Ejemplo diseño descendente

Enunciado:

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)

Ejemplo diseño descendente

Solución:

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.*

Ejemplo diseño descendente

Solución:

- *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;
```

Ejemplo diseño descendente

Solución:

- *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.

Ejemplo diseño descendente

Solución:

- *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

Ejemplo diseño descendente

Solución:

- *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

Ejemplo diseño descendente

Solución:

- *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 >.

Ejemplo diseño descendente

Solución:

- *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

Ejemplo diseño descendente

Solución:

- *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

Ejemplo diseño descendente

Solución:

- *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).*

Ejemplo diseño descendente

Solución:

- *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

Ejemplo diseño descendente

Solución:

- *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

Ejemplo diseño descendente

Solución:

- *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

Ejemplo diseño descendente

Solución:

- *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