Universidad del País Vasco
Euskal Herriko Unibertsitatea

Fundamentos de Programación:

Prácticas de C

1º Ingeniería Técnica de Telecomunicación

Bilbao, septiembre de 2009

Fundamentos de Programación: Prácticas de C.

Resumen

Con las siguientes prácticas se pretende afianzar los conceptos de programación estructurada que se van viendo en las clases de teoría de C.

Tras la realización de las prácticas se habrá implementado una aplicación en C de unas 400 líneas de código que permitirá jugar desde la consola al juego "Hundir la flota". En el juego se llevará un seguimiento de puntuaciones de distintos jugadores, se permitirá guardar y recuperar partidas, y además también se permitirá que un usuario externo pueda diseñar sus propias pantallas.

Para ello se partirá de un programa sencillo y se irá incluyendo en cada práctica un concepto nuevo de los vistos en teoría, comprobando la mejora que aporta sobre la versión anterior del programa.

Al principio de cada práctica se debe realizar una copia del directorio con todos los ficheros de la práctica anterior y, sobre esos ficheros, se incluirán las nuevas funcionalidades.

Índice general

Práctica 0

Familiarización con el Entorno

En esta primera clase se trata de que el alumno se familiarice con el entorno de programación Eclipse. Para ello se creará un workspace y se tratarán los conceptos de edición, compilación y depuración.

0.1. Primer programa

Para empezar a trabajar con Eclipse, lo primero será arrancar la aplicación haciendo doble click en su acceso directo. Enseguida se nos abrirá una ventana con información sobre Eclipse: la versión (3.2 en nuestro caso), el copyright, etc. A partir de ese momento, empezará nuestra interacción con Eclipse:

1. Cuando se nos abra la ventana con título `Workspace Launcher`, Eclipse nos está dando la opción de elegir lo que llama el `workspace`. Se trata de un directorio, nuestro directorio de trabajo. En nuestro caso, cada práctica la meteremos en un `workspace` diferente. Para ello, la ruta del `workspace` será de la siguiente forma:

   `/home/user1/workspace/<nombre del alumno>/<n° de práctica>`

   Por ejemplo, si el alumno Iker quiere trabajar en la práctica 0, tendrá que escribir el `workspace` siguiente:

   `/home/user1/workspace/Iker/0`

   En caso de querer entrar a un `workspace` que ya existe (el de una práctica antigua, por ejemplo), bastará con desplegar el menú haciendo click en la flechita o buscándolo mediante el botón `Browse`.

2. La primera vez que se entra en un `workspace`, Eclipse nos muestra una página de bienvenida `Welcome`. Para las prácticas no la vamos a utilizar, así que hay que cerrarla para poder empezar a trabajar, haciendo click en el aspa.

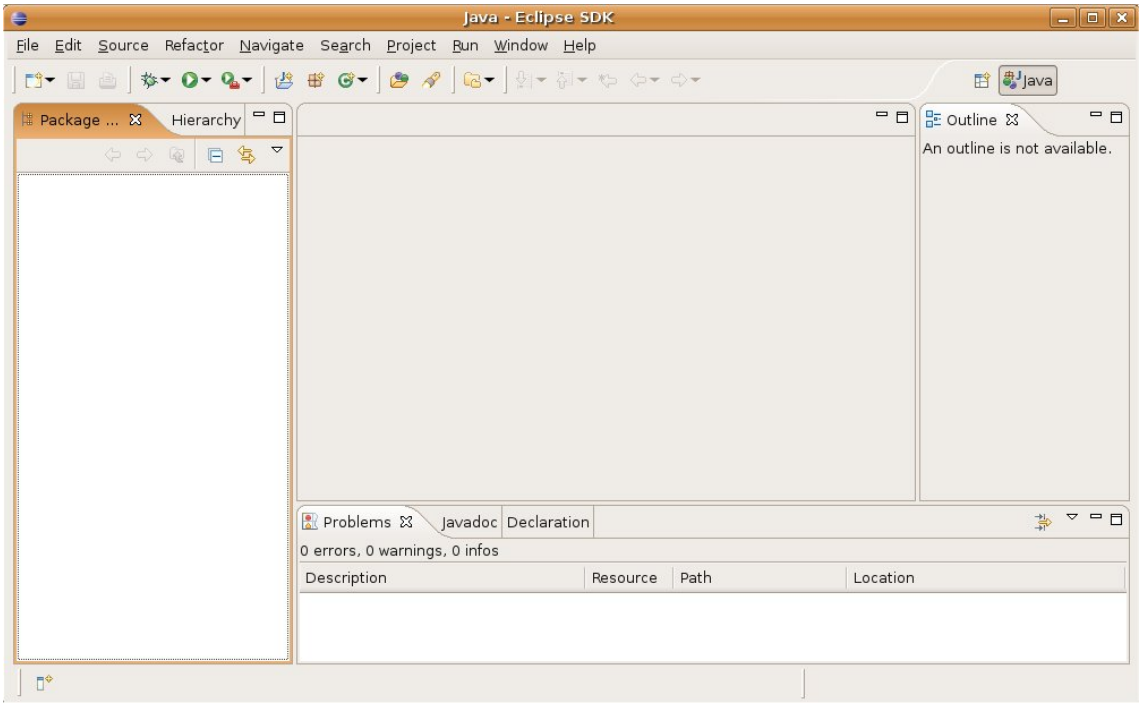

Figura 1: Entorno de Desarrollo Integrado (IDE) Eclipse

3. Una vez cerrada la ventana de bienvenida, estaremos ante lo que llamaremos la "perspectiva de desarrollo" de Eclipse, es decir, una ventana con diferentes zonas de información, como la que vemos en la figura 1.

4. Para empezar a escribir un programa en C, entraremos en el menú:

   `File->New->Project`

5. Se nos abrirá la ventana `New Project`, en la que elegiremos:

   `C-> Managed Make C Project`

6. A continuación se nos pide un nombre para el proyecto. En nuestro caso, si el `workspace` se corresponde con la práctica, el proyecto se corresponderá con un ejercicio. Así, por cada ejercicio habrá que hacer un proyecto diferente. Como nombre, le pondremos el número del ejercicio. Pulsamos `Next` para seguir.

7. En la siguiente ventana simplemente pulsaremos `Finish` sin cambiar nada.

8. Eclipse puede abrirnos una ventana llamada `Open Associated Perspective?`, a la que contestaremos que sí después de marcar la opción `Remember my decission`. Con eso evitaremos que nos lo vuelva a preguntar en otros proyectos del mismo workspace (figura 2).

9. Al crearse el proyecto, en la columna izquierda aparecerá el icono de una carpeta abierta con el nombre del proyecto. ¡Significa que se ha creado!.

