Programación I

Práctica #5 **Curso 2024-2025** **1º curso**

Funciones

Introducción

A medida que realicemos programas cada vez más complejos, la técnica utilizada hasta ahora de implementar toda la funcionalidad del programa en la función `main()` deja de ser adecuada, ya que acabaría dando lugar a una secuencia de instrucciones inmanejable y muy difícil de mantener.

En esta práctica, aprenderemos a desarrollar programas modulares, es decir, cómo dividir nuestro programa en módulos o subprogramas que hagan nuestro código más legible y manejable. En C, estas unidades en que dividimos nuestro código se denominan **funciones**.

La definición de funciones nos permitirá además reutilizar código, de tal manera que no tengamos que escribir varias veces el mismo código para realizar las mismas operaciones.

Por otro lado, en esta práctica, aprenderemos a diferenciar entre la **declaración de una función** y la **definición de una función**.

Finalmente, también aprenderemos a usar la forma básica para pasar parámetros a una función: por **valor**.

Objetivos

Esta práctica tiene como objetivos fundamentales:

- Aprender a definir y utilizar funciones como mecanismo básico de programación modular en C.
- Aprender a declarar las funciones de forma separada de su definición.
- Aprender a personalizar el comportamiento de las funciones mediante el paso de parámetros.

Planificación

Sesión 5 de laboratorio (3 horas)

1. Realización de los ejercicios 1 a 3.
2. Realización de los ejercicios 4 a 6.
3. Realización del ejercicio 7.

Trabajo fuera del aula:

- Completar los códigos no finalizados en clase.

Funciones

El concepto de función

Las funciones son el elemento básico de programación modular en C. De hecho, en C todas las instrucciones de un programa están codificadas en funciones.

Ya has escrito varias funciones, ya que, en cada uno de los programas que has realizado hasta ahora, has definido una. Cada uno de tus programas se organizó como una única función denominada `main()`. Además, en tu código has utilizado otras funciones no desarrolladas por ti, como printf(), scanf(), …

Una función en C no es más que un conjunto de líneas de código que realizan una tarea específica y puede devolver un valor. Cuando desde algún punto de nuestro programa invocamos a una función, pasamos el control de la ejecución del programa a la misma. Una vez que la función finaliza su tarea, el control es devuelto al punto desde el cual la función fue llamada.

Veamos, con un ejemplo, su utilización:

```
// Programa Hola mundo con funciones
#include <stdio.h>

void mark () {       // Función mark()
   printf ("Hola Mundo\n");
   return;            // Sale de la función
}

int main () {         // Función principal
   mark ();          // llamada a la función mark()
   return 0;          // sale de main() con valor 0
}
```

Ejercicio 1:

Analiza el código anterior e intenta comprender el significado de cada una de sus líneas.
Escribe el programa (sin los comentarios), compílalo y ejecútalo.

En el ejemplo que acabas de codificar puedes ver que la instrucción `return` hace que finalice la ejecución de una función y el control retorne al punto desde el que se hizo la llamada.

La función `mark()` fue definida con valor de retorno de tipo `void` (es decir, sin valor de retorno). En ese caso, la instrucción `return` no lleva asociado ningún valor, ya que la función no debe devolver ningún valor a la función que la llama.

En cambio, si definimos una función que sí tenga valor de retorno (como, por ejemplo, la función `main()`, que tiene un valor de retorno de tipo `int`), `return` debe ir seguido de una expresión que genere un valor compatible con dicho tipo (`0` en el ejemplo).

Ejercicio 2:

Modifica el programa del ejercicio 1, añadiendo una función que se denomine `validate()`, que:

a) presente el siguiente mensaje de invitación:

```
Confirm exit? (YN):
```

b) lea de teclado un carácter y:
- si se pulsó la tecla `'Y'` o `'y'`, retorne un entero con valor 1,
- si se pulsó la tecla `'N'` o `'n'`, retorne un entero con valor 0, y,
- si se pulsó cualquier otra tecla, no retorne; en lugar de ello, vuelva a mostrar el mensaje de invitación, hasta que el usuario introduzca una entrada correcta.

