorial(n);
mostrar_factorial(n, fact1);
fact2 = calcular_factorial(m + 3);
mostrar_factorial(m + 3, fact2);
```

4.2.4 Función que muestra un mensaje indicando si x es primo o no

A continuación se escribirá una función que dados un número entero x (≥ 1) y un valor p (1 ó 0) que indica si x es primo o no, muestra por pantalla un mensaje diciendo si x es primo o no.
Por tanto, el significado del segundo argumento (que siempre valdrá 1 ó 0) será el siguiente:

- Si vale 1, indica que x es primo.
- Si vale 0, indica que x no es primo.

- **Prototipo**:

```
void mostrar_primo (int x, int
p);
```

  ↑ Por no devolver resultado ↑ Nombre de la función ↑ Tipos y nombres de los argumentos

- **Definición**:

```
void mostrar_primo (int x, int p)

 /* Si el valor de p es 1, esta función mostrará un mensaje diciendo que
    x es primo y si el valor de p es 0, mostrará un mensaje diciendo que
    x no es primo. */
```

```
{ /* Inicio de la función */
  if (p == 1)
   {
     printf("\nEl numero %d es primo.", x);
   }
  else
   {
     printf("\nEl numero %d no es primo.", x);
   }
} /* Fin de la función */
```

- **Uso (llamada a función)**:

  A continuación se muestran algunos ejemplos para ver cómo se utilizaría la función que acabamos de definir.

  Ejemplo:
  Supongamos que queremos mostrar un mensaje que diga que 19 es un número primo. Utilizando la función *mostrar_primo* lo podemos hacer de la siguiente manera:

```
mostrar_primo(19, 1);
```

  Ejemplo:
  Ahora supongamos que queremos mostrar un mensaje diciendo que 27 no es primo. Utilizando la función *mostrar_primo* lo podemos hacer de la siguiente manera:

```
mostrar_primo(27, 0);
```

  Ejemplo:
  En este ejemplo se ve cómo se puede mirar si el númeo n es primo o no y mostrar el mensaje correspondiente. Para decidir si n es primo o no se utilizará la función *es_primo* que ya ha sido definida con anterioridad:

```
int n = 20;

if (es_primo (n) == 1)
 {
   mostrar_primo (n, 1);
 }
else
 {
   mostrar_primo (n, 0);
 }
```

  También conseguiríamos lo mismo de la siguiente forma:

```
int n = 20;

mostrar_primo (n, es_primo(n));
```

5 FUNCIONES QUE REALIZAN CÁLCULOS, MUESTRAN LOS RESULTADOS Y LLAMAN A OTRAS FUNCIONES

5.1 Características y definición

También estas funciones pueden tener o no tener argumentos. Las que no tengan argumentos deberán encargarse primero de obtener datos pidiéndoselos al usuario para a continuación realizar cálculos y mostrar resultados. Tanto a la hora de obtener datos como a la hora de llevar a cabo los cálculos y presentar los resultados, cabe la posibilidad de

llamar a otras funciones para las distintas subtareas. En el caso de las funciones de este tipo que sí tienen argumentos, los argumentos serán generalmente los datos que la función necesita para realizar el resto de las tareas.
El formato del prototipo y de la definición y la manera de utilizar este tipo de funciones es igual que en el tipo anterior que hemos visto.

5.2 Ejemplos

5.2.1 Función que cálcula el máximo y el mínimo de una secuencia de números tecleada por el usuario

A continuación se escribirá una función que irá pidiendo números enteros al usuario e irá decidiendo cuál es el máximo y cuál el mínimo. Al final mostrará un mensaje para presentar los resultados. A la hora de realizar las subtareas, la mayoría de las veces se llamará a otras funciones: *pedir_elemento_secuencia*, *mostrar_max_min* y *secuencia_vacia*. La primera de esas funciones pide un número de la secuencia, la segunda muestra el máximo y el mínimo una vez que estas hayan sido calculadas y la tercera muestra un mensaje en caso de que la secuencia de números sea vacía y no sea posible calcular el máximo y el mínimo.
Esta función no tiene datos o argumentos y no devuelve ningún resultado. En vez de devolver un resultado, muestra directamente por pantalla el máximo y el mínimo:

- **Prototipo**:

```
void calcular_max_min_secuencia (void);
```

Por no devolver resultado | Nombre de la función | Por no tener datos

- **Definición**:

```
void calcular_max_min_secuencia (void)

 /* Esta función irá pidiendo de uno en uno los elementos de una
    secuencia de números enteros, decidirá cuál es el máximo y cuál el
    mínimo y al final los mostrará. Todos los elementos de la secuencia
    han de ser >= 0 salvo el último que será negativo. */

 { /* Inicio de la función */
   int numero, mayor, menor, cont;

   numero = pedir_elemento_secuencia (1);

   if (numero < 0)
    {
      secuencia_vacia();
    }
   else
    {
      cont = 1; /* Para contar cuántos números se han pedido. */
      mayor = numero;
      menor = numero; /* Al principio el primer número es tanto el mayor
                         como el menor. */

      numero = pedir_elemento_secuencia (cont + 1); /* para obtener el
                                       segundo número de la secuencia */

      while (numero >= 0)
       {
         cont = cont + 1;
```

```
        if (numero > mayor)
         {
           mayor = numero;
         }
        else if (numero < menor)
         {
           menor = numero;
         }
        numero = pedir_elemento_secuencia (cont + 1);
      }
     mostrar_max_min (mayor, menor);
   }
} /* Fin de la función */
```

- **Uso (llamada a función)**:

  Para utilizar la función, es decir, para hacer una llamada a esta función habrá que escribir lo siguiente:

```
calcular_max_min_secuencia
();
```

5.2.2 Función que calcula y muestra los factoriales de los primeros z números primos

En este caso se va a escribir una función que, dado un número entero z ≥ 1, calcula y muestra los factoriales de los primeros z números primos.
Esta función llamará a las funciones *calcular_factorial*, *es_primo*, *mostrar_primo* y *mostrar_factorial* para realizar las correspondientes subtareas.
El argumento de la función es un dato que se le pasa a la función y que la función ha de tener en cuenta durante el proceso.

- **Prototipo**:

```
void mostrar_factoriales_de_primos (int z);
```

Por no devolver resultados. Los resultados se mostrarán por pantalla.

Nombre de la función

Tipo y nombre del dato o argumento

- **Definición**:

```
void mostrar_factoriales_de_primos (int z)

 /* Esta función calcula y muestra los primeros z factoriales primos. */

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

   num = 1; /* Para ir pasando los números de uno en uno. */
   cont = 0; /* Para ir contabilizando el número de factoriales primos
                que se vayan encontrando. */

   while (cont < z)
```

```
    {
      if (es_primo (num) == 1)
       {
         mostrar_primo (num, 1);
         fact = calcular_factorial (num);
         mostrar_factorial (num, fact);
         cont = cont + 1;
       }
      num = num + 1;
    }
} /* Fin de la función */
```

- **Uso (llamada a función)**:

  A continuación se muestra un ejemplo para ver qué haría esta función al ser llamada.

  Ejemplo:

  Supongamos que queremos los factoriales de los cuatro primeros números primos. Utilizando la función *mostrar_factoriales_de_primos* escribiríamos la siguiente llamada:

```
mostrar_factoriales_de_primos
(4);
```

  El ordenador nos mostraría por pantalla los factoriales de 2, 3, 5 y 7:

Pantalla