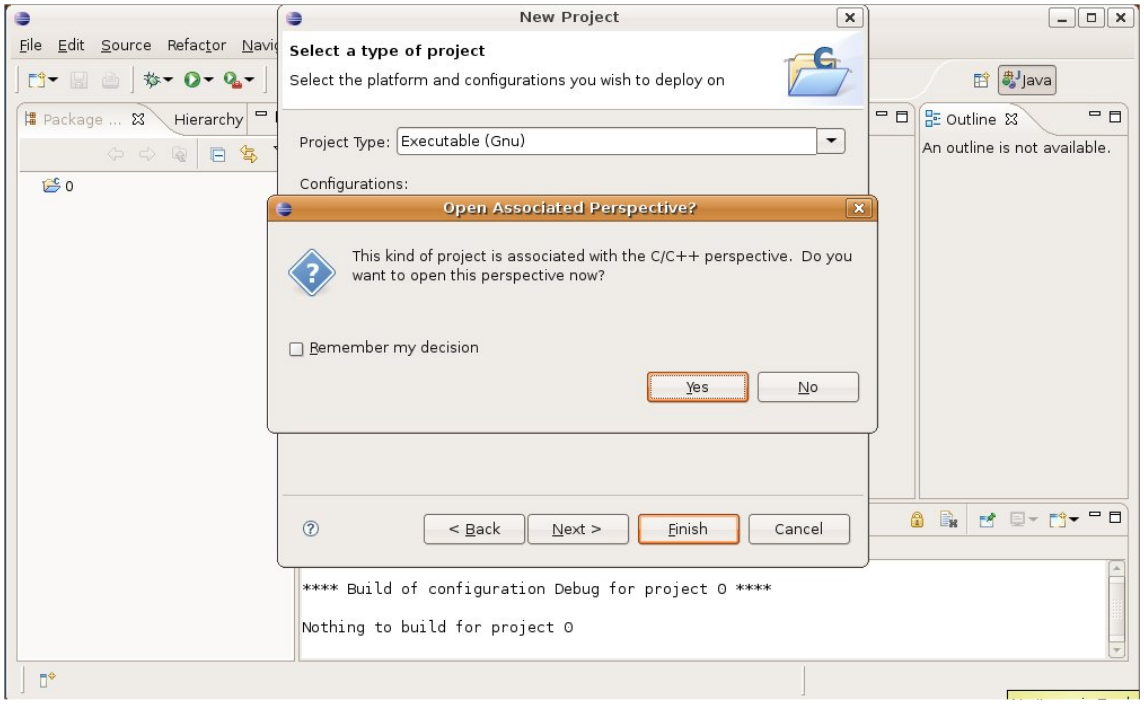

Figura 2: Abrir la perspectiva adecuada

Para empezar a escribir un fichero fuente, haremos click con el botón derecho del ratón sobre el proyecto, y en el menú desplegable que nos sale, elegiremos:

`New->Source File`

10. Se nos abrirá una ventana en la que se nos pide que indiquemos el nombre del fichero fuente. Pondremos el mismo nombre que el del proyecto, terminándolo en `.c`. Después, pulsaremos `Finish`.

11. En la columna izquierda aparecerán dos carpetas nuevas dentro de la del proyecto y el icono que indica que el fichero fuente está creado; pero está vacío. La zona central ahora representa una hoja en blanco, en la que escribiremos nuestro programa.

    Empezaremos con un programa sencillo ...

```
#include <stdio_ext.h>

int main()
{
    char nombre[20];
    int edad;

    printf("Introduzca su nombre: ");
    fflush(stdout);
```

```
    __fpurge(stdin);
    scanf(“%s”,nombre);

    printf(“Introduzca su edad: ”);
    fflush(stdout);
    __fpurge(stdin);
    scanf(“%d”,&edad);

    printf(“Hola %s, tienes %d años!!!!\n”,nombre,edad);

    return 0;
}
```

12. Guardaremos el fichero haciendo click en el icono del disquete; Eclipse, por defecto, compila los ficheros fuentes que tenga abiertos cada vez que guardemos alguno. Por este motivo, recomendamos no tener más de un proyecto abierto a la vez.

13. Una vez compilado, en la ventana `Problems` de la parte inferior de la zona del fichero fuente, veremos los problemas que hayan surgido durante la compilación y el linkado. Estos problemas se nos indicarán en dos formas: errores y `warnings` (avisos).

    Si hemos cometido errores sintácticos al escribir nuestro programa, el compilador lo indicará como errores, y no habrá creado ningún ejecutable, es decir, todavía no tendremos programa para ejecutar.

    Los `warnings` son avisos de problemas menos graves; si los problemas que ha tenido el compilador han producido “sólo” `warning`, se habrá creado un ejecutable, por lo que podremos ejecutar nuestro programa. De todas formas, es recomendable que no haya `warnings` porque con ellos el programa puede no funcionar correctamente.

0.2. Compilación y ejecución de un fichero fuente sin errores

1. Si hemos copiado el código anterior correctamente, lo único que nos puede aparecer en `Problems` es un `warning` indicando que el fichero no termina en una línea vacía ...

   ¡Así que tendremos programa para ejecutar!

2. Iremos al menú `Run->Run...`

   Se nos abrirá la ventana `Run`. Haremos doble click sobre la línea `C/C+` Local Application+ de la columna izquierda, y aparecerá una nueva línea debajo con el nombre del proyecto. En la zona de la derecha, aparecerá una ventana `Main` en la que tendremos que introducir el valor de `C/C+` Application+. Para ello

pulsaremos sobre el botón `Browse` y, en la ventana que se nos abre, elegiremos el binario `Binary` que se llame como el proyecto.

Finalmente, pulsaremos el botón `Run` para empezar a ejecutar nuestro programa.

3. La ejecución ocurrirá en la ventana `Console` que está al lado de la de `Problema`. Lo veremos con sólo pulsar su nombre ...

0.3. Compilación de un fichero fuente con errores sintácticos

1. Provocaremos un error sintáctico si, por ejemplo, comentamos la línea de declaración de la variable edad. En `Problems` aparecerá un error al guardar y compilar.

2. El error aparece con una breve explicación y el número de línea en la que se ha producido. Haciendo doble click sobre el error, nos llevará a la línea en la que ha detectado el error.

   Sin embargo, no siempre el lugar donde se ha detectado el error es donde se puede arreglar ...

0.4. Compilación, ejecución y depuración de un fichero fuente con errores de edición

1. Provocaremos un error de redacción que no produce un error sintáctico ... pero impide que el programa funcione bien. Para ello, cambiaremos el último `printf` por este otro:

   ```
   printf("Hola %d, tienes %d años!!!!\n",nombre,edad);
   ```

2. Guardaremos y compilaremos. Esta vez no habrá errores en `Problems`. Ejecutaremos el programa como antes ... y todo funcionará bien hasta llegar al final. Ahora, en el saludo no sale el nombre introducido, sino un número ... pero nosotros le hemos introducido un nombre ...

3. Estamos ante un error de funcionamiento, que se corrige depurando. En este caso, el nombre puede haberse guardado mal, o el problema es que se visualiza mal. Para depurar, entraremos esta vez en `Run->Debug`. Se nos abrirá la ventana `Debug`, muy parecida a la de `Run`, que utilizaremos para depurar.

   En `Debug` haremos lo mismo que hacemos en `Run` para ejecutar. Después, para comenzar la depuración, pulsaremos el botón `Debug`.

