ESCUELA POLITÉCNICA SUPERIOR

INFORMÁTICA – CURSO 2024-25

PRÁCTICA 5. FUNCIONES

HASTA AHORA...

En prácticas anteriores se ha aprendido:

- La estructura principal de un programa en C: la función `main` y las librerías.
- Variables y Constantes: tipos, usos y funcionamiento.
- Funciones `printf/scanf`: visionado/obtención de información por pantalla/teclado.
- Estructura condicional `if/else`.
- Estructuras iterativas `while y for`.
- Vectores, matrices y cadenas.

OBJETIVOS

Durante esta práctica se realizará una introducción al uso de las **funciones en C.** Las funciones permiten desarrollar software **reutilizable** de forma **modular** y son de gran ayuda porque permiten el desarrollo de programas "grandes" de una manera más sencilla y eficiente. A partir de esta práctica se hará énfasis en estructurar en varias funciones los programas que se desarrollen durante las prácticas.

PRERREQUISITOS

Para el correcto desarrollo de esta práctica, el alumno ANTES DE LA SESIÓN DE PRÁCTICA EN EL LABORATORIO, debe leer con atención y comprender el contenido de la parte teórica de dicha práctica, el apartado FUNCIONES de este documento. Además, el alumno habrá trabajado previamente los cuatro primeros ejercicios descritos en el apartado EJERCICIOS de este documento.

Se aconseja el uso del material de apoyo de la asignatura. Así, para esta práctica sobre funciones el alumno puede consultar la siguiente bibliografía:

- Rodríguez Jódar, M.A. y otros, "Fundamentos de informática para ingeniería industrial":
  - Del *Capítulo 4, Fundamentos de programación*: apartado 4.5, *Diseño descendente.*
- García de Jalón de la Fuente, J. y otros, "Aprenda lenguaje ANSI C como si estuviera en primero":
  - Apartado 1.3, *Concepto de función.*
  - Del *Capítulo 7, Funciones:* apartados 7.1, 7.2, 7.3.

1.- USO DE FUNCIONES PREDEFINIDAS

Descripción

El concepto de **función** en programación es similar al de función matemática: la función, identificada con un determinado nombre, puede recibir una serie de valores de entrada y generar

un determinado valor de salida que es resultado de las operaciones que tienen lugar en su interior.

En nuestro caso, la función será un **conjunto de instrucciones** etiquetadas con un nombre (nombre de la función) y delimitadas por un par de llaves **{ }**. La función actúa como una **"caja negra"** que encapsula una determinada funcionalidad, realiza ciertas acciones de una forma independiente al resto del código.

Nuestros programas hasta ahora se han compuesto de una sola función: `main()`. La función `main` es la primera que se ejecuta y que llama a las otras funciones directa o indirectamente. También hemos usado otras funciones ya definidas por C: `printf()`, `scanf()`, `sqrt()`, etc. Estas funciones se encuentran englobadas en **librerías** cuyo objetivo es facilitar la programación, al proporcionar funcionalidades comunes.

- **Librería math.h**

La librería *math* contiene un conjunto de funciones estándar del lenguaje de programación C diseñado para operaciones matemáticas básicas. A continuación, se muestran algunas de las funciones más utilizadas de esta librería (algunas de estas ya han sido utilizadas en prácticas anteriores).

Nombre de la función	Cometido
sqrt(x)	Raíz cuadrada de x
pow(x, y)	Eleva el valor x a la potencia y
abs(x)	Calcula el valor absoluto de x
cos(x)	Calcula el coseno de x (x expresado en radianes)
sin(x)	Calcula el seno de x (x expresado en radianes)
exp(x)	Calcula la exponencial de x (e^x)
floor(x)	Calcula el número entero menor o igual que x
ceil(x)	Calcula el número entero mayor o igual que x

Ejemplo de la función **pow**:

```
#include <stdio.h>
#include <math.h>
void main()
{
       double x = 6.5, y = 0.5;
       // Soporta long, int, double y float, pero el resultado de
      pow siempre sera double
       int b= 2, e=5;
       printf("base %f y exponente %f = %.2f ", x, y, pow(x,y) );
       printf("base %i y exponente %i = % .2f ", b, e, pow(b,e) );
}
```

