Funciones en C

versión 1.0 25-7-2022

1. En la carpeta `funciones-1` hemos modificado el programa `hola.c` y lo hemos llamado `hola_con_funcion_interna.c`. Crea el ejecutable mediante `gcc hola_con_funcion_interna.c -o hola_con_funcion_interna`. El programa hace lo mismo que antes pero hemos centralizado el saludo en la función `saludo`. Podemos tener muchas llamadas a dicha función en diferentes partes del programa (de la función principal `main`) y si queremos modificar dicho saludo solo tenemos que modificar dicha función. El concepto es semejante a usar estilos en LibreOffice o en html.
2. Analiza la diferencia introducida en `hola_con_funcion_interna_bis.c`
3. Ahora hemos puesto en archivos diferentes `main` y `saludo`. Primero debemos crear los códigos objeto: `gcc -c hola_con_funcion_externa.c` y `gcc -c funcion_hola.c`, que nos creará los archivos con extensión `.o` y después enlazamos para generar el ejecutable: `gcc hola_con_funcion_externa.o funcion_hola.o -o hola_con_funcion_externa`, aunque también podemos hacerlo todo en un solo paso: `gcc hola_con_funcion_externa.c funcion_hola.c -o hola_con_funcion_externa`, e incluso mezclando código fuente y objeto, por ejemplo si solo se ha modificado `funcion_hola.c` escribiremos `gcc hola_con_funcion_externa.o funcion_hola.c -o hola_con_funcion_externa`
4. Cuando trabajamos con un programa formado por varios archivos es muy útil usar el programa `make`, que se encarga de compilar solo los archivos que se han modificado, además de poder configurarlo con reglas para automatizar y simplificar tareas, como por ejemplo borrar el código objeto, los ejecutables, copiar los ejecutables en los directorios adecuados (lo que habitualmente llamamos instalar), etc. El archivo `Makefile` de ejemplo creará el ejecutable `hola_con_funcion_externa`, solo tienes que ejecutar la orden: `make`
5. La función `saludo` no recibe ningún argumento por eso es mejor declararlo explícitamente y eso se hace así: `int saludo (void)`. Si tampoco queremos que devuelva ningún valor la definimos así: `void saludo(void)`. Mira el ejemplo explicativo `hola_con_funcion_interna_void.c`
6. En la carpeta `funciones-2` tenemos el programa `suma.c` modificado. En el archivo `suma.c` tenemos `main` y solo se encarga de solicitar los números y mostrar el resultado y en el archivo `f_suma.c` tenemos la función `f_suma` que se encarga de realizar la suma. Analiza y prueba el código.
7. En la carpeta `funciones-3` hacemos lo mismo que en `funciones-2` pero usando variables externas o globales. Lee las explicaciones (empieza por `f_suma_globales.c`), compila y ejecuta `suma_globales` para comprobar que todo funciona correctamente. Unifica en un solo archivo llamado

`suma_globales_interno.c` las funciones `main` y `f_suma` y comprueba que ahora no hace falta la declaración de las variables, con su definición es suficiente, y que debes definirlas antes de usarlas en cualquier función.

8. En `funciones-4` se encuentra el fichero `suma_glob.c`, que es el mismo programa que se encuentra en `funciones-2` pero todo incluido en un único archivo para trabajar más fácilmente. Tienes que modificarlo para que el código que intercambia los valores de las variables, en el caso de que el primer número sea el mayor, vaya en una función. Para que pueda intercambiar los valores defínelas como globales (aunque es una solución muy ineficiente).

   En el fichero `suma_referencia.c` ya está hecho lo propuesto en el párrafo anterior pero para intercambiar las variables se pasan los valores por referencia en lugar de por valor, para que la función pueda acceder a los valores originales (esta es la manera adecuada de realizar esta tarea). Expliquemos primero los punteros con un ejemplo:

```
int m;  /* m es una variable entera, se reserva un espacio en la
           memoria RAM para ella, por ejemplo en la dirección
           120 de la memoria RAM */

int *p; /* p es un puntero que apuntará a una variable entera */

m=2;    /* m ahora vale 2 */

p=&m;   /* p apunta ahora a la dirección de memoria de m, por lo
           que p vale 120  */

*p=3    /* modificamos el valor al que apunta p, no el valor de
p. Ahora m valdrá 3.  */
```

   y después solo tienes que analizar, compilar y ejecutar `suma_referencia.c`

9. Modifica `funciones-5/vectores_mono.c` para que imprima por la pantalla los enteros desde el 100 al 91.

10. A partir de `vectores_mono.c` crea un programa llamado `suma_componentes.c` que sume todos los componentes del vector `v` y muestra el resultado por la pantalla.

11. Analiza y ejecuta `funciones-5/caracteres.c`.

12. Analiza y ejecuta `funciones-5/cadenas_1.c`.

13. Analiza y ejecuta `funciones-5/cadenas_2.c`.

14. Analiza y ejecuta `funciones-5/cadenas_3.c`.

15. Escribe un programa que solicite texto y que lo muestre en pantalla deletreado con un carácter por línea. Llámalo `salto.c`

16. Escribe un programa que solicite texto y que lo muestre en pantalla deletreado desde el final hasta el principio. Llámalo `inverso.c`

17. Añade dos bucles a `funciones-6/vectores_bi-1.c` para que imprima todos los valores del vector bidimensional.

18. Analiza y ejecuta `funciones-6/vectores_bi-2.c`.
19. Analiza y ejecuta `funciones-6/argumentos.c`.
20. Analiza y ejecuta `funciones-6/doble.c`.
21. A partir de `doble.c` escribe un programa que multiplique dos números introducidos como argumento y que muestre por pantalla el resultado. Llámalo `multiplica-1.c`
22. Modifica `multiplica-1.c` para que multiplique los números en una función aparte. Llámalo `multiplica-2.c`
23. Modifica `multiplica-2.c` para que multiplique los números mediante la definición de la multiplicación. Llámalo `multiplica-3.c`