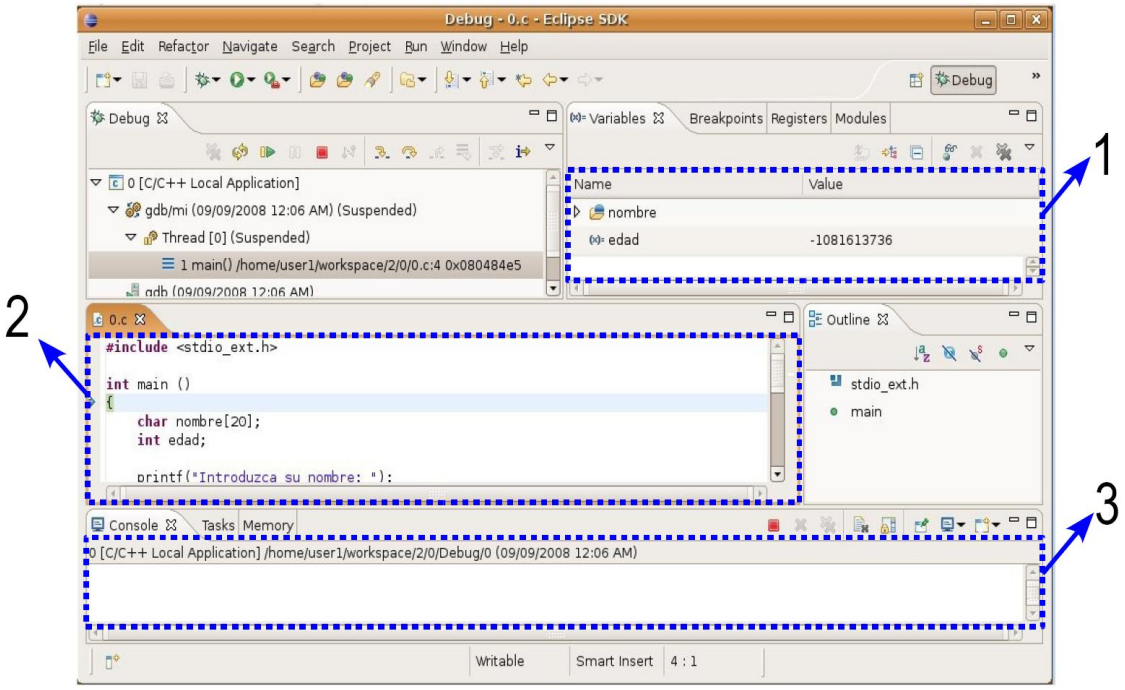

Figura 3: Perspectiva de depuración

4. La primera vez que se depura en un `workspace`, Eclipse pregunta si estamos de acuerdo en cambiar la perspectiva . . . como en el punto anterior. La perspectiva de depuración tiene diferentes zonas, como se ve en la figura 3.

   1: Variables en memoria
   2: Fichero Fuente
   3: Consola

5. El cursor en el fichero fuente estará sobre la primera llave de bloque `{`. Pulsando F6, el programa se ejecutará paso a paso. Para depurar, es importante comprobar lo que hay en la consola y los valores de las variables, cada vez que ejecutemos un paso (pulsemos F6). Así, podemos comprobar cómo no sale nada en la consola hasta ejecutar los `fflush`, o cómo `scanf` espera hasta que introduzcamos algo por teclado. En nuestro caso, veremos que los valores guardados en las variables son las que se han introducido por teclado (pulsando sobre las variables en la zona 1).

6. Al llegar al último `printf` comprobaremos que la variable nombre tiene el nombre adecuado, así que es el `printf` el que no hace lo que queremos . . . efectivamente, el cambiar la `s` por una `d` ha provocado este mal funcionamiento, que hemos podido localizar gracias al depurador.

Práctica 1

Asignaciones y Operaciones Básicas

En esta primera práctica se pretende familiarizar al alumno con la creación y el manejo de variables simples. Además se usarán algunas funciones básicas de entrada por teclado y salida por pantalla que nos resultan de utilidad y en las que se profundizará más adelante en el tema de funciones.

1.1. Declaración de variables

Vamos a comenzar definiendo el tamaño de nuestro tablero de *hundir la flota* dentro del cual estarán los distintos barcos. La siguiente figura muestra un ejemplo de cómo será este tablero:

	0	1	2	3	4	5	6	7	8	9
A										
B										
C										
D										
E										
F										
G										
H										
I										
J										

En la función main crea las dos variables siguientes:

- Una de tipo `int` que contenga el número de la última columna del tablero. Si el tablero tiene 20 columnas y numeramos desde el 0 (0, 1, 2, ..., 19) la última será la 19.

- Otra de tipo `char` que contenga la letra de la última fila del tablero. Si el tablero tiene 10 filas (A, B, C, ..., J) la última será la J.

Desde el mismo código fuente debes asignar ya un valor inicial a estas variables. Esto lo puedes hacer en la misma declaración de la variable o mediante una nueva instrucción de asignación.

Prueba a compilar el programa y que todo funcione correctamente (sin *warnings* ni errores).

1.2. Salida por pantalla

Un vez asignado un valor a estas dos variables que nos indican el tamaño del tablero, mostrar por pantalla un mensaje al usuario similar al siguiente:

```
¡Bienvenido a "Hundir la Flota"!

- El tamaño del tablero es: 20 columnas x 10 filas.
- La columna se indica con un número (0, 1, 2, ..., 19)
- La fila se indica con una letra mayúscula (A, B, C, ..., J)
```

Los números y letras que aparecen en el mensaje se deben calcular en función de los valores contenidos en la variables. Probar a cambiar los valores de la variables y volver a recompilar y ejecutar el programa para ver cómo se actualiza el mensaje.

1.3. Entrada por teclado

A continuación vamos a permitir que el usuario introduzca una coordenada por teclado y vamos a visualizar su valor.

Para ello creamos dos nuevas variables que almacenen la coordenada:

- Una de tipo `char` para almacenar la fila.
- Otra de tipo `int` para almacenar la columna.

A continuación mostrar un mensaje de texto pidiendo al usuario que introduzca la coordenada. Por ejemplo:

```
Introduce una coordenada (ej. B12) :
```

Leer el valor que introduce el usuario dejando guardando la letra en la variable de tipo `char` y el número en la de tipo `int`.

Finalmente mostrar al usuario la coordenada introducida con un mensaje similar al siguiente:

```
(Fila B, Columna 12)
```

Práctica 2

Sentencias Selectivas

Mediante el uso de sentencias selectivas podemos hacer que el programa realice comparaciones y en función del resultado ejecute unas instrucciones u otras. En esta práctica vamos a utilizarlas para comprobar que los datos introducidos por el usuario sean correctos.

2.1. Procesando la entrada del usuario

Partiendo del código de la práctica anterior, vamos a habilitar la posibilidad de que el usuario introduzca la letra tanto en mayúsculas como en minúsculas. Para ello:

- Nada más leer la entrada del usuario comprobar si la letra (correspondiente a la fila) introducida está entre los valores 'a' y 'z'. Si es así, restarle 32 al código ASCII para transformar la letra en una mayúscula.
- Ahora será este nuevo valor el que se utilice en el resto del código, tanto para comprobar si está en un rango de valores válido (siguiente apartado) como para visualizar la coordenada por pantalla.

2.2. Validando la entrada del usuario

Añadir las instrucciones necesarias para que, tras disponer de las coordenadas introducidas por el usuario ya procesadas, compruebe si son válidas o no. En caso de que no sean válidas, avisar de ello y salir. En concreto:

- Comprobar si la fila está entre 'A' y la letra correspondiente a la última fila. Si no es así, se visualizará un mensaje por pantalla indicando el rango válido de filas.

- Comprobar si la columna está entre 0 y el número correspondiente a la última columna. Si no es así, se visualizará un mensaje por pantalla indicando el rango válido de columnas.

2.3. Finalizando el juego

Finalmente realizamos una comprobación adicional que servirá al usuario para notificar que desea terminar de jugar (esto será de utilidad más adelante).