Ejemplo de la función **cos**:

```
#include <stdio.h>
#include <math.h>
void main()
{
	double resul;
	double x = 0.5;
	resul = cos(x);
	printf("\n El coseno de %lf es %lf", x, resul);
}
```

Ejemplo de las funciones **floor** y **ceil**:

```
#include <math.h>
#include <stdio.h>
void main()
{
	double num = 211.84;
	double baja, sube;
	baja = floor(num);
	sube = ceil(num);
	printf("Número original %f", num);
	printf("Número redondeado con floor %lf", sube);
	printf("Número redondeado con ceil %lf", baja);
}
```

- **Librería string.h**

Si queremos manipular cadenas de forma mucho más simple, se puede hacer uso de una serie de funciones específicas contenidas en la librería *string*. En la siguiente tabla resumimos las funciones más importantes que posee dicha librearía, así como su funcionamiento:

Nombre de la función	Cometido	Valor de salida (resultado)
strlen(cadena)	Calcula el número de caracteres de la cadena	*int* Longitud de la cadena (sin contar el '\0')
strcmp(cadena1, cadena2)	Comparación de ambas cadenas (usando el código ASCII)	*int* Valor<0 si cad1<cad2 0 si cad1==cad2 Valor>0 si cad1>cad2
strcpy(cadena1, cadena2)	Copia el contenido de la cadena2 a la cadena1. Origen: derecha. Destino: izquierda. Se sobrescribe lo que tuviera	Nada*
gets(cadena)	Lectura por teclado de una cadena de caracteres.	Nada*

**Nota:* algunas de las funciones aquí indicadas no son exactamente tal como se detalla, pero, por motivo de simplicidad y por el hecho de mostrar sólo aquellos aspectos que vamos a utilizar, se muestran con leves variaciones.*

Ejemplos:

1. **strlen**

```
…
int num;
char cad[256];
printf("Introduzca una cadena:   ");
scanf("%s", cad);
num = strlen(cad);
printf("El tamanyo de la cadena es de %d caracteres\n", num);
…
```

Nota: el hecho de que la cadena se cree con un máximo de 256 caracteres no implica que strlen vaya a devolver un valor de 256. Contabilizará el número de caracteres antes del fin de cadena.

2. **strcmp**

```
int a;
char cad1[256]="hola", cad2[256]="caracola";
a = strcmp(cad1, cad2);
//a contiene un valor mayor que 0, ya que cad1>cad2 alfabéticamente
```

3. **strcpy**

```
char cad1[256]="hola", cad2[256]="caracola";
strcpy(cad1, cad2); //En cad1 se copia el contenido de cad2, eliminando lo que tuviera
```

4. **gets**

```
char cad[256];
gets(cad); //Pide por teclado una cadena y la almacena en cad.
```

DIFERENCIA ENTRE SCANF Y GETS

Se han explicado dos formas de recoger una cadena de caracteres desde teclado: haciendo uso de la nueva función **gets** o, siguiendo con lo explicado en prácticas anteriores, utilizar la función **scanf**; en este caso con el identificador "%s".

¿Cuál es la diferencia entre ambas?

- **scanf** recoge la cadena de caracteres introducida por el usuario hasta que se encuentra un intro, un tabulador o un espacio.
- **gets** recoge la cadena de caracteres hasta intro.

De esta forma, si se va a hacer uso de cadenas de caracteres con espacios, solo se podrá hacer uso de la función **gets**, puesto que **scanf** no almacenará todo lo que haya detrás del espacio.

EJERCICIOS

Ejercicio 1: Escriba un programa que pida un valor de tipo real por teclado (float), y a continuación, escriba en pantalla el valor de su raíz cúbica, antilogaritmo natural (exponencial), seno y coseno. Recuerda que expresiones tales como 1/3 asumen que tanto el 1 como el 3 son enteros, por tanto, se realiza una división entera.

