Prácticas de Programación

Programar en C

Programar en C

- Este manual va dirigido a personas **con un nivel básico de programación en C**, que necesiten un recordatorio de temas elementales.

  - Se muestran los conceptos básicos de la estructura de los programas y los pasos necesarios para pasar desde el código fuente al ejecutable. También se introducen conceptos más avanzados, que no es necesario adquirir, y están marcados con una etiqueta de "sólo curiosidad".
    1. Trabajar con múltiples ficheros.
    2. Compilación/vinculación/ejecución desde línea de comandos y programas más comunes.

  - Se proponen algunos ejercicios de autoevaluación, con la idea que:
    1. Se intenten resolver con los conocimientos actuales del estudiante, utilizando las buenas prácticas de programación.
    2. Una vez resueltos, se diga la resolución propuesta paso a paso, comprobando si se ha llegado a los mismos resultados, y si se ha analizado los distintos puntos del problema.
       1. No hay una única solución buena, por lo tanto sólo se quiere que os planteéis ciertos aspectos de la resolución de un problema.
       2. La solución no se da tal cual sino que se resuelve paso a paso y haciendo que os fijes en ciertos detalles importantes de la resolución de problemas. Es muy recomendable hacer el seguimiento.
    3. Se pregunte en las aulas de laboratorio, aquellos puntos que no queden claros.

Programar en C

- Estructura de un programa en C
  - Un programa en C consiste en uno o más ficheros, los cuales contienen las declaraciones y la implementación de un conjunto de funciones y acciones.
    - Las declaraciones se suelen poner en los ficheros con extensión *.h, mientras que la implementación de les funciones y acciones suele estar en un fichero con extensión *.c
  - Tanto los ficheros *.h como los *.c son ficheros de texto, se pueden crear y editar con cualquier editor de texto que permita trabajar con texto plano (*.txt).
    - Los entornos de programación como el Dev-C++, Eclipse o el Microsoft Visual Studio entre otros, no son imprescindibles, pero facilitan la tarea del programador, automatizando los procesos de compilación y ejecución de los programas, así como la detección de errores en el código y otras herramientas más sofisticadas.
  - Tenemos a nuestra disposición gran cantidad de librerías estándar que nos permiten interactuar con el usuario y el sistema. Para utilizar acciones y funciones de las librerías estándar, deberemos importar las declaraciones. Esto se hace mediante la directiva #include.
    - Ejemplos de estas funciones predefinidas, son el *printf* que nos permite escribir en la pantalla, o el *scanf* que nos permite leer de teclado. También hay las equivalentes para leer y escribir de ficheros o inclusos para enviar o recibir datos por la red.

Programar en C

- Estructura de un programa en C
  - Todo programa en C que haya de ser ejecutado, necesita tener definida la función ***main***. En el cas más simple, ésta función puede declarar-se sin parámetros. Cuando el sistema ejecuta nuestro programa, espera saber si se ha ejecutado correctamente. Para informar de cómo ha ido la ejecución, se utilizan las constantes ***EXIT_FAILURE*** en caso de error o ***EXIT_SUCCESS*** si todo ha ido de forma correcta. Generalmente al final de la función **main** se retorna el valor de que todo ha ido como se esperaba, y en caso de error, se utiliza la función ***exit(EXIT_FAILURE)***, que acaba la ejecución e indica al sistema que ha habido un error.

```
int main(void) {
        ......
        return EXIT_SUCCESS;
}
```

  - Una opción más interesante es capturar la secuencia de parámetros pasados al programa en ser ejecutado. El parámetro argc contiene el número de parámetros de entrada, y el parámetro argv es una taula con argc posiciones, que contiene los parámetros.

```
int main(int argc, char *argv[]) {
        ......
        return EXIT_SUCCESS;
}
```

Programar en C

- Estructura de un programa en C
  - Comenzamos con el programa más simple y típico, el “Hello World”. En la versión más simple, éste programa estará dentro de un solo fichero. Fijaos que hemos añadido dos líneas al principio, importando los ficheros stdio.h y stdlib,h. Estos ficheros contienen las declaraciones y constantes más habituales. Muchos compiladores los añaden por defecto, pero si no queremos tener problemas, es bueno que siempre los añadamos al principio.