```
El numero 2 es primo.
El factorial de 2 es 2.
El numero 3 es primo.
El factorial de 3 es 6.
El numero 5 es primo.
El factorial de 5 es 120.
El numero 7 es primo.
El factorial de 3 es 5040.
```

6 EJERCICIOS RESUELTOS

6.1 Calcular los factoriales de seis, ocho y quince

En este ejerecicio se ha de escribir un programa que calcula y muestra por pantalla los factoriales de 6, 8 y 15.

6.1.1 Primera versión: utilizando la función *calcular_factorial*

Con anterioridad se ha definido la función que calcula el factorial de un número entero ≥ 0. Ahora veremos dónde colocar esa función al escribir un programa completo y cómo utilizarla.
Funciones a utilizar:

- *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>

/* Datos: Este programa no pedirá datos al usuario. */
/* Resultados: Calculará y mostrará por pantalla los factoriales de A, B y C.
*/

/* Constantes:
          - A, B, C: El programa calcula los factoriales de estas tres
                     constantes. */

/* Variables:
          - f1, f2, f3: para guardar los factoriales de A, B y C. */

#define A 6
#define B 8
#define C 15

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

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

/* Función principal */

void main()
 { /* Inicio de la función main */
   long f1, f2, f3;

   f1 = calcular_factorial (A);
   f2 = calcular_factorial (B);
   f3 = calcular_factorial (C);

   mostrar_factorial (A, f1);
   mostrar_factorial (B, f2);
   mostrar_factorial (C, f3);

   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 */

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 */

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

El ordenador siempre ejecuta la función main (la función principal) y si desde main se le llama a otra función, se realizará con los argumentos que se le pasen en la llamada lo que se hace en la definición de la función con los parámetros formales.
En este ejercicio se le ha llamado a la función *calcular_factorial* tres veces: *calcular_factorial (A)*, *calcular_factorial (B)* y *calcular_factorial (C)*. A, B y C son los argumentos que se le pasan a la función en las diferentes llamadas mientras que el x que se ha utilizado en la definición es el parámetro formal. Al definir la función se indica cómo calcular el factorial para cualquier valor x y al llamar a la función en lugar de x podemos poner cualquier constante, variable o expresión entera.
Por ejemplo cuando se realiza la llamada *calcular_factorial(A)*, el ordenador hará con A lo que en la definición de la función se hacía con x y de esa forma calculará el factorial de A.

6.1.2 Segunda versión: Sin utilizar funciones

A continuación se presenta un programa que resuelve el mismo ejercicio pero sin utilizar las dos funciones que se han definido en la primera versión. Tal como se va a ver, para calcular los factoriales de A, B y C hará falta escribir tres veces el proceso a seguir para calcular un factorial:

```
#include <stdio.h>

/* Datos: Este programa no pedirá datos al usuario. */

/* Resultados: Calculará y mostrará por pantalla los factoriales de A, B y C.
*/

/* Constantes:
          - A, B, C: El programa calcula los factoriales de estas tres
                     constantes. */

/* Variables:
          - f1, f2, f3: para guardar los factoriales de A, B y C. */

#define A 6
#define B 8
#define C 15

void main()
 { /* Inicio de la función main */
   int num;
   long f1, f2, f3;

   /* Cálculo del factorial de A */
   if (A == 0)
    {
      f1 = 1;
    }
   else
```

```
    {
      f1 = 1;
      for (num = 1; num <= A; num = num + 1)
       {
         f1 = f1 * num;
       }
    }

   /* Cálculo del factorial de B */
   if (B == 0)
    {
      f2 = 1;
    }
   else
    {
      f2 = 1;
      for (num = 1; num <= B; num = num + 1)
       {
         f2 = f2 * num;
       }
    }

   /* Cálculo del factorial de C */
   if (C == 0)
    {
      f3 = 1;
    }
   else
    {
      f3 = 1;
      for (num = 1; num <= C; num = num + 1)
       {
         f3 = f3 * num;
       }
    }

   printf("\nEl factorial de %d es %ld.", A, f1);
   printf("\nEl factorial de %d es %ld.", B, f2);
   printf("\nEl factorial de %d es %ld.", C, f3);

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

Analizando la segunda versión de este ejercicio, se pueden apreciar las ventajas de utilizar la función *calcular_factorial*:

- El programa de la primera versión es más corto.
- En la primera versión la función main es nás fácil de entender y de seguir.
- En la primera versión sólo se especifica una vez cómo se calcula el factorial de un número, mientras que en la segunda versión se hace tres veces.

6.2 Decidir si los tres números dados por el usuario son primos

El programa que se va a dar a continuación pide al usuario tres números enteros m, n y p mayores o iguales que 1 y mostrará un mensaje por cada uno de ellos indicando si es primo o no.
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.

- *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.