Ejercicio 2: Ejecute el siguiente código:

```
#include <stdio.h>
#include <math.h>
int main()
{
      float i,j,I,J;

      i=5.4;
      j=5.6;
      I = round(i);
      J = round(j);

      printf("round of  %f is  %f\n", i, I);
      printf("round of  %f is  %f\n", j, J);
}
```

¿Qué objetivo tiene la función `round()`? ¿Qué diferencia hay con respecto a las funciones `floor()` y `ceil()`?

Ejercicio 3: Escriba un programa que pida por teclado tres valores que correspondan a las longitudes de los tres lados de un triángulo (variables float a, b, c). El programa debe comprobar si con esos tres lados es posible construir o no un triángulo. La condición que deben cumplir a, b, c para ser un triángulo es que la suma de cualesquiera dos lados debe ser mayor que el tercer lado. Usando la fórmula de Heron (mostrada a continuación), calcule su área en caso de que los lados puedan formar un triángulo, y escriba un mensaje tal como "Se puede formar un triángulo y su área es" (y aquí vendría el área calculada). Si no pueden formar triángulo, tan solo debe mostrar el mensaje "No pueden formar triángulo\n".

La fórmula de Herón es: $S = \sqrt{\mathrm{p(p-a)(p-b)(p-c)}}$, donde ha de calcularse previamente el valor p=(a+b+c)/2.

Ejercicio 4: Escriba un programa que muestre por pantalla si un número (recogido por teclado) es potencia de 2 utilizando la función pow(). Para ello, siga los siguientes pasos:

1. El programa debe pedir al usuario que ingrese un número entero positivo.
2. Para determinar si el número es una potencia de 2, el programa debe calcular sucesivas potencias de 2 (es decir $2^0, 2^1, 2^2 \ldots, 2^i$) hasta que esta potencia alcance o supere el número introducido por teclado.
3. En cada iteración del bucle, el programa debe usar la función pow(2, i) para calcular 2^i, donde i es el exponente que se incrementa en uno en cada iteración.
4. En cuanto se alcance o se supere el número recogido por teclado, el programa debe verificar si 2^i es exactamente igual a este número. Si lo es, entonces el número es una potencia de 2; si no, el número no es una potencia de 2.

Ejercicio 5: Vamos a calcular el tamaño de una determinada cadena introducida por parámetros:

- **a)** Cree una cadena de caracteres de tamaño 256.
- **b)** Haciendo uso de la función ***scanf*** y del especificador "%s", recoja una cadena por teclado y asígnesela a la cadena creada anteriormente. Muéstrela por pantalla utilizando la función ***printf***.
- **c)** Haga lo mismo que en "*b*", pero utilizando ***gets*** en vez de ***scanf***. Pruebe la diferencia entre "*b*" y "*c*" utilizando una cadena con espacios.
- **d)** Haciendo uso de la función ***strlen*** de la librería ***string.h***, muestre por pantalla el tamaño de la cadena de caracteres que acaba de introducir.

Ejercicio 6: Escriba un programa que introduzca por teclado un mensaje de texto y una letra (carácter). El programa devolverá el número de veces que aparece dicho carácter en el mensaje (utilice la función ***strlen***).

Ejercicio 7: Escriba un programa que introduzca por teclado un mensaje de texto. El programa deberá contar el número de letras mayúsculas que contenga el mensaje (utilice la función ***strlen***).

Ejercicio 8: Escriba un programa que recoja por teclado una cadena de caracteres, cuyo contenido representa un código de activación para una aplicación (puede tener dígitos y/o letras). El programa debe mostrar por pantalla 1 si el código de activación es válido, y 0 si no lo es. Los requisitos para que un código de activación sea válido son los siguientes (deben cumplirse todos):

- El código de activación debe tener al menos 8 caracteres.
- El código de activación no puede contener la letra 'B'.
- La suma de los valores de los dígitos (no de sus códigos ASCII*) que haya en el código de activación debe ser múltiplo de 6.