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

int main(void) {
        printf(“Hello World”);
        return EXIT_SUCCESS;
}
```

hello.c

  - Para poder ejecutar este fichero, necesitaremos compilarlo y enlazarlo. Entender estos procesos queda fuera de los objetivos de esta asignatura, pero a grandes rasgo consisten en:
    - **Compilar:** Elimina las partes no funcionales del código (espacios, comentarios, tabuladores, ...) y lo convierte en un código que la máquina pueda entender.
    - **Enlazar:** Se añaden las partes de código que necesitará nuestro programa para ser un ejecutable (librerías del sistema u otros ficheros compilados).

Programar en C

- Compilar y ejecutar un programa en C
  - Empezaremos con la forma más tradicional de compilar, la línea de comandos. Utilizaremos el compilador GCC. Éste programa lo podéis encontrar en:
    - Directorio/Carpeta ***Bin*** de la instalación del DevC++ (i.e "C:\DevCPP\Bin")
    - En internet http://gcc.gnu.org
    - En los sistemas Linux se instala por defecto en instalar los paquetes de programación.
  - Abrimos una ventana de comandos, terminal, consola. En Windows la podéis abrir ejecutando el comando "cmd.exe". En Linux y Mac generalmente hay un icono de acceso directo.
  - Si queremos compilar y enlazar directamente, ejecutamos el siguiente comando:

gcc –o <nombreEjecutable> <fichero1.c> <fichero2.c> ... <ficheroN.c>

  - Por ejemplo, en el caso del "Hello World", tendremos:

gcc –o hello hello.c

  - Como resultado obtendremos el fichero *hello.exe* (en Windows) o un fichero *hello* en Linux y/o Mac.
  - Utilizando entornos como el DevC++, todo éste proceso se hace de forma automática, aunque si tenemos más de un fichero *.c tendremos que crear un proyecto que los contenga todos.

Programar en C

- Compilar y ejecutar un programa en C
  - Para practicar un poco más, vamos a suponer que queremos definir e implementar las acciones escribirEntero y escribirHelloWorld. Dado que son métodos que utilizaremos a menudo, una buena práctica seria guardarlos para futuras ocasiones. Por consiguiente, los definiremos en ficheros a parte, los ficheros (utils.h y utils.c). Rescribimos nuestro programa "Hello World" para utilizar estas acciones:
    - El fichero utils.h contiene sólo la declaración
    - El fichero utils.c implementa las acciones
    - Para compilarlo todo haremos:

```
gcc –o hello hello.c utils.c
```

```c
#include "utils.h"

int main(void) {
    escribirHelloWorld();
    return EXIT_SUCCESS;
}
```

hello.c

```c
#include <utils.h>

void escribirEntero(int value) {
    printf("%d",value);
}
void escribirHelloWorld(void) {
    printf("Hello World");
}
```

utils.c

```c
#include <stdio.h>
#include <stdlib.h>

void escribirEntero(int value);
void escribirHelloWorld(void);
```

utils.h

Programar en C

- Compilar y ejecutar un programa en C. (**Sólo curiosidad**)
  - Para compilar y enlazar en dos etapas, podéis hacerlo con el siguiente comando:

gcc –c <fichero1.c> <fichero2.c> ... <ficheroN.c>

  - El resultado es un fichero con extensión *.o para cada fichero *.c. Éstos ficheros tienen el código en lenguaje máquina de nuestro programa. Ahora. Para enlazarlos y obtener el fichero ejecutable, utilizaremos el mismo comando que antes, pero con los ficheros *.o

gcc –o <nombreEjecutable> <fichero1.o> <fichero2.o> ... <ficheroN.o>

  - Antes de crear el código máquina final, se genera unos ficheros *.s en código ensamblador. Si queréis podéis evitar que se eliminen añadiendo el parámetro –S en la compilación:

gcc –c –S <fichero1.c> <fichero2.c> ... <ficheroN.c>

Programant en C

- Compilar y ejecutar un programa en C. (**Sólo curiosidad**)
  - Aspecto de nuestro código una vez compilado.

```
        .file   "hello.c"
gcc2_compiled.:
___gnu_compiled_c:
        .def    ___main;        .scl    2;      .type   32;     .endef
.text
        .align 4
.globl _main
        .def    _main;  .scl    2;      .type   32;     .endef
_main:
        pushl %ebp
        movl %esp,%ebp
        subl $8,%esp
        call ___main
        call _escriuHelloWorld
        xorl %eax,%eax
        jmp L2
        .p2align 4,,7
L2:
        leave
        ret
        .def    _escrbirHelloWorld;     .scl    2;      .type   32;     .endef
```

hello.s