- Añadir las instrucciones necesarias para que en caso de que el usuario introduzca como coordenada "Z0" se muestre un mensaje indicando que el juego ha terminado y se finalice la aplicación.

Práctica 3

Sentencias Repetitivas

Mediante las sentencias repetitivas podemos hacer que un fragmento de código se repita un determinado número de veces a la vez que van cambiando sus variables. En esta práctica las vamos a utilizar en primer lugar para dibujar el tablero y en segundo lugar para permitir que el usuario introduzca múltiples coordenadas.

3.1. Vamos a dibujar el tablero

De nuevo partimos del código de la práctica anterior. Vamos a cambiar el mensaje de inicio donde se explica cómo indicar fila y columna, así como su rango válido, por un dibujo del tablero (y es que vale más una imagen que mil palabras).

Como primera prueba, dibujamos los ejes de coordenadas:

- Mediante un bucle `for` mostrar por pantalla el número de cada columna según el siguiente formato:

  ```
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
  ```

  Notar que al principio de la línea hay dos espacios en blanco y que entre cada número hay otro espacio en blanco. Además los números de un sólo dígito deben llevar un espacio en blanco adicional con el objetivo de ocupar lo mismo que los números de dos dígitos.

- Mediante otro bucle `for` fuera del anterior, mostrar en vertical la letra asociada a cada fila dejando una línea en blanco antes de cada letra:

  ```
  A

  B
  ```

```
    C
```

El resultado final debería ser algo similar a lo mostrado a continuación:

```
¡Bienvenido a "Hundir la Flota"!

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19

A

B

C

D

E

F

G

H

I

J

Introduce una coordenada (ej. B12) :
```

El tablero así ya podría ser funcional, pero mejor si dibujamos las celdas. Para ello vamos a modificar el segundo bucle `for` anidando otros bucles dentro de él.

- Substituir la línea en blanco por una línea que empiece por un espacio en blanco y que a continuación repita tantas veces los caracteres "`|__`" (sin las comillas) como columnas tenga el tablero. Ejemplo:

```
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
```

- En la línea que muestra la letra, hacer que a continuación repita el patrón "`|  `" (sin las comillas) tantas veces como columnas tenga el tablero.

De nuevo, el resultado final debería ser algo similar a lo mostrado a continuación:

```
¡Bienvenido a "Hundir la Flota"!

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
A|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
B|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
C|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
D|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
E|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
F|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
G|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
H|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
I|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
J|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Introduce una coordenada (ej. B12) :
```

Prueba a modificar el número de filas y columnas y ver cómo efectivamente cambia el número de iteraciones del bucle y, por consiguiente, el tamaño del tablero dibujado.

3.2. Múltiples entradas

Ahora que ya conocemos las sentencias repetitivas, vamos a utilizarlas para permitir que el usuario pueda introducir varias coordenadas en lugar de tan solo una.

- Crear una variable `salir` de tipo `char` que inicialmente valga 0.
- Meter en un bucle `do-while` la parte de código correspondiente a la interacción con el usuario. De esta forma se le irá pidiendo que meta coordenadas.
- El bucle terminará cuando el usuario introduzca la combinación Z0, momento en el que se mostrará el mensaje al usuario y se pondrá a 1 la variable `salir`.

A continuación se muestra un ejemplo de la salida que debería mostrar el programa con la nueva modificación:

```
¡Bienvenido a "Hundir la Flota"!

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
A|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
B|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
C|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
D|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
E|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
F|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
G|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
H|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
I|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
J|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Introduce una coordenada (ej. B12) : g6

(Fila G, Columna 6)

Introduce una coordenada (ej. B12) : l8

(Fila L, Columna 8)
El rango válido de filas es: A - J

Introduce una coordenada (ej. B12) : l99

(Fila L, Columna 99)
El rango válido de filas es: A - J
El rango válido de columnas es: 0 - 19

Introduce una coordenada (ej. B12) : z0

--- Aplicación finalizada a petición del usuario ---
```

Práctica 4

Funciones

¡Por fin podemos usar funciones!

La función principal ya va creciendo demasiado y el código complicándose. Es normal que un programa medianamente complejo ocupe varios miles de líneas y sería muy complicado abordarlo empleando una única función. Precisamente la programación estructurada consiste en enfrentarse a este problema aplicando el principio de *divide y vencerás*. Las funciones son el resultado de esta división del problema en partes más pequeñas.

4.1. Diseño

Antes de comenzar a escribir la primera línea de código de cualquier programa se debe hacer un trabajo previo (tanto o más importante que escribir el código) y que consiste en realizar un buen **diseño**. Se debe pensar cómo estructurar el programa, es decir, en qué partes más pequeñas dividirlo y cómo comunicar estas partes.

Ahora que ya sabemos cómo crear funciones propias, vamos a re-estructurar el código antes de que siga creciendo y sea demasiado tarde. Como regla general la función `main` no debería ocupar más de una pantalla y debería reflejar todo lo que hace el programa. Típicamente lo que contendrá será llamadas a otras funciones que serán las que hagan el trabajo *engorroso*.

Teniendo esto en cuenta, realiza un gráfico donde indiques en qué funciones vas a dividir el código actual (el que viene de la práctica anterior) y un diagrama de flujo de cómo se van a utilizar esas funciones desde el `main`.

En el resto de la práctica se propone un diseño que deberás contrastar con el que hayas hecho tú previamente y debatir en clase las ventajas de uno y de otro.

4.2. Re-estructurando el código

Una vez hecho el diseño, vamos a separar el código en las funciones que hemos considerado:

- Función `DibujaTablero`:
  - Toma dos parámetros de entrada, uno de tipo `char` con la letra de la última fila y otro de tipo `int` con el número de la última columna.
  - No devuelve ningún valor como salida.
  - Dibuja el tablero con las dimensiones indicadas en los parámetros.
- Función `LeeCoordenadas`:
  - Toma como parámetros de entrada la última fila y la última columna del tablero.
  - Devuelve dos parámetros de salida, uno de tipo `char` con la fila seleccionada y otro de tipo `int` con la columna seleccionada.
  - Además asociado a su identificador devuelve un valor de tipo `char` que valdrá 0 en caso de que el usuario haya indicado que desea terminar y 1 en caso contrario.
  - Esta función realiza la lectura de las coordenadas introducidas por teclado, comprobando si son correctas y solicitando al usuario que las vuelva a introducir en caso de que no sean correctas. Utilizar para ello el código ya implementado en la práctica anterior.
- Función `main`:
  - Rehacer la función `main` para que use las dos funciones anteriores. Deberá mostrar el tablero y pedir una entrada de datos hasta que el usuario indique que desea finalizar, cosa que sabrá gracias al valor de retorno de `LeeCoordenadas`.
  - Recordad que en esta función tiene que quedar claro de una forma muy legible y simplificada lo que hace el programa.

4.3. El primer barco

Hasta ahora el juego no tiene mucho sentido ya que no se hace nada con las coordenadas que introduce el usuario. Para solucionar esto vamos a crear un primer barco muy simple. En posteriores prácticas aumentaremos el número y tipo de barcos disponibles.