Ejemplo: la cadena "Azu6w6GhHZ" es un código de activación válido.

* **Nota**: Para convertir un carácter que representa un dígito (por ejemplo, '5') a su valor numérico correspondiente (es decir, el número 5), se puede usar la propiedad de los caracteres en ASCII. Los caracteres de los dígitos del 0 al 9 tienen valores consecutivos en la tabla ASCII, donde el valor de '0' es 48, '1' es 49, y así sucesivamente. La manera más sencilla es restar el valor ASCII del carácter '0' al carácter del dígito -> *valor = digito – '0'*; siendo *valor* una variable entera, y *digito* una variable de tipo carácter, o un elemento de una cadena.

Ejercicio 9: Escriba un programa que recoja por teclado una cadena de caracteres que contenga una serie de palabras escritas en minúsculas, y separadas por espacios (un espacio entre cada palabra). El programa debe escribir en pantalla las iniciales de cada palabra, en mayúsculas.

Ejemplo: para la cadena "escuela politecnica superior", el programa debe mostrar en pantalla: EPS

Ejercicio 10: Cree un programa que solicite al usuario una serie de cadenas de caracteres (hasta que se introduzca la cadena "fin") y que, tras finalizar, muestre el número de caracteres de la cadena más larga de entre las introducidas. Para ello, puede usar las funciones ***strlen*** **y** ***strcmp***. Amplie el ejercicio para que además muestre por pantalla la cadena más larga de todas las introducidas (utilice la función ***strcpy***).

2.- FUNCIONES DEFINIDAS POR EL USUARIO

En este apartado, aprenderemos a crear nuestras propias funciones y usarlas en cualquier punto de nuestro programa. Mediante el uso de funciones agrupamos un conjunto de instrucciones que realizan una tarea y las dotamos de un nombre a partir del cual puede invocarse dentro de nuestro programa. De esta forma, si realizamos una función capaz de calcular la distancia entre dos puntos, podremos usarla tantas veces como necesitemos dentro de nuestro programa sin tener que duplicar código: escribiríamos una única vez el código que realiza el cálculo de la distancia y este código sería invocado desde distintos puntos del programa cada vez que el cálculo de una distancia fuese necesario.

La función, al actuar como una "caja negra", no tiene conocimiento de lo que "ocurre" en el resto del programa. La única forma a través de la cual la función puede recibir valores desde el "exterior", es decir, desde otras partes del programa desde donde se invoca a través de su nombre, es a través de lo que se denominan **parámetros o argumentos de entrada**. A su vez, la única forma que tiene la función de comunicar y devolver valores al exterior es a través de su **valor devuelto o resultado**. Más adelante se describirán con más detalles estos elementos.

Declaración: prototipo

El primer paso para crear una función es realizar su **declaración**, también denominado como **definir el prototipo de la función**. El prototipo puede considerarse como una breve presentación de la función: indica su nombre, el tipo de los datos que necesita que se le proporcionen para poder llevar a cabo su tarea y también el tipo de dato del resultado que devuelve la función. De esta forma, el prototipo contiene toda la información necesaria para usar la función, sin entrar en detalles de cómo está hecha por dentro.
La estructura básica del prototipo es la siguiente:

```
<tipo> nombre_de_funcion (<lista_argumentos>);
```

Donde:

- `tipo`: declara el tipo del valor que devuelve la función (`void`, conocido como tipo vacío, si no devuelve ningún valor). De esta forma se indica que la función **devolverá un valor o resultado** del tipo indicado.
- `nombre_de_funcion`: identificador de la función. Sigue las mismas reglas de nombrado que los identificadores de variables y estructuras.
- `lista_argumentos`: Lista de **argumentos o parámetros de entrada**. Es una lista detallada de los datos que la función recibe desde el "exterior" al ser invocada. Para cada uno de estos parámetros se especifica su tipo de dato. Los argumentos pueden considerarse como un tipo especial de variable declarada como parte intrínseca de la propia función y que reciben su valor cuando se realiza una llamada a la función pasándole una serie de valores como parámetro.
- El prototipo siempre finaliza con el carácter punto y coma "`;`"