- *mostrar_primo*: 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 primo y si p es 0, muestra un mensaje por pantalla indicando que x no es primo.

```
#include <stdio.h>

/* Datos: Este programa pedirá al usuario tres números enteros y >= 1. */

/* Resultados: Mostrará por pantalla tres mensajes indicando si cada número es
               primo o no. */

/* Variables:
          - m, n y p: Para guardar los tres números que tecleará el usuario.
*/

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

int pedir_numero_mayor(int z);
int es_primo (int x);
void mostrar_primo (int x, int p);

/* Función principal */

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

   m = pedir_numero_mayor (0);
   n = pedir_numero_mayor (0);
   p = pedir_numero_mayor (0);

   if (es_primo (m) == 1)
    {
      mostrar_primo (m, 1);
    }
   else
    {
      mostrar_primo (m, 0);
    }

   if (es_primo (n) == 1)
    {
      mostrar_primo (n, 1);
    }
   else
    {
      mostrar_primo (n, 0);
    }

   if (es_primo (p) == 1)
    {
      mostrar_primo (p, 1);
    }
   else
    {
      mostrar_primo (p, 0);
    }

   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 */

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

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 */

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

void mostrar_primo (int x, int p)

 /* Si el valor de p es 1, esta función mostrará un mensaje diciendo que
    x es primo y si el valor de p es 0, mostrará un mensaje diciendo que
    x no es primo. */

 { /* Inicio de la función */
   if (p == 1)
    {
      printf("\nEl numero %d es primo.", x);
```

```
    }
   else
    {
      printf("\nEl numero %d no es primo.", x);
    }
 } /* Fin de la función */

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

6.3 Encontrar y mostrar los primeros n números primos

En este ejercicio se escribirá un programa que pedirá al usuario un número entero n (≥ 1) y cuando obtenga un número n que cumpla esa condición, calculará y mostrará los primeros n números primos.
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.

- *encontrar_primos*: función que, dados dos enteros v (≥ 1) y z (≥ 1), encuentra y muestra los primeros z números primos a partir de v. Esta función llamará a las funciones *es_primo* y *mostrar_primo*.

- *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.

- *mostrar_primo*: 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 primo y si p es 0, muestra un mensaje por pantalla indicando que x no es primo.

```
#include <stdio.h>

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

/* Resultados: Mostrará por pantalla los primeros n números primos. */

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

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

int pedir_numero_mayor(int z);
void encontrar_primos (int v, int z);
int es_primo (int x);
void mostrar_primo (int x, int p);

/* Función principal */

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

   n = pedir_numero_mayor (0);

   encontrar_primos (1, n); /* encontrará los primeros n números primos a
                               partir de 1 */

   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 encontrar_primos (int v, int z)

 /* Esta función encuentra y muestra los primeros z números primos a partir de
    v. Tanto v como z serán >= 1. */

 {
   int num, cont;

   num = v; /* Se utilizará para ir pasando los números de uno en uno a partir
               de v. 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 a partir del %d son los "
          "siguientes: ", v, z);

   while (cont < z)
    {
      if (es_primo (num) == 1)
       {
         mostrar_primo (num, 1);
         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 */

                              /********************/
void mostrar_primo (int x, int p)

 /* Si el valor de p es 1, esta función mostrará un mensaje diciendo que
    x es primo y si el valor de p es 0, mostrará un mensaje diciendo que
    x no es primo. */

 { /* Inicio de la función */
   if (p == 1)
    {
      printf("\nEl numero %d es primo.", x);
    }
   else
    {
      printf("\nEl numero %d no es primo.", x);
    }
 } /* Fin de la función */

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

6.4 Calcular y mostrar los primos menores que n

A continuación se va a escribir un programa que pedirá al usuario un número entero n (≥ 1) y cuando obtenga un número n que cumpla esa condición, buscará y mostrará los primos menores que n.
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.

- *encontrar_primos_menores*: función que, dado un número entero z, calcula y muestra los números primos menores que z. Esta función llamará a las funciones *es_primo* y *mostrar_primo*.

- *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.

- *mostrar_primo*: 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 primo y si p es 0, muestra un mensaje por pantalla indicando que x no es primo.

```
#include <stdio.h>

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

/* Resultados: mostrará por pantalla los primos menores que n. */

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

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

int pedir_numero_mayor(int z);
void encontrar_primos_menores (int z);
int es_primo (int x);
void mostrar_primo (int x, int p);

/* Función principal */

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

   n = pedir_numero_mayor (0); /* se quiere un número mayor que 0 */

   encontrar_primos_menores (n); /* encontrará los números primos menores que
                                    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 encontrar_primos_menores (int z)

 /* Esta función encontrará y mostrará los primos menores que z, donde z >= 1.
*/
 {
   int num;

   num = 1; /* 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. */
```

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

   while (num < z)
    {
      if (es_primo (num) == 1)
       {
         mostrar_primo (num, 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 */

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

void mostrar_primo (int x, int p)

 /* Si el valor de p es 1, esta función mostrará un mensaje diciendo que
    x es primo y si el valor de p es 0, mostrará un mensaje diciendo que
    x no es primo. */

 { /* Inicio de la función */
   if (p == 1)
    {
      printf("\nEl numero %d es primo.", x);
    }
   else
    {
      printf("\nEl numero %d no es primo.", x);
    }
 } /* Fin de la función */

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

6.5 Decidir si n es factorial de algún número

En este ejercicio hay que escribir un programa que pida al usuario un número entero n (≥ 1) y cuando obtenga un n que cumpla esa condición ha de decidir si n es factorial de algún número, es decir, si existe un m tal que m! = n.
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.