Nuestro primer barco va a ser de tan solo una casilla. Para ello vamos a añadir una nueva función y a ampliar la función `main`:

- Función `CompruebaTiro`:
  - Toma cuatro parámetros de entrada: las coordenadas introducidas por el usuario y las coordenadas con la posición del barco.
  - Devuelve un valor de tipo entero con la puntuación del tiro. La puntuación será 0 si no se ha dado al barco y, de momento, 1 si sí se le ha dado.
  - Esta función es excesivamente simple, de hecho se podría poner el código dentro del `main` y evitarse el crearla. Pero aquí es donde entra de nuevo el diseño, y es que esta función se va a ir ampliando en las prácticas posteriores, con lo que ya dejamos el camino comenzado.
- Función `main`:
  - Crear e inicializar las variables correspondientes para almacenar la posición del barco.
  - Crear una variable vidas de tipo entero e inicializarla con el número de tiros que queráis permitir al jugador.
  - Crear otra variable de tipo entero en el que se irán sumando los puntos que consiga el jugador y que inicialmente valdrá 0.
  - En el bucle principal, y tras leer las coordenadas del usuario, llamar a la función `CompruebaTiro`. Se deberá comprobar el código de retorno de esta función de forma que:
    - Si es igual a cero, se le reste una vida al usuario. Se debe indicar al usuario que ha fallado y mostrarle el número de vidas restante. Además si el usuario se queda sin vidas el programa debe finalizar.
    - Si es distinto de cero, su valor se suma a la variable que lleva los puntos. Se debe mostrar un mensaje al usuario indicando que ha acertado y el número de puntos acumulados.
  - Finalmente el programa antes de terminar debe indicar si se ha salido porque lo ha solicitado el usuario (Z0) o porque se le han acabado las vidas (*Game Over*). En este último caso además debe mostrarse la puntuación obtenida.

Práctica 5

Arrays Unidimensionales

En esta práctica vamos a introducir otro concepto muy importante, los arrays. Un array permite tener una colección de variables del mismo tipo. En concreto lo vamos a usar para permitir que nuestro juego tenga más de un barco.

5.1. Creando la flota

En este apartado vamos a permitir crear una flota de barcos, de momento, todos de una única casilla. Para ello vamos a crear una nueva función y actualizar otras ya existentes:

- Función `CreaFlota`
  - Toma como entrada dos arrays, uno de tipo `char` y otro de tipo `int`. En el primero guardará las filas en las que estén los barcos que queramos poner y en el segundo las columnas correspondientes. De esta forma, por cada barco tendremos su fila guardada en el primer array y su columna guardada en la misma posición del segundo array.
  - Devuelve como salida un valor de tipo `int` indicando el número de barcos que se han creado.
  - En esta función se debe inicializar estos arrays con las posiciones que elijamos para, por ejemplo, 3 barcos.
- Función `CompruebaTiro`
  - Ahora que tenemos varios barcos en lugar de tan solo uno, habrá que actualizar esta función.
  - Como parámetros de entrada, en lugar de la fila y columna del barco, ahora tomará dos arrays con las filas y columnas de todos los barcos de la flota. Además, relacionado con estos arrays, recibirá otro parámetro de tipo `int` que indicará el número de elementos válidos en ambos arrays.

  - Deberá recorrer esos arrays para comprobar si el tiro ha dado a algún barco:
    - En caso afirmativo, escribirá en las posiciones correspondientes al barco las coordenadas (Z,0). De esta forma evita que el usuario sume puntos disparando siempre a la misma posición. Finalmente devolverá un 1 como valor de retorno.
    - En caso negativo devolverá 0 como valor de retorno.

- Función `main`
  - Deberá crear dos arrays, uno de tipo `char` para las filas de los barcos y otro de tipo `int` para las columnas.
  - Deberá llamar a la función `CreaFlota` pasándole estos dos arrays para que los inicialice. Además, también recogerá el valor de retorno de la función para así saber cuántas de las posiciones del array tienen datos válidos.
  - Se deberá actualizar la llamada a la función `CompruebaTiro` para que le pase los arrays con las posiciones de los barcos y el número de elementos válidos en ambos arrays.

5.2. Mostrando los tiros

Bien, ya es hora de dar un poco de animación a ese tablero que siempre aparece vacío. Para ello vamos a llevar un seguimiento de los tiros realizados para dibujar posiciones aún sin usar, tiros acertados y tiros fallados. Para hacer esto, vamos a tener que crear nuevos arrays y modificar funciones existentes:

- Función `main`
  - Crear tres nuevos arrays para llevar un seguimiento de los tiros. Un array indicará la fila del tiro, otro la columna y otro el resultado devuelto por `CompruebaTiro`.
  - Además habrá que crear otra variable de tipo entero que lleve un seguimiento del número de tiros almacenados en estos tres arrays.

- Función `DibujaTablero`
  - Ahora esta función deberá mostrar las casillas vacías si aún no se han usado y el valor de resultado del tiro en caso de que sí se hayan usado.
  - Para ello habrá que pasarle como nuevos parámetros de entrada los tres nuevos arrays y el número de elementos válidos en ellos.

A continuación se muestra un ejemplo de cómo quedaría la salida por pantalla con el código como está ahora:

```
¡Bienvenido a "Hundir la Flota"!

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
A|  |  |  | 1|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
B|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
C|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
D|  |  |  | 0| 0|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
E|  |  |  |  |  |  |  |  | 0| 0| 1| 0|  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
F|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
G|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
H|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
I|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
J|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Fallaste!!, te quedan 0 vidas.

Puntuación: 2

--- Game Over ---
```

Práctica 6

Arrays Bidimensionales

El manejo del tablero nos ha venido pidiendo a gritos que usemos arrays bidimensionales, por fin es el momento. En esta práctica vamos a ver cómo se simplifica el código por el hecho de usar estos arrays.

6.1. Cambiando a arrays bidimensionales

Usaremos un array bidimensional para almacenar la posición y tipo de los barcos y otro array bidimensional para almacenar los tiros realizados junto con su resultado. Además ahora podremos utilizar ya barcos de distintos tamaños.

- El array de barcos en cada posición (fila, columna) contendrá un 0 si no hay ningún barco que pase por ella y un número con el tamaño del barco (número de casillas que ocupa) si sí hay un barco que pase por ella.
- El array de tiros en cada posición contendrá un `-1` si aún no se ha realizado un tiro en esa posición, un 0 si se ha disparado en esa posición y no había barco, y el tamaño del barco en caso de que sí hubiese barco.

La reserva de memoria de estos arrays la vamos a realizar de forma estática. Para ello vamos a definir dos macros con las dimensiones del tablero, una con el número de filas y otra con el número de columnas.

Este cambio en la forma de representar los datos va a implicar las siguientes adaptaciones al código que teníamos en la práctica anterior:

- Función `main`
  - Ya no hacen falta las variables que indican la dimensión del tablero, ya que ahora está indicada por las dos macros que hemos creado.
  - Las variables que indicaban número y posición de los barcos se pueden sustituir por un único array bidimensional. Este array será de tipo `int`

y tendrá las mismas dimensiones que el tablero. Cada posición del array indicará si hay algún barco y su tamaño en caso afirmativo.

- Las variables utilizadas para llevar el seguimiento de los tiros también se podrán sustituir por un único array bidimensional. Este array será de tipo `int` y tendrá las mismas dimensiones que el tablero. Cada posición del array indicará si se ha realizado un tiro y su resultado en caso afirmativo.
- Habrá que actualizar los parámetros que se les pasa a las distintas funciones a las que llama. Los cambios a estas funciones se detallan a continuación.