El prototipo de la función se situará a continuación de las etiquetas `#include` y `#define` de nuestro programa, y antes de comenzar la función `main()`. La estructura de nuestros programas con funciones será la siguiente:

```
#include <stdio.h>
#define CONSTANTE valor_constante

// prototipos de funciones
tipo nombre_funcion1(lista_argumentos);
tipo nombre_funcion2(lista_argumentos);
…
```

```
// función main
main(){
            // declaración de variables
            // instrucciones

}
```

Algunos ejemplos de prototipos de funciones serían los siguientes:

```
int esBisiesto(int a);
float parabola(float a, float b, float c, float x);
```

El primer prototipo define una función que devuelve un valor de tipo entero (`int`), su nombre es `esBisiesto` y recibe un parámetro de entrada de tipo entero (`int a`). El segundo prototipo define una función que devuelve un valor de tipo real (`float`), su nombre es `parabola` y recibe como parámetro de entrada cuatro valores de tipo real: `float a, float b, float c, float x.`

Definición: implementación de la función

El prototipo sólo nos proporciona una descripción básica de la función, indicándonos los datos necesarios para poder usarla e interactuar con ella, pero aún es preciso escribir las instrucciones en lenguaje C que definen qué va a realizar la función. Especificar mediante líneas de código lo que va a hacer una determinada función se denomina "**implementar la función"**.

La implementación de una función presenta la siguiente estructura:

```
tipo nombre_funcion(lista_argumentos){
      // declaración de variables
      // instrucciones
      // devolver resultado
      return <valor>;
}
```

La primera línea es idéntica a nuestro prototipo, con la diferencia que en lugar de finalizar con un punto y coma (`;`), se abre una llave (`{`) que indica dónde comienza la función. Se usará un cierre de llave (`}`) para indicar dónde acaba la función. En esta primera línea se define una lista de argumentos cuya declaración es similar al de las variables. Cada argumento tendrá un tipo y un nombre y podrán usarse dentro de la función como si de variables se tratase. El valor inicial de esas variables se establece cuando se realiza la llamada a la función desde algún punto del programa, por tanto, la inicialización de los valores de los parámetros se realiza fuera de la función, cuando ésta es invocada desde algún punto del programa. En el interior de la función la estructura es similar a la que hemos usado dentro de la función `main()`: en primer lugar se declaran las variables necesarias para llevar a cabo los cálculos y operaciones que realizará la función. Finalmente, si la función ha de devolver algún valor, se utiliza la instrucción *`return`*, que transmite a la función que realizó la llamada el valor de retorno indicado.

A continuación, se muestra como ejemplo la declaración e implementación de la función `esBisiesto`, la cual recibe como parámetro de entrada un valor entero, el cual se corresponde con un año, y devuelve el valor entero 1 (cierto) si el año es bisiesto o 0 (falso) en caso de que no lo sea.

```
#include <stdio.h>

// prototipo de la función
int esBisiesto(int anyo);
```

```
main(){
      // instrucciones de la función main
}

// implementación de la función
int esBisiesto(int anyo){
      int resultado = 0;

      if( (anyo % 4 == 0) && (anyo % 100 != 0) ){
            // multiplo de 4 y no de 100, es bisiesto
            resultado = 1;
      }else if(anyo % 4 == 0 && anyo % 400 == 0){
            // multiplo de 4 y de 400, es bisiesto
            resultado = 1;
      }else{
            // no es bisiesto
            resultado = 0;
      }
      // se lanza al exterior o devuelve el resultado
      return resultado;
}
```

Invocación: llamada a la función

Se entiende como invocación al uso de una función en algún punto determinado de nuestro programa, para lo cual se indica el nombre de la función que queremos ejecutar y se le pasan como parámetro los valores que necesita dicha función para poder llevar a cabo su misión. Así pues, la invocación de funciones en C se realiza especificando su nombre seguido de la lista de valores de entrada que recibe entre paréntesis:

```
      // nombre de la función: printf
      // valor / argumento de entrada: "Hola Mundo\n"
      printf("Hola Mundo\n");
```