Para escribir esta función puedes tomar como base parte del código del ejercicio 7 de la Práctica #3.

Modifica la función `main()` del programa para que, de forma repetitiva:
a) imprima el mensaje "`Hola Mundo`" (usando `mark()`), y, luego,
b) pregunte si se desea salir hasta que se conteste `'y'` o `'Y'` (usando `validate()`).

Como puedes ver, la primera utilidad de las funciones es permitirnos organizar nuestro código en diferentes módulos (en el último ejercicio, en tres funciones: `mark()`, `validate()` y `main()`).

Otra utilidad de las funciones es evitar repeticiones de código en nuestros programas cuando queremos realizar el mismo conjunto de operaciones en diferentes puntos de nuestro programa.

Ejercicio 3:

Modifica el programa del ejercicio 2, añadiendo una nueva función:
- `boundary()`, que imprima en pantalla 1 línea de 50 signos `'-'`.

Modifica la función `mark()`, para que, en vez de imprimir "`Hola Mundo`", imprima:

```
LIBRARY
```

Esta función no debe devolver ningún valor (tienes el código necesario para desarrollar `boundary()` en tu solución para el ejercicio 8 de la Práctica #3).

Modifica la función `main()` del programa para que, en vez de imprimir:

```
Hola mundo
```

imprima, llamando a `boundary()` y a la versión modificada de la función `mark()`:

```
--------------------------------------------------
LIBRARY
--------------------------------------------------
```

Paso de parámetros

*Las funciones son un conjunto de instrucciones que realizan una tarea específica. En general toman ciertos valores de entrada, llamados **argumentos** o **parámetros** y devuelven un valor de salida o valor de retorno, aunque, en C, tanto unos como el otro son opcionales, y pueden no existir.*

Los parámetros se definen en la **cabecera** de la función, entre paréntesis después del identificador de la función. Para cada uno de ellos se debe indicar su tipo y su identificador. Así, el formato general de la cabecera de una función será el siguiente:

```
tipo_retorno id_funcion (tipo_par1 id_par1, tipo_par2 id_par2)
```

Cuando se realiza la llamada a la función, a cada uno de los parámetros se le asigna el valor indicado para él en dicha llamada. Los parámetros, con los valores de entrada asignados, los podemos utilizar dentro del código de la función como cualquier otra variable del tipo indicado.

Ejercicio 4:

Modifica la función `boundary()` para que tome como parámetro el carácter que hay que imprimir, de tal manera que imprima las filas con el carácter indicado, en vez de imprimir siempre un signo `'-'`.

Prueba el programa para diferentes valores del parámetro:

Por ejemplo, si, en el `main()`, invocamos a la función mediante:

```
boundary ('-');
```

debería mostrar:

```
------------------------------------------------
```

Pero, si la invocamos mediante:

```
boundary ('*');
```

debería mostrar:

```
************************************************
```

Declaración de funciones

Ejercicio 5:

Modifica el programa realizado en el ejercicio 4 para que las funciones `validate()`, `boundary()` y `mark()` aparezcan al final del fichero de tu código fuente (después de la función `main()`).
Compila de nuevo el programa y observa lo que ocurre. ¿Cuál es el motivo?

En el lenguaje C, cualquier elemento debe ser declarado antes de poder ser utilizado. Seguramente ya hayas experimentado esta situación si, por error, has intentado utilizar una variable antes de declararla. Lo mismo ocurre con las funciones: en este caso has intentado llamar a tus funciones antes de declararlas.

Si una función no se declara explícitamente, C toma su definición también como declaración. En los ejercicios 1 a 4, definíamos al principio las funciones, con lo que, implícitamente, quedaban declaradas. Cuando, más abajo en el código, las llamábamos, no había ningún problema.

En este ejercicio 5, las definimos después del `main()`, con lo que, en realidad, las estamos declarando implícitamente después. Por eso, cuando las llamamos dentro del `main()`, aún no están declaradas.