- Función `IniciaArrays`
  - Reemplaza a la anterior función `CreaFlota`
  - Toma como parámetros los dos arrays bidimensionales, el que indica la posición de los barcos y el que almacena los tiros realizados.
  - Ya no devuelve ningún valor como parámetro de retorno asociado a su identificador.
  - Inicializa los arrays con valores seguros. El de barcos todo con ceros y el de tiros todo con `-1`. Un 0 en el array de barcos indica que en esa posición no hay ningún barco. Un `-1` en el array de tiros indica que en esa posición aún no se ha realizado ningún tiro.
  - Finalmente almacena la posición de dos barcos en el array de barcos. Uno horizontal que ocupe 3 casillas y otro vertical que ocupe 4 casillas. En cada casilla ocupada por cada barco se debe sobreescribir el 0 que estaba almacenado por un número que indique el tamaño del barco correspondiente.

- Función `DibujaTablero`
  - Ahora no requiere que se le pasen las dimensiones del tablero ya que se pueden usar directamente las macros que indican el número de filas y de columnas.
  - Para indicarle los tiros bastará con pasarle el array bidimensional correspondiente.
  - A la hora de dibujar el tablero consultará por cada casilla la posición correspondiente en el array bidimensional de tiros.
    - Si su valor es `-1`, es que aún no se ha realizado ningún tiro en esa posición y deberá dibujarse la casilla en blanco.
    - En caso contrario habrá que dibujarse en la casilla el valor almacenado en el array. Este valor será 0 si se ha realizado un tiro y no había barco y distinto de 0 en caso contrario.

- Función `LeeCoordenadas`

  - Ya no requiere que se le indiquen las dimensiones del tablero para hacer la comprobación. Ahora estas dimensiones las puede obtener directamente a partir de las macros creadas.
  - Habrá que actualizar las comprobaciones ya que ahora no disponemos de una variable que indique la última fila o columna, sino el número total de filas y de columnas.
  - Como valor de fila, en lugar de devolver la letra, se va a devolver el número de la fila (manteniendo el tipo de dato como `char`). El valor de la columna se devuelve como siempre.

- Función `CompruebaTiro`
  - Se le pasan como parámetros la fila y columna del tiro, así como los dos arrays bidimensionales con la posición de los barcos y los tiros realizados.
  - En primer lugar debe consultar el array con la posición de los tiros para ver si el tiro ya se había realizado en esa misma posición. Para ello bastará con comprobar si el valor correspondiente en el array es distinto de `-1`, en cuyo caso la función debe devolver 0 y terminar.
  - Si la posición del array aún no se había usado (caso normal), almacenará en ella el valor que haya en la misma posición del array de barcos. Este valor será 0 si no hay ningún barco en esa posición y distinto de cero en caso de que haya algún barco.
  - Finalmente devolverá este valor como parámetro de retorno para que pueda ser consultado directamente por la función que la ha llamado (`main`) y saber si ha fallado o acertado para así actualizar las vidas o los puntos.

6.2. Flota hundida

Hasta ahora habíamos pasado por alto un pequeño detalle: ¿qué pasa cuando hemos hundido todos los barcos? ...Lo que pasa es que el juego permite seguir disparando hasta que se acaben las vidas. Vamos a solucionar esto comprobando si la partida se puede dar por ganada.

- Función `ComparaArrays`
  - Toma como parámetros de entrada los arrays bidimensionales con la posición de los barcos y los tiros realizados.
  - Devuelve como parámetro de salida un `char` que valdrá 0 si quedan barcos por hundir o 1 si se ha hundido toda la flota.
  - Para ello debe comprobar si en las posiciones de los barcos el valor correspondiente del array de tiros es distinto de `-1`.

- Función `main`

- Después de comprobar el tiro debe llamar a la función `ComparaArrays` para saber si la partida ha concluido.
- En caso de que la partida concluya por haber hundido toda la flota, debe mostrar un mensaje de enhorabuena.

A continuación se muestra un ejemplo de una posible salida por pantalla tras una partida:

```
¡Bienvenido a "Hundir la Flota"!

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
A|  | 0|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
B|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
C|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
D|  |  |  | 3| 3| 3|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
E|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
F|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0| 4| 0|  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
G|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4|  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
H|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4|  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
I|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4|  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
J|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Tocado!!, vas 7 puntos.

¡¡¡Enhorabuena, juego superado!!!
```

Práctica 7

Cadenas de Caracteres

En esta práctica vamos a usar un nuevo tipo de datos, las cadenas de caracteres. En C este tipo de datos se trata como un array unidimensional de caracteres acabado siempre en el carácter nulo. En la librería estándar se proporcionan distintas funciones básicas que nos simplifican el trabajo con cadenas de caracteres y con las que vamos a practicar.

7.1. Dando un nombre al usuario

Al empezar el juego vamos a pedirle al usuario su nombre. Este nombre lo utilizaremos en prácticas posteriores para realizar un seguimiento de las puntuaciones de distintos usuarios.

- Función `main`
  - Crea una cadena de caracteres de un tamaño máximo que definiremos en una macro.
  - Antes de empezar el juego pide el nombre al usuario mediante la función que se explica a continuación. Este nombre deberá quedar guardado en la variable que se ha creado.
  - Cuando se termine el juego por cualquiera de los tres casos (se acaban las vidas, se hunde toda la flota, o el usuario pide terminar la partida) se debe mostrar un mensaje indicando el nombre del usuario y el número de puntos conseguidos.
- Función `PideNombre`
  - Utiliza como parámetro de entrada/salida un puntero a una cadena de caracteres ya reservada en memoria y con el tamaño indicado por la macro.
  - Muestra un mensaje al usuario preguntándole por su nombre y lo lee por teclado guardándolo en la posición indicada por el puntero.

7.2. Procesando la entrada del usuario

Al ejecutar el programa del apartado anterior habréis observado que junto al nombre del usuario se guarda también el carácter de final de línea. Además, si un usuario no introduce ningún nombre el programa mostrará el mensaje con las puntuaciones con el campo de nombre en blanco. A continuación solucionamos estos y otros asuntos.

- Función `ProcesaCadena`
  - Utiliza como parámetro de entrada/salida un puntero a una cadena de caracteres acabada en el carácter nulo.
  - Recorre esta cadena eliminando los espacios en blanco, tabuladores y saltos de línea (`'\n'` y `'\r'`) y guarda el resultado en la misma dirección sobreescribiéndola.
- Función `PideNombre`
  - Debe llamar a la función `ProcesaCadena` tras leer la entrada del usuario.
  - Después de procesar la cadena, debe comprobar si ésta está vacía y, en caso afirmativo, almacenar en ella como nombre de usuario `Anónimo`.

Práctica 8

Estructuras

¡Por fin podemos usar estructuras! Las estructuras nos permiten agrupar una serie de datos de distintos tipos y que hacen referencia a un mismo elemento.

8.1. Posición

En primer lugar vamos a empezar practicando con una estructura sencilla que usaremos para referenciar un punto en el tablero.

- Estructura `Posicion`
  - Contiene un primer campo de tipo `char` que indica la fila en el tablero.
  - Como segundo campo contiene la columna en el tablero almacenada como dato de tipo `int`.
- Función `LeeCordenadas`
  - En lugar de los parámetros de fila y columna, ahora toma por referencia una estructura de tipo `Posicion` y almacena ahí las coordenadas leídas del teclado.
- Función `CompruebaTiro`
  - En lugar de la fila y columna del tiro, ahora toma como parámetro de entrada una estructura de tipo `Posicion` con la posición del tiro. Esta estructura se pasará por valor.
- Función `main`
  - Deberá ser actualizada para que utilice la nueva estructura y llame de forma correcta a las funciones que se han modificado.