El valor devuelto por una función puede almacenarse en una variable usando la asignación:

```
y = f(x);
var1 = sqrt(var2);
```

A continuación, se muestra un ejemplo en el que usamos la función `esBisiesto` para determinar si el año introducido por teclado es o no bisiesto:

```
#include <stdio.h>
// prototipo de funciones
int esBisiesto(int); //declaración de la función

// funcion pricipal main
main()
{     // declaración de variables
      int year, resultado;

      // pedir anyo por teclado
      printf("Introduzca un anyo: \n");
      scanf("%d",&year);

      //invocación de la función
      resultado = esBisiesto(year);
```

```
      // mostrar resultados por pantalla
      if(resultado == 1){
           printf("Es Bisiesto\n");
      }else{
           printf("No es Bisiesto\n");
      }
}
//implementación de la función
int esBisiesto(int anyo)
{
      int resultado;
      // implementación de la función (descrita anteriormente)
      ...
      // devolver resultado
      return resultado;
}
```

Nótese que, al invocar la función, le pasamos el valor contenido en la variable `year`. Cuando se invoca, el nombre que se le ha dado a los argumentos de la función es indiferente, sólo hay que tener en cuenta que durante la invocación, el primer valor pasado por parámetro se copiará en el primer argumento, y así sucesivamente. De esta forma, el valor de la variable `year` se copia dentro del argumento `anyo`. Dentro de la función, sólo podremos trabajar con este argumento `anyo`.

***ANEXO 1*: PASO DE PARÁMETROS POR REFERENCIA**

Hasta ahora, las funciones que hemos visto recibían parámetros, pero sólo consultaban sus valores para realizar cálculos u operaciones internas. Es decir, pasábamos *los parámetros por valor*. Para los vectores y matrices nos interesa alterar los valores de sus elementos y que cuando termine la función se queden estos elementos modificados, por eso se dice que para los vectores y matrices se *pasan los parámetros por referencia.*

En el lenguaje C una función sólo puede devolver un valor (usando la sentencia *return*), pero mediante el paso de *parámetros por referencia*, puede modificar los valores de un vector dentro del cuerpo de la función y así "devolver" todo el contenido que este tipo de datos almacenan.

Ejemplo de una función que tiene parámetros por referencia:

Crear una función llamada **copiarVector** que reciba dos vectores de enteros y el tamaño de los mismos (deben ser del mismo tamaño) y que consiga copiar en el segundo vector el contenido del primero.

```
#include <stdio.h>
#define TAM 10

void copiaVector(int [], int[], int);

main()
{
int a[TAM]= {1,2,3,4,5,6,1,2,3,4}, b[TAM];
int i;
copiaVector(a, b, TAM);
```

```
for(i=0;i<TAM;i++)
          printf("%d ", b[i]);

}

void copiaVector(int vect1[], int vect2[], int t)
{
int i;
for(i=0;i<t;i++)
  vect2[i]=vect1[i]; // cuando termine la función, vect2 tiene los
                            // mismos valores de vect1

}
```

EJERCICIOS:

Ejercicio 1: Realizar una función que reciba dos sumandos y devuelva el resultado de la suma de ambos (todos de tipo entero). Para ello, por partes:

- **a)** Defina el prototipo de la misma.
- **b)** Escriba la implementación. Hágalo de dos formas: primero, mostrando un mensaje por pantalla y, posteriormente, devolviendo el resultado.
- **c)** Escriba la función *`main`* del programa donde llame a dicha función y muestre el resultado (para los dos casos anteriores).