```
L☺♥  └ ♫  ♦☺.text        î ¼  ☻  `.data
  @ └.bss                 Ç └UëÕâÞ  Þ  1└Ù ╔╞
¶ .file  ■ g☺hello.c      ♦  ☺ ♠ ‼  ☺ ♠ _ma
in  ☺  ☻ .text  ☺ ♥☺  ☻      .data  ☻ ♥☺
    .bss  ♥ ♥☺          ___main  ☻☺
%  ☻ 7 gcc2_compiled. ___gnu_compiled_c _escribirHelloWorld
```

hello.o

Programar en C

- Compilar y ejecutar un programa en C.
  - Aunque es interesante saber como se puede hacer en línea de comandos, generalmente utilizaremos un entorno de programación. En estos entornos, con un solo botón se ejecutará todo el proceso.
    - En Fundamentos de Programación se utiliza el entorno **DevC++**, el cual sigue siendo válido para esta asignatura. Podéis seguir trabajando con este, y los colaboradores docentes os darán soporte para este entorno, por lo tanto es el **entorno recomendado**.
    - Podéis utilizar entornos de programación abiertos, como **Eclipse** o **Netbeans**, que están muy implantado en entornos Linux.
    - Para entornos Windows, también podéis utilizar el **Microsoft Visual Studio**, del cual hay versión limitada gratuita, y como estudiantes de la UOC podéis acceder a la versión profesional.
  - No os podemos asegurar que los colaboradores docentes dominen todos estos entornos, pero si os resultan más cómodos podéis usarlos, siempre que:
    - Todo el código debe ser estándar, no se podrán utilizar librerías de alto nivel.
    - En caso de duda preguntad a vuestro colaborador docente de laboratorio.

Programar en C

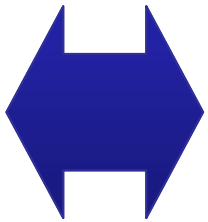
- Estructuras básicas en C
  - Composición de acciones alternativa: Se crea una bifurcación, y el flujo del programa puede seguir distintos caminos según el resultado de evaluar una expresión.

```
if (expresion) {
    acciones_True
}
```

```
if (expresion) {
    acciones_True
} else {
    acciones_False
}
```

Image by laenulfean

```
switch ( expresion ) {
    case const_val1:
            acciones_1
            break;
    case const_val2:
            acciones_2
            break;
              .....
    case constant-expresion_n:
            acciones_n
            break;
    default :
            accions_otros
}
```



```
if (expresion==const_val1) {
    acciones_1
} else if (expresion==const_val2) {
    acciones_2
                .....
} else if (expresion==const_valn) {
    acciones_n
} else {
    acciones_otros
}
```

Programar en C

- Estructuras básicas de C
  - Composición de acciones iterativa: Una parte del código se repite varias veces de forma cíclica.

```
while (expresión) {
        acciones
}
```

```
do  {
        acciones
} while(expresión);
```

Image by Olivier Bacquet

```
for ( índex = valor_inicial ; índex <= valor_final ; índex++ ) {
        acciones
}
```

Programar en C

- Estructuras básicas de C
  - Las distintas estructuras de control se pueden combinar.
  - Analizad los siguientes ejemplos de utilización de las estructuras de control, intentendo ver cómo se adaptan las expresiones a cada enunciado.
    1. Dado un valor en una variable entera a, mostrar un mensaje indicando si es par o impar.

```
if(a%2==0) {
    printf("El numero %d es par",a);
} else {
    printf("El numero %d es impar",a);
}
```

    2. Escribimos los números enteros del 1 al 10, utilizando las distintas opciones.

```
int i;
for(i=1;i<=10;i=i+1) {
        printf("%d\n",i);
}
```

```
int i;
i=1;
while(i<=10) {
        printf("%d\n",i);
        i=i+1
}
```

```
int i;
i=1;
do {
        printf("%d\n",i);
        i=i+1
} while(i<=10);
```

Programar en C

- Estructuras básicas de C

3. Mostrar para cada número del 1 al 10, si es par o impar.

```
int i;
for (i=1;i<=10;i=i+1) {
      if(i%2==0) {
            printf("El numero %d es par“,i);
       } else {
            printf("El numero %d es impar“,i);
       }
}
```

4. Creamos una función *escribirNumero*, que dado un parámetro entero *n*, con un valor entre el 1 y el 3, escriba el numero en inglés por pantalla. En caso que el numero no esté en el rango, no escribe nada. Fijaros en las dos opciones.