- *decidir_factorial*: función que, dado un número entero z ≥ 1, decide si z es factorial de algún número. En caso de que lo sea, mostrará el número y si no, mostrará un mensaje indicando que z no es factorial de ningún número.

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

```
#include <stdio.h>

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

/* Resultados: si n es factorial de algún número, mostrará ese número y si no,
mostrará un mensaje diciendo que no existe ningún número cuyo factorial sea n.
*/

/* Variables:
          - n: Para guardar el número tecleado por el usuario. */

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

int pedir_numero_mayor(int z);
void decidir_factorial (int z);
long calcular_factorial (int x);

/* Función principal*/

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

   n = pedir_numero_mayor (0); /* se quiere un número mayor que 0 */

   decidir_factorial (n); /* decidirá si n es factorial de algún número */

   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 decidir_factorial (int z)

 /* Esta función decidirá si z es factorial de algún número y mostrará un
    mensaje indicándolo. */

 {
   int num;
   long fact;

   if (z == 1) /* Caso particular */
    {
      printf ("\n%d es factorial de 0 y 1.", z);
    }
   else
    {
      num = 2; /* Se pasarán los números de uno en uno a partir de 2. En cada
                  vuelta del while se analizará si z es el factorial del
                  número que está en num. */

      fact = calcular_factorial (num);

      while (fact < z)
       {
         num = num + 1;
         fact = calcular_factorial (num);
       }
      if (fact == z)
       {
         printf("\n%d es el factorial de %d.", z, num);
       }
      else
       {
         printf("\nNo existe ningun numero cuyo factorial sea %d.", z);
       }
    }
 }

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

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 */

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

6.6 Calcular y mostrar los factoriales de los primeros n números primos

En este ejercicio se ha de escribir un programa que pida al usuario un número entero n (≥ 1) y cuando obtenga un número n que cumpla esa condición, ha de calcular y mostrar los factoriales de los n primeros números primos.
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.

- *mostrar_factoriales_de_primos*: función que, dado un entero z, calcula y muestra los factoriales de los primeros z números primos. Esta función ha de llamar a las siguientes funciones: *es_primo*, *calcular_factorial*, *mostrar_primo* y *mostrar_factorial*.

- *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.

- *mostrar_primo*: 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 primo y si p es 0, muestra un mensaje por pantalla indicando que x no es primo.

- *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>

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

/* Resultados: Mostrará por pantalla los factoriales de los primeros números
               primos. */

/* Variables:
          - n: Para guardar el número que tecleará el usuario. */

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

int pedir_numero_mayor (int z);
void mostrar_factoriales_de_primos (int z);
int es_primo (int x);
void mostrar_primo (int x, int p);
long calcular_factorial (int x);
void mostrar_factorial (int x, long f);

/* Función principal */

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

```
   n = pedir_numero_mayor (0);

   mostrar_factoriales_de_primos (n); /* Mostrará los factoriales de los
                                         primeros n números primos */

   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 mostrar_factoriales_de_primos (int z)

 /* Esta función calculará y mostrará por pantalla los factoriales de los
    primeros z números primos. */

 {
   int num, cont;
   long fact;

   num = 1; /* Para pasar los números de uno en uno. */
   cont = 0; /* Para contar cuántos factoriales se han mostrado. */

   while (cont < z)
    {
      if (es_primo (num) == 1)
       {
         mostrar_primo (num, 1);
         fact = calcular_factorial (num);
         mostrar_factorial (num, fact);
         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 contabilizar 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 */

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

void mostrar_primo (int x, int p)

 /* Si el valor de p es 1, esta función mostrará un mensaje diciendo que
    x es primo y si el valor de p es 0, mostrará un mensaje diciendo que
    x no es primo. */