8.2. Jugador

A continuación vamos a crear otra estructura que nos permita referenciar de forma conjunta los datos del jugador. Esta estructura nos resultará de utilidad más adelante cuando llevemos un seguimiento de puntuaciones de distintos usuarios o cuando queramos guardar y recuperar partidas.

- Estructura `Jugador`
  - Contiene un primer campo con el nombre del jugador. Está almacenado como una cadena de caracteres con tamaño máximo el indicado por la macro que usamos en la práctica anterior.
  - Como segundo campo contiene la puntuación del usuario almacenada como un dato de tipo `int`.
  - Finalmente tiene otro campo con el número de vidas del jugador.
- Función `IniciaJugador`
  - Reemplaza a la función `PideNombre` de la práctica anterior.
  - Toma como parámetro de entrada/salida un puntero a una estructura de tipo `Jugador`.
  - Pide el nombre del usuario por teclado y, tras procesarlo según se describió en la práctica anterior, lo devuelve en el campo correspondiente de la estructura.
  - Inicia a 0 el campo de la estructura que contiene la puntuación del usuario.
  - Inicia al máximo el número de vidas del jugador.
- Función `main`
  - Deberá ser actualizada para que utilice la nueva estructura en lugar de las variables separadas que se usaban para el nombre del jugador, su puntuación y las vidas restantes.

8.3. Barco

Vamos a complicarlo un poco más. Por un lado vamos a usar una estructura como campo de otra estructura, y por otro lado vamos a trabajar con un array de estructuras. Para ello vamos a crear una estructura que almacene la posición del barco y que usaremos para dibujar la flota al principio.

- Estructura `Barco`
  - Contiene la posición de un extremo del barco almacenada como un dato de tipo `Posicion`.

  - A continuación tiene un campo de tipo `char` que indica la dirección en la que está el barco. Este campo valdrá `'H'` si el barco está en horizontal, `'V'` si el barco está en vertical y `'D'` si el barco está en diagonal.
  - Finalmente tiene otro campo de tipo `int` que indica la longitud del barco.

- Función `DibujaBarco`
  - Toma dos parámetros de entrada: los datos del barco a dibujar como una estructura de tipo `Barco`, y el array bidimensional con las posiciones de los barcos.
  - Dibuja en las casillas correspondientes del array bidimensional el barco indicado como parámetro.

- Función `IniciaArrays`
  - Conserva el código que inicializa con valores seguros los dos arrays y elimina el código en el que se dibujaban unos barcos iniciales.
  - Crea un array de estructuras de tipo `Barco` e inicialízalo con las posiciones de 3 barcos: uno horizontal de longitud 3, otro vertical de longitud 5 y otro diagonal de longitud 4.
  - Recorre este array mediante un bucle llamando a la función `DibujaBarco` de forma adecuada para que dibuje los tres barcos.

Práctica 9

Ficheros binarios

Mediante el uso de ficheros podemos hacer que la información generada a lo largo de la ejecución del programa sea persistente, es decir, que no se pierda al finalizar el mismo. De esta forma podemos retomarla en una ejecución posterior del programa. Esto la vamos a utilizar en esta práctica para poder salvar partidas entre otras cosas.

9.1. Salvando partidas

Dicho y hecho, ahora que ya podemos manejar ficheros, vamos a guardar la partida en caso de que el usuario salga sin haber terminado el juego y vamos a reanudar la partida automáticamente en la próxima ejecución.

- Función `SalvaPartida`
  - Toma como parámetros de entrada un puntero a un estructura de tipo `Jugador` (usamos el puntero por eficiencia en el paso de parámetros), un puntero al array de barcos y otro puntero al array de tiros.
  - Devuelve como valor asociado a su identificador un `int` que valdrá 0 en caso de que haya habido algún error al guardar la partida y 1 en caso de que haya ido bien.
  - Abre un fichero binario y escribe los datos almacenados en los tres parámetros que se le han pasado a la función. Si hay algún error en este proceso, deberá mostrar un mensaje de error y devolver 0.
  - Cierra el fichero.
- Función `CargaPartida`
  - Toma como parámetros de entrada un puntero a un estructura de tipo `Jugador`, un puntero al array de barcos y otro puntero al array de tiros.

- Devuelve como valor asociado a su identificador un `int` que valdrá 0 en caso de que haya habido algún error al cargar la partida y 1 en caso de que haya ido bien.
- Abre un fichero binario y lee los datos almacenados en los tres parámetros que se le han pasado a la función. Si hay algún error en este proceso, deberá mostrar un mensaje de error y devolver 0.
- Cierra el fichero.

- Función `Terminar`
  - Como la gestión del fin del juego se está complicando, vamos a quitarlo de la función `main` y lo ponemos en esta función.
  - Toma como parámetros de entrada un puntero a un estructura de tipo `Jugador` (usamos el puntero por eficiencia en el paso de parámetros), un puntero al array de barcos y otro puntero al array de tiros.
  - Comprueba las tres posibilidades de finalización del juego (juego superado, *game over* y partida abandonada) y muestra el mensaje correspondiente en pantalla.
  - Ahora además, si el juego termina porque el usuario ha solicitado salir, se le preguntará al usuario si desea salvar la partida para retomarla la próxima vez que ejecute el programa. Si el usuario responde que sí, se llamará a la función `SalvaPartida`.
  - Esta función devolverá 0 si ha habido algún error y 1 en caso de que todo haya ido bien.

- Función `Iniciar`
  - Como la inicialización del juego se está complicando, vamos a quitarla de la función `main` y la ponemos en esta función.
  - Toma como parámetros de entrada un puntero a un estructura de tipo `Jugador`, un puntero al array de barcos y otro puntero al array de tiros.
  - Comprueba si existe el fichero de partida guardada:
    - Si no existe, llama a las funciones `IniciaArrays` e `IniciaJugador` como hasta ahora.
    - Si sí existe, llama a la función `CargaPartida`.
  - De nuevo la función devuelve 1 en caso de que todo vaya bien y 0 en caso de que haya habido algún error.

- Función `main`
  - Deberá actualizarse para hacer uso de las nuevas funciones `Iniciar` y `Detener` comprobando sus códigos de retorno.

9.2. Registrando las puntuaciones

En este apartado vamos a hacer que se mantenga un registro actualizado de las puntuaciones más altas. Para ello tras cada partida se va a ir actualizando un fichero binario con las puntuaciones.

En primer lugar crear una marco que indique el tamaño máximo del registro de puntuaciones (5) y otra con el nombre del fichero de puntuaciones. De esta forma usaremos las macros en las distintas funciones en lugar de repetir continuamente los valores, con lo que evitamos redundancia.