```
void escribirNumero(int n) {
      if(n==1) {
            printf(“one”);
       } else if(n==2) {
            printf(“two”);
       } else if(n==3) {
            printf(“three”);
      }
}
```

```
void escribirNumero (int n) {
      switch(n) {
            case 1:
                  printf(“one”);
                  break;
            case 2:
                  printf(“two”);
                  break;
            case 3:
                  printf(“three”);
                  break;
      }
}
```

Programar en C

Autoevaluación:

1. Escribe un programa en C que gestione un menú de opciones. Se mostrará la lista de opciones disponibles, identificadas con un valor numérico y se pedirá al usuario que introduzca el valor numérico de la opción que desee. Ten en cuenta los siguientes requisitos:
   a) La primera opción tiene el identificador 1.
   b) La última opción permite salir de la aplicación.
   c) Al seleccionar una opción distinta a la de salir, se mostrará un mensaje indicando que se ha seleccionado la acción correspondiente y se volverá a lista la lista de opciones.
   d) En seleccionar una opción incorrecta, se mostrará el error correspondiente, y se pedirá otra vez que se introduzca una opción, sin volver a mostrar la lista de opciones.
   e) Las opciones disponibles son: "Añadir fichero", "Eliminar fichero" y "Listar ficheros". (y la de salir)

Programar en C

Autoevaluación:

2. Siguiendo el caso propuesto en el ejercicio 1, crea los ficheros Menu.h y Menu.c, que contengan todos los métodos relacionados con la gestión del menú. Adapta tu código del ejercicio 1 para que utilice estos ficheros. Como mínimo ha de contener las siguientes funciones/acciones:

   a) Mostrar una lista de opciones, enumeradas de 1 hasta el número de opciones disponibles *nOptions*. Las opciones disponibles se pasan como una tabla de cadenas de texto *optionsDesc*. La declaración debe ser:

      void showOptions(**int** nOptions , **char** * optionsDesc[]);

      Nota: Las cadenas de texto se pueden mostrar utilizando:

      printf("%s", optionsDesc[o]);

      donde *o* es un valor entero entre 0 i *nOptions*-1.

   b) Gestionar la obtención de la opción por parte el usuario comu un parámetro de salida, y retornando un valor de error de la siguiente forma:

      errorVal getOpcions(**int** nOptions, **int*** selOption);

   donde:

   errorVal: es un tipo de datos que puede tomar valores constantes según la opción que haya introducido el usuario: OK (opción correcta), NO_NUMBER (el usuario ha introducido un valor que no es numérico), LOW_RANGE (el usuario ha introducido un valor numérico menor a 1) o UP_RANGE (el usuario ha introducido un valor numérico mayor al número de opciones).

Programar en C

Autoevaluación (Respuesta):

1. Escribe un programa en C que gestione un menú de opciones. Se mostrará la lista de opciones disponibles, identificadas con un valor numérico y se pedirá al usuario que introduzca el valor numérico de la opción que desee. Ten en cuenta los siguientes requisitos:
   a) La primera opción tiene el identificador 1.
   b) La última opción permite salir de la aplicación.
   c) Al seleccionar una opción distinta a la de salir, se mostrará un mensaje indicando que se ha seleccionado la acción correspondiente y se volverá a lista la lista de opciones.
   d) En seleccionar una opción incorrecta, se mostrará el error correspondiente, y se pedirá otra vez que se introduzca una opción, sin volver a mostrar la lista de opciones.
   e) Las opciones disponibles son: "Añadir fichero", "Eliminar fichero" y "Listar ficheros". (y la de salir)

A medida que vamos ganando práctica programando, podemos inferir muchos aspectos de nuestra solución a partir del enunciado. A continuación iremos construyendo la solución paso a paso, resaltando aquellas cosas que nos puedan ayudar:

- *Se pide un valor numérico al usuario con la opción deseada. Esto nos debe indicar que tendremos una variable entera numérica, la cual estará dentro de un dominio acotado [1,..,N], donde N es el número de opciones del menú.*
- *La aplicación tiene un comportamiento secuencial. Iremos pidiendo opciones hasta que el usuario decida salir de la aplicación. Por lo tanto, sabemos que tendremos una composición iterativa. Además, si utilizamos los conocimientos sobre composiciones iterativas, podremos ver que tenemos una secuencia de valores acabada con una opción de salir, esto es una estructura de tipo recorrido, y la podemos representar como tal.*

Programar en C

Autoevaluación (Respuesta):

- *Se pide un valor numérico al usuario con la opción deseada. Esto nos debe indicar que tendremos una variable entera numérica, la cual estará dentro de un dominio acotado [1,..,N], donde N es el número de opciones del menú.*
  - *Cuando tenemos una variable dentro de un dominio acotado, se debe verificar siempre que el valor que se le asigna es correcto y dentro de este dominio.*
    - *Se debe verificar que se ha entrado un valor numérico.*
    - *Se debe verificar que el valor numérico es mayor o igual a 1 y menor o igual a N.*
  - *Si el dominio es discreto y estático, o sea, que tenemos un conjunto de etiquetas que no cambia con el tiempo, podemos pensar en definir constantes o un tipo de datos Enumerado, que represente los distintos valores.*
    - *En este caso tenemos cuatro opciones definidas en el enunciado, por lo tanto podemos definir el Enumerado, asignando a cada valor numérico una constante.*