Si queremos declarar explícitamente una función, porque, por ejemplo, queremos realizar en nuestro código una llamada a la misma antes de definirla, lo único que tenemos que hacer es escribir antes la declaración, que consiste en la cabecera de la función, pero terminada en ";", en vez de ir seguida del código de la función entre llaves. Es decir[1]:

```
…
// Declaración de la función, al principio del código
tipo_retorno  id_funcion (tipo_par1, tipo_par2, …);
…
// Llamada a la función, dentro de alguna otra función, por ejemplo, dentro del main()
id_funcion (par1, par2, …);
…
// Definición de la función, después del main()
tipo_retorno id_funcion (tipo_par1 id_par1, tipo_par2 id_par2) {
      …
      // Instrucciones de la función
      …
}
```

La declaración de las funciones suele situarse explícitamente al principio del código.

Ejercicio 6:

Modifica el programa del ejercicio anterior, añadiendo una declaración explícita de tus 3 funciones:
`validate()`, `boundary()` y `mark()`
al principio de tu fichero fuente (por ejemplo, después de las líneas `#include` que tengas)
Prueba de nuevo a compilar el programa.

[1] Al contrario que en la definición, en la declaración de una función no es necesario indicar el identificador de los parámetros. Con indicar el tipo es suficiente, aunque, si se pone también el identificador, no es erróneo.

Ejercicio 7:

Copia aquí el programa que escribiste en el ejercicio 3 de la Práctica #4 (que presentaba el menú de opciones de una aplicación), y realiza sobre él las siguientes modificaciones:

1) Añade la función `boundary()` que desarrollaste en el ejercicio 4.
2) Añade la función `mark()` que has desarrollado en el ejercicio 3, pero modificada para que reciba como parámetro la cadena de caracteres que queremos imprimir . Es decir, dependiendo de la llamada a la función:

`mark ("LIBRARY");` debe imprimir: `LIBRARY`

pero:

`mark ("PROGRAMACION I");` debe imprimir: `PROGRAMACION I`

3) Añade la función validate() que desarrollaste en el ejercicio 2, pero modificada para que reciba como parámetro la cadena de caracteres que queremos usar como invitación a teclear. Es decir, dependiendo de la llamada a la función:

`validate ("Confirm exit");` debe imprimir: `Confirm exit? (YN):`

pero:

`validate ("Esta usted seguro");` debe imprimir: `Esta usted seguro? (YN):`

4) Modifica la función `main()` para que:

a) En primer lugar, llame a `boundary()` y a `mark()` para mostrar una carátula como la del ejercicio 3.

b) En el bucle en el que se muestra el menú y se aceptan entradas del usuario:

- Cuando el usuario seleccione la opción `'X'`, se siga asegurando de que el usuario realmente quiere salir, pero lo haga llamando a la función `validate()`.
- Cuando el usuario seleccione la opción `'N'`, en vez de decir: `"New"` , llame a una nueva función `p_new()`, que, de momento, se limita a sacar por pantalla: `"New"`. Esta función retorna un valor entero (de momento, siempre un 0).
- Cuando el usuario seleccione la opción `'A'`, en vez de decir: `"Add"`, llame a una nueva función `p_add()`, que, de momento, se limita a sacar por pantalla: `"Add"`. Esta función retorna un valor entero (de momento, siempre un 0).
- Cuando el usuario seleccione la opción `'V'`, en vez de decir: `"Vote"`, llame a una nueva función `p_vote()`, que, de momento, se limita a sacar por pantalla: `"Vote"`. Esta función retorna un valor entero (de momento, siempre un 0).
- Cuando el usuario seleccione la opción `'C'`, en vez de decir: `"Clean"`, llame a una nueva función `p_clean()`, que, de momento, se limita a sacar por pantalla: `"Clean"`. Esta función retorna un valor entero (de momento, siempre un 0).

Resumen

Los principales resultados esperados de esta práctica son:

- Saber cómo declarar, definir y utilizar funciones.
- Saber cómo personalizar el comportamiento de las funciones mediante el paso de parámetros.

Como posibles líneas de trabajo adicional del alumno, se proponen las siguientes:

- Terminar los ejercicios no completados en clase.