{ /* Inicio de la función */
  if (p == 1)
   {
     printf("\nEl numero %d es primo.", x);
   }
  else
   {
     printf("\nEl numero %d no es primo.", x);
   }
} /* 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 */

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

6.7 Calcular y mostrar los primeros n factoriales primos

En este ejercicio se escribirá un programa que pedirá al usuario un número entero n (≥ 1) y cuando obtenga un n que cumpla esa condición, calculará los primeros n factoriales primos. Funciones a utilizar:

- *preguntar_cuantos_factoriales_primos*: función que pregunta al usuario cuántos factoriales primos quiere, es decir, pide al usuario un número entero mayor o igual que 1. El proceso de petición se repetirá hasta que el usario teclee un número que cumple esa condición.

- *calcular_factoriales_primos*: función que, dado un número entero z (≥ 1), calcula y muestra los primeros z factoriales primos. Esta función ha de llamar a las siguientes funciones: *calcular_factorial*, *es_primo* y *mostrar_factorial*.

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

- *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.

- *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>

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

/* Resultados: Mostrará por pantalla los primeros n factoriales primos. */

/* Variables:
          - n: Para guardar el número que tecleará el usuario.*/

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

int preguntar_cuantos_factoriales_primos (void);
void calcular_factoriales_primos (int z);
long calcular_factorial (int x);
int es_primo (int x);
void mostrar_factorial (int x, long f);

/* Función principal */

void main()
```

```
{ /* Inicio de la función main */
  int n;

  n = preguntar_cuantos_factoriales_primos ();

  calcular_factoriales_primos (n); /* calculará los primeros n factoriales
                                      primos */
  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 preguntar_cuantos_factoriales_primos (void)

 /* Esta función pedirá al usuario un número entero >= 1 y cuando el usuario
    teclee un número que cumpla esa condición, lo devolverá como resultado. Se
    repetirá el proceso de pedir un número hasta obtener uno que cumpla la
    condición. */

{
  int num; /* para guardar el número que tecleará el usuario. */

  do
   {
     printf("\nCuantos factoriales primos quieres? (teclea un numero "
            "entero >= 1): ");
     scanf("%d", & num);
     if (num < 1)
       {
         printf("\nEl numero tecleado no es adecuado.");
       }
   } while (num <= 0);

  return(num);
}

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

void calcular_factoriales_primos (int z)

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

{
  int num, cont, facto;

  num = 0; /* Para pasar los números de uno en uno a partir de 0. En cada
              vuelta del while se calculará el factorial del número que esté
              en num y se mirará si es primo o no. */

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

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

  while (cont < z)
   {
     facto = calcular_factorial (num);
     if (es_primo (facto) == 1)
      {
        mostrar_factorial (num, 1);
        cont = cont + 1;
      }
     num = num + 1;
```

```
    }
 }

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

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 */

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

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 contabilizar 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 */

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

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

6.8 Longitud de una secuencia de números

En este ejercicio hay que escribir un programa que vaya pidiendo de uno en uno los números de una secuencia de enteros no negativos (≥ 0) y al final muestre un mensaje indicando cuántos números hay en la secuencia.
El último número de la secuencia será negativo y no hay que contabilizarlo. Se considerará que una secuencia es vacía si el primer número es negativo.
Funciones a utilizar:

- *pedir_contar_mostrar*: función que va pidiendo los elementos de la secuencia de enteros de uno en uno y al final muestra un mensaje diciendo cuántos elementos hay. Esta función ha de llamar a las siguientes funciones: *pedir_elemento_secuencia* y *mostrar_cantidad*.

- *pedir_elemento_secuencia*: función que, dado un número entero z, pide al usuario que teclee el z-avo número de la secuencia. La función devolverá como resultado el número tecleado por el usuario.