```
typedef   enum  {
        OPT_ADD_FILE=1,
        OPT_DEL_FILE=2,
        OPT_LIST_FILES=3,
        OPT_EXIT=4
} opcionesMenu;
```

Programar en C

Autoevaluación (Respuesta):

- *La aplicación tiene un comportamiento secuencial. Iremos pidiendo opciones hasta que el usuario decida salir de la aplicación. Por lo tanto, sabemos que tendremos una composición iterativa. Además, si utilizamos los conocimientos sobre composiciones iterativas, podremos ver que tenemos una secuencia de valores acabada con una opción de salir, esto es una estructura de tipo recorrido, y la podemos representar como tal.*

```
algoritmo recorridoEntrada
    var elem: T fvar
    inicio tratamiento
    elem:=leer();
    mientras no (elem = marca)
    hacer
        tratar elemento (elem)
        elem := leer();
    fmientras
    tratamiento final
falgoritmo
```

T será un tipo numérico sin signo.

leer() será la función de lectura de elementos. Podemos utilizar la estándar para leer enteros, *leerEntero()*, o definir nuestra propia función, que permita controlar el dominio de los valores leídos, por ejemplo *leerOpcion()*.

En el enunciado no se detalla ningún tratamiento final. El tratamiento inicial puede ser mostrar las opciones y el tratamiento de cada elemento es mostrar la acción seleccionada.

Programar en C

Autoevaluación (Respuesta):

- *Una manera de empezar a resolver un problema de forma modular, aplicando el concepto de diseño descendiente de forma intuitiva, es escribir el "main" utilizando funciones/acciones para todos los pasos que necesitamos, utilizando comentarios para explicar qué harán, y después ir repitiendo lo mismo para cada acción/función que haya aparecido, hasta que lleguemos a acciones/funciones triviales. Por ejemplo:*

```
int main(void) {
        /*Definimos la variable dónde se guardará la opción. El tipo dependerá de las opciones que
        tengamos. Puede ser un enumerado o un entero.*/
        opcionesMenu opcion;

        /* Mostramos el menú de opciones y pedimos al usuario que entre una. En caso que la entrada sea
        incorrecta, continuamos pidiendo una opción hasta que el valor entrado sea correcto. */
        opcion=leerOpcion();

        /* Vamos analizando las opciones elegidas hasta que se pida salir. Definiremos la marca de salida
        con la constante OPT_EXIT. */
        while (opcion!= OPT_EXIT) {
                /* Hacemos las acciones definidas para cada opción */
                hacerAcciones(opcion);
                /* Pedimos otra opción. */
                opcion=leerOpcion();
        }
        /* Finalizamos la ejecución del programa*/
        return EXIT_SUCCESS;
}
```

Ejercicio1.c

Programar en C

Autoevaluación (Respuesta):

- *Los tipos de datos y las acciones/funciones que vamos utilizando, las iremos declarando en el fichero de cabeceras (*.h). Junto a la declaración de cada acción/función, podemos mantener la descripción de qué hacen. Por ejemplo:*

```
/*Definimos el tipo de datos utilizado para guardar las opciones del menú. Por ejemplo, con un enumerado,
comentando cada opción.*/
typedef enum {
        OPT_ADD_FILE=1, /*Opción para añadir un fichero.*/
        OPT_DEL_FILE=2, /*Opción para eliminar un fichero.*/
        OPT_LIST_FILES=3, /* Opción para listar los ficheros.*/
        OPT_EXIT=4 /* Opción de salir de la aplicación. */
} opcionesMenu;

/* Método principal. Se encarga de la lógica de la aplicación. */
int main(void);

/* Mostramos el menú de opciones y pedimos al usuario que entre una. En caso que la entrada sea
incorrecta, continuamos pidiendo una opción hasta que el valor entrado sea correcto. */
opcionesMenu leerOpcion(void);