- Función `SalvaPuntuaciones`
  - Toma coma parámetro un puntero a un array de datos de tipo `Jugador` que contiene todas las puntuaciones.
  - Abre un fichero de puntuaciones en modo de escritura binario truncando el fichero si ya existía o creando uno nuevo en caso de que no.
  - Guarda las puntuaciones hasta llegar a una con 0 puntos o hasta alcanzar el máximo de registros indicado en la macro correspondiente.
  - Cierra el fichero.
- Función `CargaPuntuaciones`
  - Toma como parámetro un puntero a un array de datos de tipo `Jugador` previamente reservado y donde se almacenarán las puntuaciones.
  - Abre el fichero de puntuaciones en modo de lectura binario.
  - Si el fichero no existe, inicializará a 0 el campo de puntos del primer elemento del array. Esto servirá para identificar cuándo parar de leer los registros de puntuaciones.
  - Lee las puntuaciones del fichero hasta acabar el fichero o hasta llegar al máximo de registros permitidos. En caso de que se haya acabado el fichero sin llegar al máximo de registros, habrá que inicializar a 0 el campo de puntos del elemento del array siguiente al último válido, de esta forma no se considerará ese elemento ni los siguientes.
  - Cierra el fichero.
- Función `ActualizaPuntuaciones`
  - Toma como parámetros un puntero a un dato de tipo `Jugador` que representa al jugador actual y otro puntero al array de puntuaciones.
  - Se inserta la puntuación actual en la posición correspondiente. Para ello:
    - Si el jugador tiene 0 puntos la función termina sin hacer nada más ya que no se van a registrar puntuaciones nulas. Precisamente el 0 lo utilizamos para indicar que ese elemento del array y los siguientes no contienen valores válidos de puntuaciones.

- Si el jugador tiene una puntuación no nula, se busca la posición en la que iría en la tabla de puntuaciones, estando en primer lugar las puntuaciones mayores y en último las menores.
  - Si la puntuación es menor que las de la tabla, se podrá insertar la última siempre y cuando queden huecos libres en la tabla para llegar al máximo de registros indicados en la macro.
  - Al insertar la puntuación actual en un punto intermedio se debe tener en cuenta que habrá que desplazar las puntuaciones menores ya existentes en la tabla descartando la última en caso necesario.

- Función `VisualizaPuntuaciones`
  - Toma como parámetro un puntero a las puntuaciones.
  - Recorre el array de puntuaciones y las visualiza en ese orden indicando el nombre del jugador y su puntuación.

- Función `Terminar`
  - Hay que actualizar esta función para que antes de salir del juego actualice el registro de puntuaciones y muestre el resultado.
  - Para ello llamará a las nuevas funciones que hemos creado: `CargaPuntuaciones`, `ActualizaPuntuaciones`, `SalvaPuntuaciones` y finalmente `VisualizaPuntuaciones`.

A continuación se muestra un ejemplo de la salida del programa:

```
¡Bienvenido a "Hundir la Flota"!

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
A|  | 0|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
B|  |  |  |  |  | 5|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
C|  |  |  |  |  | 5|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
D|  |  |  |  |  | 5|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
E|  |  |  |  |  | 5|  |  |  |  | 4|  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
F|  |  |  |  |  | 5|  |  |  |  |  | 4|  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
G|  |  |  |  |  |  |  |  |  |  |  |  | 4|  |  |  |  |  |  |  |
```

```
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
H|  |  |  |  |  |  |  |  |  |  |  |  |  | 4|  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
I|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
J|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Fallaste!!, te quedan 0 vidas.

Puntuación de Gorka: 9

--- Game Over ---

Puntuaciones:
Gorka   9
aaa     6
Gorka   5
```

Práctica 10

Ficheros de texto

Mediante los ficheros de texto podemos almacenar información en un formato directamente legible por una persona. En esta práctica los vamos a emplear para permitir cargar pantallas diseñadas con un editor de texto externo cualquiera.

10.1. Creando la pantalla

Mediante un editor de textos[1] generar un fichero formado por números agrupados en filas y columnas según el tamaño del tablero de juego, y separados por espacios en blanco.

A continuación se muestra un ejemplo de fichero:

```
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0
0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 5 0 0 0 0
0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 5 0 0 0
0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0
0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0
3 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

[1] Si se usa un procesador de textos como OpenOffice, asegurarse de guardar el fichero en formato TXT y no ODT

10.2. Cargando la pantalla

Ahora vamos a dar la opción de que, si el usuario le pasa un parámetro al programa, éste lo interprete como el nombre de un fichero que contiene la pantalla a cargar. Si el programa se invoca sin parámetros, seguirá funcionando como hasta ahora.

- Función `CargaPantalla`
  - Toma como parámetros un puntero al array bidimensional con la posición de los barcos y una cadena de caracteres con el nombre del fichero que contiene la pantalla.
  - Como valor de retorno devuelve 1 si todo ha ido bien y 0 si ha ocurrido algún fallo.
  - Abre el fichero indicado en modo lectura de texto. Deberá informar en caso de fallo y devolver el código de error correspondiente.
  - Va rellenando el array con las posiciones de los barcos con los números contenidos en el fichero.
  - Cierra el fichero y muestra un mensaje por pantalla indicando que se ha cargado una nueva pantalla y el nombre del fichero correspondiente.
- Función `Iniciar`
  - Ahora toma como parámetro adicional el nombre del fichero a cargar.
  - Si se está retomando una partida pendiente o si el parámetro con el nombre del fichero es `NULL`, no carga la nueva pantalla.
  - En caso de que se haya indicado un nombre de fichero válido, llamará a la función `CargaPantalla` pasándole los parámetros correspondientes.
- Función `main`
  - Ahora deberá consultar los parámetros que se le pasen, para ello habrá que actualizar su prototipo de forma adecuada.
  - Cuando llame a `Iniciar`, tendrá que indicarle como nombre de fichero el que ha recibido como primer parámetro o `NULL` en caso de que no se le hayan pasado parámetros.

Ejemplo de partida

> ¡Hemos acabado la aplicación!. Ahora puedes jugar unas partidas e intercambiar pantallas con tus compañeros o, mejor aún, seguir añadiendo nuevas funcionalidades al juego ;)

Ejecutamos el juego indicando que cargue una pantalla externa del fichero `demo.txt`:

```
$ ./juego demo.txt
Pantalla "demo.txt" cargada
Dime tu nombre: Gorka

¡Bienvenido a "Hundir la Flota"!

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
A|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
B|  |  |  |  |  | 0|  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
C|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
D|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
E|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
F|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
G|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
H|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
I|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
J| 3| 3| 3|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Fallaste!!, te quedan 4 vidas.
```

```
Introduce una coordenada (ej. B12) : z0

Puntuación de Gorka: 3

--- Aplicación finalizada a petición del usuario ---

¿Deseas guardar la partida para recuperarla más adelante? (s/n): s
Guardando partida ... ok!
```

Y ahora volvemos a ejecutar el juego para que retome la partida en la que estábamos:

```
./juego
Cargando partida previa ... ok!

¡Bienvenido a "Hundir la Flota"!

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
A|  |  |  |  |  |  |  |  |  |  | 1|  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
B|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
C|  |  |  |  |  |  |  |  |  |  |  |  |  | 5|  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
D|  |  |  | 0|  |  |  |  |  |  |  |  |  |  | 5|  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
E|  |  |  |  |  |  | 4|  |  |  |  |  |  |  |  | 5|  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
F|  |  |  |  |  |  | 4|  |  |  |  |  |  |  |  |  | 5| 0|  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
G|  |  |  |  |  |  | 4|  |  |  |  |  |  |  |  |  |  | 5|  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
H|  |  |  |  |  |  | 4|  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
I|  |  |  |  |  |  |  |  | 2| 2| 0|  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
J| 3| 3| 3|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Tocado!!, vas 15 puntos.

Puntuación de Gorka: 15

¡¡¡Enhorabuena, juego superado!!!

Puntuaciones:
```

```
Gorka   15
aaa     3
Gork    3
```