Ejercicio 2: Analice el siguiente código **sin utilizar ZinjaI** e indique qué se mostraría por pantalla una vez ejecutado:

```
#include <stdio.h>
int calculaPares(int p1, int p2);
int main()
{
      int a = 2, b = 6, par;
      par = calculaPares(a, b);
      printf("par - %d\n", par);
}

int calculaPares(int p1, int p2)
{
      int maxPar, i;
      for( i= p1; i < p2; i++)
      {
            if(i%2 == 0)
                        {
                        printf("%d\n", i);
                        maxPar = i;
                        }
      }
      return maxPar;
}
```

Ejercicio 3: Realizar un programa que, con el radio de una circunferencia, calcule tanto su área como su longitud. Para ello:

a) Cree una única función, que recibirá el radio como parámetro. Realizará el cálculo del área y la longitud y, al no poder devolver dos valores a la vez, los mostrará por pantalla. La función, para este caso, no devuelve ningún valor. La función *`main`* será la encargada de solicitar por teclado el radio de la circunferencia e invocar a la función pasándole como parámetro dicho valor del radio. El prototipo de la función será:

```
void calculaAreaYLongitud(float radio);
```

a) Se dividirá el problema en dos funciones, donde cada una de ellas calculara uno de los dos resultados; de esta forma se podrán devolver. Una de ellas calculará únicamente el área de la circunferencia asociada y la otra calculará su longitud. Estas funciones recibirán el radio de la circunferencia, calcularán el resultado y lo devolverán mediante la instrucción `return`, en ningún momento se encargarán de mostrar ningún resultado por pantalla. La función `main()` será la encargada de solicitar por teclado el radio, invocar a ambas funciones, recoger los resultados que calculen y mostrarlos por pantalla. Los prototipos de ambas funciones serán:

```
float calculaArea(float radio)
float calculaLongitud(float radio)
```

Ejercicio 4: Implemente una función que reciba por parámetros un valor de tipo entero. Esta función mostrará por pantalla un mensaje indicando si el número es par o impar. Para ello:

a) Defina el prototipo. Recuerde que hay que especificar "void" si no devuelve nada. Por ejemplo:

```
void esPar (int num);
```

b) Escriba la implementación (debajo de la función principal).
Nota: haga uso del operador "%", que indica el resto de la división entera.

c) Escriba la función principal del programa (*main*). En ella se solicita un valor entero por teclado y se hace la llamada a la función implementada anteriormente pasándole por parámetros el valor recogido. Realice pruebas para comprobar que funciona correctamente.

Ejercicio 5: Implemente una función que reciba por parámetros un valor de tipo entero. Esta función mostrará por pantalla TODOS los números pares desde el 0 hasta el valor recibido por parámetros; así pues, lo utilizará como límite de esa sucesión de números. Siga los siguientes pasos:

a) Defina el prototipo teniendo en cuenta que la función no devuelve nada (debe indicarse tipo "void"). Podría ser, por ejemplo:

```
void muestraPares(int numMax);
```

b) Escriba la implementación (debajo de la función principal).

c) Escriba la función principal del programa (*main*). Aquí se pide por teclado un valor entero, se almacena en una variable de tal tipo, y se hace la llamada a la función creada anteriormente pasándole por parámetros el valor recogido.

Ejercicio 6: Implemente una función que reciba como parámetros un número entero, calcule su factorial (ejercicio 8 de la práctica 3) y lo devuelva. Para ello:

a) Defina el prototipo:

```
int factorial (int x);
```

b) Implemente la función. El resultado debe devolverlo (NO mostrarlo por pantalla).

c) Cree la función *main*, que recoja un número entero por teclado, llame a la función, almacene el valor devuelto en una variable y la muestre por pantalla.

Algunos valores de ejemplo para comprobar que funciona correctamente:

7! = 5040 10! = 3628800 0! = 1

Ejercicio 7: Usando la **función creada en el ejercicio anterior**, implemente otra función que calcule y devuelva el número combinatorio de dos números enteros. El número combinatorio $\binom{a}{b}$ se define como $\frac{a!}{b!(a-b)!}$. Para ello:

a) Defina el prototipo:

```
int comb (int a, int b);
```

b) Implemente la función usando la función `factorial()` para calcular el factorial de a, el factorial de b y el factorial de a-b.

c) Cree la función *main* que recoja dos números enteros por teclado, llame a la función `comb()`, almacene el valor devuelto en una variable y la muestre por pantalla.