/* Hacemos las acciones definidas para cada opción */
void hacerAcciones(opcionesMenu opcion);
```

Ejercicio1.h

Programar en C

Autoevaluación (Respuesta):

- *Importamos los ficheros de cabecera necesarios para que todo funcione. Esta tarea se hace con la directiva "include", a la que se le pasa el fichero correspondiente. En nuestro caso, importaremos dos ficheros estándar y el que hemos creado con nuestras definiciones. Fijaros que se hace de forma ligeramente distinta. La diferencia es que en utilizar "comillas", le decimos que empiece a buscar por el directorio/carpeta donde se encuentra el fichero de código, y no por los directorios/carpetas estándares.*

```
#include <stdio.h>
#include <stdlib.h>
#include "Ejercicio1.h"

int main(void) {
   ....
        return EXIT_SUCCESS;
}
```

Ejercicio1.c

stdio.h contiene las declaraciones de los métodos de "entrada/salida", que son los que permiten escribir y leer de los dispositivos estándar (pantalla y teclado o ficheros).

stdlib.h contiene las declaraciones de los métodos básicos, así como declaraciones de constantes de sistema, como por ejemplo la que indica que se ha finalizado la aplicación correctamente (EXIT_SUCCESS).

Ejercicio1.h es el fichero que hemos creado nosotros. Fijaros que este lo añadimos utilizando comillas en lugar de los operadores de desigualdad < y >.

Programar en C

Autoevaluación (Respuesta):

- *Además de las declaraciones, ahora añadiremos al fichero de código las implementaciones de las acciones y funciones que hemos declarado, añadiendo un mensaje que indique que aún no están hechas, pero retornando un valor por defecto, que permita compilar y ejecutar la aplicación. También es útil dejar la descripción de lo que se debe hacer, con una indicación TODO (a hacer en inglés), indicando que aún se debe implementar.*

```
int main(void) {
   ....
}

opcionesMenu leerOpcion(void) {
        /*Mostramos el menú de opciones y pedimos al usuario que entre una. En caso que la
        entrada sea incorrecta, continuamos pidiendo una opción hasta que el valor entrado sea
        correcto. */
        printf("TODO: Función leerOpcion\n");

        return OPT_EXIT;
}

void hacerAcciones(opcionesMenu opcion) {
        /* Hacemos las acciones definidas para cada opción*/
        printf("TODO: Acción hacerAcciones\n");
}
```

Ejercicio1.c

Programar en C

Autoevaluación (Respuesta):

- *En estos momentos nuestro programa ya se puede compilar y ejecutar, asegurando que la estructura/esqueleto de la aplicación es correcto. Como es de esperar, no funcionará, ya que no hemos implementado nada, pero nos permitirá comprobar que hemos declarado correctamente todo lo que necesitamos. La salida debería ser la siguiente:*

```
>> Exercici1
        TODO: Funció llegirOpcio
>>
```

- *Dado que la opción por defecto que hemos puesto es la de salir, nunca de llamará a la acción para tratar las opciones. Vamos a implementar la función para leer la opción. Al igual que hemos hecho en el caso del main, indicaremos los pasos que debemos seguir, pero ahora solo utilizaremos comentarios, para decidir si declaramos una nueva acción/función para solucionarlo o lo hacemos directamente en esta.*

```
opcionesMenu leerOpcion(void) {

        /* Mostramos la lista de opciones*/
        /* Vamos pidiendo opciones al usuario hasta que entre una de correcta.*/
        /* Convertimos la entrada por teclado al retorno deseado. */
}
```

Ejercicio1.c

Programar en C

Autoevaluación (Respuesta):

- *Aunque mostrar la lista de opciones lo podemos hacer directamente, ya que simplemente es mostrar cadenas de texto por pantalla, dado que quizás nos puede interesar poner un titulo o quizás hacer alguna otra acción previa, como limpiar la pantalla antes de mostrar el menú, o que lo que sea, vamos a definir una acción a parte (mostrarOpciones).*
- *Pedir una opción al usuario y validar que sea correcta, incluye conversiones de tipo y una composición iterativa, ya que debemos analizar todas las opciones hasta que encontremos un valor correcto. Al igual que en el caso anterior, podríamos ponerlo en el mismo cuerpo de la función, pero vamos a definir una función auxiliar, que se encargue de leer un entero en el rango [a,b]. En caso que el valor entrado sea incorrecto, retornará un valor a-1, que dado que está fuera del dominio, nos servirá de valor de error. (getDomainValue).*
- *La conversión de tipo de datos la haremos en el propio método, sabiendo que hemos asignado a las constantes del enumerado los valores que les corresponde en el menú. En caso contrario, deberíamos hacer la conversión de forma manual (varios bloques if-else o un bloque switch).*

Programar en C

Autoevaluación (Respuesta):

- *Con los cambios propuestos, el método leerOpcion quedaría de la siguiente manera:*