- *mostrar_cantidad*: función que, dado un número entero c, muestra un mensaje indicando que en la secuencia hay c elementos.

```
#include <stdio.h>

/* Datos: Este programa irá pidiendo de uno en uno al usuario los elementos de
          una secuencia de enteros. El proceso terminará cuando el usuario
          teclee un número negativo. */

/* Resultados: El programa mostrará un mensaje indicando cuántos elementos
               hay en la secuencia tecleada por el usuario. */

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

void pedir_contar_mostrar (void);
int pedir_elemento_secuencia (int z);
void mostrar_cantidad (int c);

/* Función principal */

void main()
 { /* Inicio de la función main */

   pedir_contar_mostrar();

   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 */

void pedir_contar_mostrar (void)

 /* Esta función irá pidiendo de uno en uno, al usuario, los elementos de una
    secuencia de enteros y al final mostrará un mensaje diciendo cuántos
    elementos hay en la secuencia. Todos los números han de ser no negativos
    (>= 0) salvo el último.*/
```

```
{
  int numero, cont;

  cont = 1; /* Esta variable nos indica la posición del número que se va a
               pedir. Al principio vale 1 porque se va a pedir el primero. */

  numero = pedir_elemento_secuencia (cont);

  while (numero >= 0)
   {
     cont = cont + 1;
     numero = pedir_elemento_secuencia (cont);
   }

  /* El último número obtenido no se ha de contar porque es negativo */

  mostrar_cantidad(cont - 1);
}

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

int pedir_elemento_secuencia (int z)

 /* Esta función pide al usuario el z-avo número de la secuencia. */

 {
   int num; /* Para guardar el número tecleado por el usuario.*/

   printf("\nTeclea el %d-avo numero de la secuencia: ", z);
   scanf("%d", & num);

   return(num);
 }

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

void mostrar_cantidad (int c)

 /* Esta función mostrará un mensaje indicando que en la secuencia hay c
    elementos. */

 {
   printf("\n En la secuencia hay %d elementos.", c);
 }

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

6.9 Máximo y mínimo de una secuencia de enteros

En este ejercicio hay que escribir un programa que vaya pidiendo de uno en uno los números de una secuencia de enteros no negativos (≥ 0) y al final muestre el máximo y el mínimo de la secuencia.
El último número de la secuencia será negativo y no hay que tenerlo en cuenta. Se considerará que una secuencia es vacía si el primer número es negativo.
Si la secuencia es vacía habrá que mostrar un mensaje indicándolo ya que en ese caso no será posible calcular el máximo y el mínimo.
Funciones a utilizar:

- *calcular_max_min_secuencia*: función que pide los elementos de la secuencia de uno en uno y calcula y muestra el máximo y el mínimo. Esta función ha de utilizar las siguientes funciones: *pedir_elemento_secuencia*, *mostrar_max_min* y *secuencia_vacia*.

- *pedir_elemento_secuencia*: función que, dado un número entero z, pide al usuario que teclee el z-avo número de la secuencia. La función devolverá como resultado el número tecleado por el usuario.

- *mostrar_max_min*: función que, dados dos enteros ma y mi, muestra un mensaje diciendo que ma es el máximo y mi es el mínimo de la secuencia.

- *secuencia_vacia*: función que muestra un mensaje indicando que la secuencia es vacía.

```
#include <stdio.h>

/* Datos: Este programa irá pidiendo de uno en uno al usuario los elementos de
          una secuencia de enteros. El proceso terminará cuando el usuario
          teclee un número negativo. */

/* Resultados: Mostrará un mensaje indicando cuáles son el máximo y el mínimo
de
               la secuencia. */

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

void calcular_max_min_secuencia (void);
int pedir_elemento_secuencia (int z);
void mostrar_max_min (int ma, int mi);
void secuencia_vacia (void);

/* Función principal */

void main()
 { /* Inicio de la función main */

   calcular_max_min_secuencia();

   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 */


void calcular_max_min_secuencia (void)

 /* Esta función irá pidiendo de uno en uno los elementos de una secuencia de
    números enteros, decidirá cuál es el máximo y cuál el mínimo y al final
    los mostrará. Todos los elementos de la secuencia han ser >= 0 salvo el
    último que será negativo. */

 { /* Inicio de la función */
   int numero, mayor, menor, cont;

   numero = pedir_elemento_secuencia (1);

   if (numero < 0)
    {
      secuencia_vacia();
    }
   else
    {
      cont = 1; /* Para contar cuántos números se han pedido. */
      mayor = numero;
      menor = numero; /* Al principio el primer número es tanto el mayor
                         como el menor. */
```

```
      numero = pedir_elemento_secuencia (cont + 1); /* para obtener el
                                        segundo número de la secuencia */

      while (numero >= 0)
       {
         cont = cont + 1;

         if (numero > mayor)
          {
            mayor = numero;
          }
         else
          {
            menor = numero;
          }
         numero = pedir_elemento_secuencia (cont + 1);
       }
      mostrar_max_min (mayor, menor);
    }
 } /* Fin de la función */