Algunos valores de ejemplo para comprobar que funciona correctamente:

$\binom{0}{0} = 1$ $\binom{3}{1} = 3$ $\binom{6}{4} = 15$ $\binom{10}{6} = 210$

Ejercicio 8: Desarrolle una función que devuelva un número de tipo *float* comprendido dentro de un rango concreto. Para ello:

a) Haga uso del prototipo indicado a continuación. Fíjese que se reciben dos valores reales por parámetros, que indicarán los límites de dicho rango.

```
float lee_numero(float vmin, float vmax);
```

b) Implemente la función para que pida por teclado un valor de tipo real y compruebe que dicho valor se encuentra dentro del rango determinado por los dos parámetros recibidos. Si el valor recogido por teclado se encuentra dentro del rango, se devuelve; si dicho valor sale del rango, se vuelve a solicitar dicho valor. Esta operación se repetirá hasta que el valor recogido se encuentre dentro del rango.

c) Programe la función *main*, para que pida por teclado dos valores reales (límites del rango), realice la llamada a la función y recoja el resultado de la llamada. Finalmente, muestre por pantalla el valor resultante.

Ejercicio 9: Realizar un programa que realice lo siguiente:

a) Implementar una función que reciba dos parámetros de entrada: una nota de prácticas y otra de teoría y calcule la nota final resultante. Esta nota se divide en un 70% de nota de prácticas y un 30% de nota de teoría. Su prototipo será `float calcularNota(float teoria, float practica);`

b) Implementar otra función que reciba el valor numérico de la nota final y muestre por pantalla un mensaje de texto que indique si el valor se corresponde con "SUSPENSO", "APROBADO", "NOTABLE" o "SOBRESALIENTE". El prototipo será:
`void mostrarEvaluacion(float nota);`

c) La función *main* solicitará al usuario que indique cuántas notas de práctica y teoría va a introducir en el sistema, y a continuación comenzará a solicitar todos los pares teoría-práctica. Cada vez que el usuario introduzca una pareja de estas notas, el programa mostrará por pantalla el valor de su nota final correspondiente y el texto que indica si se trata de "APROBADO", "SUFICIENTE", etc.

Ejercicio 10: Implementar una función que calcule la distancia euclídea entre dos puntos con coordenadas X e Y. Recuerde que la distancia entre dos puntos es: $\sqrt{(x2-x1)^2+(y2-y1)^2}$

Ejercicio 11: Partiendo del Ejercicio de "la calculadora" de la práctica 2, se pide realizar las siguientes modificaciones:

a) Implementar una función para cada una de las operaciones que puede realizar la calculadora. Sustituir los cálculos en la función *main* por las llamadas a las funciones que son las nuevas encargadas de realizar los cálculos.
b) Implementar una función que se encargue únicamente de mostrar el menú de la calculadora.
c) Añadir a la calculadora las operaciones: Raíz Cuadrada y la potencia, haciendo todos los cambios que estime oportunos.

Ejercicio 12: Implemente una función que muestre por pantalla todos los divisores de un número entero positivo "n". La función *main* se encargará de solicitar al usuario el valor de "n" e invocar a la función encargada de mostrar sus divisores.

Ejercicio 13: Implemente una función que muestre por pantalla todos los divisores que tienen en común dos valores enteros (parámetros de entrada de la función). La función *main* pedirá dos números enteros por teclado y llamará a la función para mostrar sus divisores comunes.

Ejercicio 14: Implemente una función que calcule y devuelva el máximo común divisor de dos valores enteros (parámetros de entrada de la función). La función *main* pedirá dos números enteros por teclado y llamará a la función.

Ejercicio 15: Implemente una función que transforme una letra minúscula en su correspondiente mayúscula. La función tendrá un parámetro de tipo *char*. Si efectivamente el carácter pasado por parámetro es una letra minúscula, la función devolverá la mayúscula correspondiente, en caso contrario, devolverá el mismo carácter que se pasó como parámetro.