```
opcionesMenu leerOpcion(void) {
        int opcion=-1;
        opcionesMenu opcionRet;

        /* Mostramos la lista de opciones*/
        mostrarOpciones();

        /* Vamos pidiendo opciones al usuario hasta que nos dé una de correcta.*/
        printf("Entra la opción deseada: ");
        opcion=getDomainValue(1,4);
        while(opcion<1) {
                printf("Entra la opción deseada: ");
                opcion=getDomainValue(1,4);
        }

        /* Convertir la entrada por teclado al valor de retorno deseado. */
        opcionRet=(opcionesMenu)opcion;

        return opcioRet;
}
```

Ejercicio1.c

Programar en C

Autoevaluación (Respuesta):

- *También añadimos las declaraciones y implementaciones de los métodos auxiliares:*

```
/* Mostramos la lista de opciones*/
void mostrarOpciones(void);

/* Leemos un valor entero por teclado, y valida que esté en el rango [valMin,valMax].
En caso contrario retorna valMin-1*/
int getDomainValue(int valMin,int valMax);
```

Ejercicio1.h

```
void mostrarOpciones(void) {
    /* Mostramos la lista de opciones*/
    printf("TODO: Acción mostrarOpciones\n");
}

int getDomainValue(int valMin,int valMax) {
    /* Leemos un valor entero por teclado, y valida que esté en el rango [valMin,valMax].
    En caso contrario retorna valMin-1*/
    printf("TODO: Función getDomainValue\n");

    return valMin-1;
}
```

Ejercicio1.c

Programar en C

Autoevaluación (Respuesta):

- *Definimos el método mostrarOpciones para que muestre un titulo y la lista de opciones descrita en el enunciado:*

```
void mostrarOpciones(void) {
    /* Mostramos la lista de opciones*/
    printf("============================\n");
    printf("====          MENU          ====\n");
    printf("============================\n");
    printf("\n");
    printf("%d.- %s\n", 1 , "Añadir Fichero");
    printf("%d.- %s\n", 2 , "Eliminar Fichero ");
    printf("%d.- %s\n", 3 , "Listar Ficheros ");
    printf("%d.- %s\n", 4 , "Salir");
}
```

Ejercicio1.c

Programar en C

Autoevaluación (Respuesta):

- *Finalmente, en la siguiente y última iteración de éste proceso de desarrollo, implementaríamos el método getDomainValue, donde se ha decidido no añadir ningún método auxiliar:*

```
int getDomainValue(int valMin,int valMax) {
     int retVal;

    /* Leemos un valor entero por teclado, y valida que esté en el rango [valMin,valMax].
    En caso contrario retorna valMin-1*/

    /* Pedimos la opción */
    printf("Entra  una opcion: ");
    scanf(" %d",&retVal);

    /* Comprobamos el valor entradao */
    if(retVal<valMin || retVal>valMax) {
          retVal=valMin-1;
    }

    return retVal;
}
```

Ejercicio1.c

Programar en C

Autoevaluación (Respuesta):

2. Siguiendo el caso propuesto en el ejercicio 1, crea los ficheros Menu.h y Menu.c, que contengan todos los métodos relacionados con la gestión del menú. Adapta tu código del ejercicio 1 para que utilice estos ficheros. Como mínimo ha de contener las siguientes funciones/acciones:
   a) Mostrar una lista de opciones, enumeradas de 1 hasta el número de opciones disponibles *nOptions*. Las opciones disponibles se pasan como una tabla de cadenas de texto *optionsDesc*. La declaración debe ser:

      void showOptions(**int** nOptions , **char** * optionsDesc[]);

      Nota: Las cadenas de texto se pueden mostrar utilizando:

      printf("%s", optionsDesc[o]);

      donde *o* es un valor entero entre 0 i *nOptions*-1.

Programar en C

Autoevaluación (Respuesta):

2.

a) El método showOptions será muy similar al método mostrarOpciones del ejercicio 1, solo que se le pasa las opciones como parámetro. Declaramos el método y lo implementamos:

```
/* Show list of options */
void showOptions(int nOptions, char* optionsDesc[]);
```

Menu.h

```
#include "Menu.h"

void showOptions(int nOptions, char* optionsDesc[]) {
    /* Show Header*/
    printf("============================\n");
    printf("====          MENU          ====\n");
    printf("============================\n");
    printf("\n");