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

int pedir_elemento_secuencia (int z)

 /* Esta función pide al usuario el z-avo número de la secuencia. */

 {
   int num; /* Para guardar el número tecleado por el usuario. */

   printf("\nTeclea el %d-avo numero de la secuencia: ", z);
   scanf("%d", & num);

   return(num);
 }

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

void mostrar_max_min (int ma, int mi)

 /* Esta función muestra un mensaje indicando que ma es el máximo y que mi es
    el mínimo de la secuencia. */

 {
   printf("\nEl maximo de la secuencia es %d y el minimo es %d.", ma, mi);
 }

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

void secuencia_vacia (void

 /* Esta función muestra un mensaje por pantalla indicando que la secuencia de
    números introducida por el usuario es vacía. */

 { /* Inicio de la función */

   printf("\nLa secuencia de numeros es vacia.");

 } /* Fin de la función */

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

7 NOTA SOBRE LOS PARÁMETROS QUE SON DATOS DE ENTRADA

Los parámetros que son datos de entrada no conviene modificarlos dentro de la función.

Ejemplo 1:

Supongamos que queremos escribir una función que, dados dos números enteros x e y ($x \leq y$), muestra los números comprendidos entre x e y.

Se van a dar dos versiones:

- En la primera versión el valor de x irá cambiando dentro de la función. Esta versión no es aconsejable o correcta.
- En la segunda versión el valor de x no cambiará dentro de la función. Esta versión es la correcta.

Versión no correcta (x va cambiando):

```
void mostrar_numeros_del_intervalo (int x, int y)

 /* Esta función muestra los números comprendidos entre x e y. Si x es mayor
    que y, no se mostrará ningún número. */

 { /* Inicio de la función */

   while (x <= y)
    {
      printf("\n%d", x);
      x = x + 1;
    }

   /* Esta versión no es correcta porque el valor del parámetro de entrada (o
      dato) x es modificado dentro de la función. */ }
```

Versión correcta (x no cambia):

```
void mostrar_numeros_del_intervalo (int x, int y)

 /* Esta función muestra los números comprendidos entre x e y. Si x es mayor
    que y, no se mostrará ningún número. */

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

   num = x; /* para pasar de uno en uno los números comprendidos entre x e y
               */

   while (num <= y)
    {
      printf("\n%d", num);
      num = num + 1;
    }

   /* Esta versión es correcta porque el valor del parámetro de entrada (o
      dato) no es modificado dentro de la función. Para conseguir eso se ha
      utilizado otra variable: num. */
 }
```

Ejemplo 2:

A continuación se va a dar otro ejemplo. Supongamos que queremos escribir una función que, dados dos enteros x e y ($x \leq y$), calcula de manera iterativa el número de veces que x puede ser divido por y.
Se van a presentar dos versiones:

- En la primerá se irá cambiando el valor de x dentro de la función. Esta versión no es correcta.
- En la segunda, x no cambiará dentro de la función. Esta versión es correcta.

Versión incorrecta (x va cambiando):

```
int numero_divisiones (int x, int y)

 /* Esta función calcula de manera iterativa el número de veces que x puede
    ser dividido por y, devolviendo al final ese resultado. */

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

   cont = 0; /* Esta variable será utilizada para ir contando cuántas veces es
                posible dividir x por y. */

   while (x % y == 0)
    {
      cont = cont + 1;;
      x = x / y;
    }

   return(cont);

   /* Esta versión no es correcta porque el valor del parámetro de entrada (o
      dato) x es modificado dentro de la función. */
 } /* Fin de la función */
```

Versión correcta (x no cambia):

```
int numero_divisiones (int x, int y)

 /* Esta función calcula de manera iterativa el número de veces que x puede
    ser dividido por y, devolviendo al final ese resultado. */

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

   cont = 0; /* Esta variable servirá para ir contando el número de veces que
                x es divisible por y. */

   aux = x; /* Los cambios y las operaciones se realizarán sobre aux y no
               sobre x. */

   while (aux % y == 0)
    {
      cont = cont + 1;;
      aux = aux / y;
    }

   return(cont);

   /* Esta versión es correcta porque el valor del parámetro de entrada (o
      dato) x no es modificado dentro de la función. Para conseguir eso se ha
      utilizado otra  variable: aux */

 } /* Fin de la función */
```