    /* Show the list of options */
    for(i=1;i<=nOptions;i++) {
            printf("%d.- %s\n", i , optionsDesc[i]);
    }
}
```

Menu.c

Programar en C

Autoevaluación (Respuesta):

b) Para poder retornar el conjunto de errores, debemos crear un nuevo tipo enumerado, que contenga estos errores. Lo declaramos en el fichero Menu.h, junto con el método getOption:

errorVal getOption(**int** nOptions, **int** &selOption);

```
/* Show list of options */
void showOptions(int nOptions, char* optionsDesc[]);

/* Error values for input options */
typedef enum  {OK,NO_NUMBER, LOW_RANGE, UP_RANGE} errorVal;

/* Get an option value in the range [1,nOptions] */
errorVal getOption(int nOptions, int &selOption);
```

Menu.h

Programar en C

Autoevaluación (Respuesta):

b) El método será similar al método getDomainValue del ejercicio 1, pero en este caso cambiará:

1. minVal siempre es 1.
2. Debemos comprobar si se ha entrado un número. En el ejercicio 1 asumíamos que el usuario entraba un número. Para hacer esta comprobación utilizaremos el número que retorna el método scanf, que corresponde al número de elementos leídos. Si no nos han entrado ningún número, este valor será cero. También podríamos leerlo como una cadena de caracteres y comprobar que cada carácter es un calor correcto, utilizando su código ASCII.
    1. Se debe tener en cuenta que si la cadena de entrada no es un valor numérico, no se lee nada, y por lo tanto queda en el buffer de teclado y lo tenemos que eliminar. Por lo tanto, en caso de error, leeremos todo lo que se haya escrito como una cadena de caracteres, pero no la asignaremos a ninguna variable (se descarta).
3. Cuando nos hemos asegurado que es un valor numérico, comprobamos que está dentro del dominio, pero ahora consideramos los dos casos por separado.
4. Finalmente leemos un carácter, para eliminar el salto de línea. Siempre se debe tener en cuenta este carácter, ya que si no se lee, se leerá la siguiente vez que leamos de teclado..

Programar en C

Autoevaluación (Respuesta):

b) El método quedaría como:

```
/* Get an option value in the range [1,nOptions] */
errorVal getOption(int nOptions, int &selOption) {
        /* Get input value and check if it is a number */
        if(scanf("%d",&selOption)<1) {
                /* Read the input and return the error */
                scanf("%s");
                return NO_NUMBER;
        }
        /* Check lower bound of domain */
        if(selOption<1) {
                return LOW_RANGE;
        }
        /* Check upper bound of domain */
        if(selOption>nOptions) {
                return UP_RANGE;
        }
        /* Remove new line char*/
        getchar();
        /* Return correct value*/
        return OK;
}
```

Menu.c

Programar en C

Autoevaluación (Respuesta):

- *Para utilizar el nuevo código, primeramente copiaremos el contenido de Ejercicio1.h y Ejercicio1.c a los ficheros Ejercicio2.h y Ejercicio2.c. Ahora incluiremos el fichero Menu.h desde el fichero Ejercicio2.c y tendremos que modificar el método leerOcion de la siguiente manera:*

```
#include "Menu.h"

opcionesMenu leerOpcion(void) {
        int option;

        /*Mostramos el menú de opciones y pedimos al usuario que entre una. En caso que la
        entrada sea incorrecta, continuamos pidiendo una opción hasta que el valor entrado sea
        correcto. */
        char* optionsList[]={"Añadir Fichero","Eliminar Fichero", "Listar Ficheros", "Salir"};

        /* Mostramos las opciones */
        showOptions(4,optionsList);

        /* Leemos la opción */
        while(getOption(4, option)!=OK);

        return (opcionesMenu) option;
}
```

Ejercicio2.c

Programar en C

Autoevaluación (Respuesta):

- *Si queremos tratar los distintos errores al introducir una opción, podemos modificar el while, cambiándolo por un do-while y un switch que muestre un mensaje explicativo de cada error. Por ejemplo:*

```
opcionesMenu leerOpcion(void) {
        int option;
        errorVal retVal;
        ....
        /* Leemos la opción */
        do {
                /* Get option value */
                retVal=getOption(4, option);
                /* Show error message */
                switch(retVal) {
                    case NO_NUMBER:
                        printf("Error: Se debe introducir un valor numérico\n");
                        break;
                    case LOW_RANGE :
                    case UP_RANGE :
                        printf("Error: Has entrado una opción inexistente.\n");
                        break;
                }
        } while(retVal!=OK);

        return (opcionesMenu) option;
}
```

Ejercicio2.c