Práctica 2

Aplicación de Estándares de Codificación

Autores:
Andrés Rojas Ortega 77382127F
Esteban Jódar Pozo 26003112W

Índice

1. Introducción

La aplicación seleccionada para el desarrollo de la primera práctica de calidad del software se trata de una aplicación destinada a la gestión de una biblioteca.

Esta aplicación se desarrollo para una de las prácticas de la asignatura Estructuras de datos, impartida en el segundo curso del grado de Ingeniería informática ofertado en la Universidad de Jaén. Los alumnos que realizaron esta práctica fueron Esteban Jódar Pozo (integrante de este grupo de prácticas) y Julian Yopis Ruiz.

Esta aplicación permite dar de alta a usuarios, los cuales pueden registrarse y hacer pedidos de libros tanto por temática, nombre o ISBN. Además, tiene un esquema de administrador, en el cual se podrá controlar aspectos relativos a pedidos, usuarios, libros que han solicitado los usuarios, etc.

Por tanto, tiene dos esquemas de entrada, de Administrador y de Usuario.

El administrador, una vez introducida su clave, tendrá acceso a:

- Crear pedidos para la biblioteca y tramitarlos.
- Cerrar dichos pedidos una vez finalizados.
- Ver los pedidos que tenga pendiente un usuario en concreto y tramitarlos.

Y un usuario, si no está registrado lo puede hacer, y si ya lo está:

- Puede introducir login y contraseña.
- Consultar un libro.
- Realizar un pedido.

Las estructuras de datos principales que sirven de soporte a la aplicación son listas simples enlazadas tanto de usuarios, como de libros, como de pedidos de usuario y pedidos de biblioteca.

2. Selección de los estándares a seguir

A continuación se presentan, a modo de lista no numerada, todos los estándares que hemos decidido aplicar a nuestro proyecto:

- **DCL51-CPP**. No declare ni defina un identificador reservado.
- **DCL52-CPP**. Nunca califique un tipo de referencia con constante o volátil.
- **OOP53-CPP**. Escribir los inicializadores de miembros de los constructores en el orden canónico.
- **MSC52-CPP**. Las funciones que devuelven un valor deben devolver un valor desde todas las rutas de salida.
- **FIO51-CPP**. Cerrar los archivos cuando ya no sean necesarios.

3. Aplicación de estándares sobre el proyecto

3.1. DCL51-CPP. No declare ni defina un identificador reservado

3.1.1. Enlace al sitio web del estándar seleccionado

`https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL51-CPP.+Do+not+declare+or+define+a+reserved+identifier`

3.1.2. Explicación sobre su utilidad

El estándar de C++ (ISO/IEC 14882-2014) que hace referencia a los nombres reservados especifica las siguientes reglas:

- "*Una unidad de traducción que incluye un encabezado de biblioteca estándar no debe #define o #undef nombres declarados en ningún encabezado de biblioteca estándar.*
- *Una unidad de traducción no debe #define o #undef nombres léxicamente idénticos a palabras clave, a los identificadores enumerados en la Tabla 3, o a los atributos-tokens descritos en 7.6.*
- *Cada nombre que contenga un guión bajo doble _ _ o comience con un guión bajo seguido de una letra mayúscula está reservado a la implementación para cualquier uso.*
- *Cada nombre que comienza con un guión bajo se reserva para la implementación para su uso como nombre en el espacio de nombres global.*

- *Cada nombre declarado como un objeto con enlace externo en un encabezado está reservado a la implementación para designar ese objeto de biblioteca con enlace externo, tanto en el espacio de nombres estándar como en el espacio de nombres global.*
- *Cada firma de función global declarada con enlace externo en un encabezado está reservada a la implementación para designar esa firma de función con enlace externo.*
- *Cada nombre de la biblioteca C estándar declarado con enlace externo está reservado a la implementación para su uso como un nombre con enlace C externo, tanto en el espacio de nombres estándar como en el espacio de nombres global.*
- *Cada firma de función de la biblioteca C estándar declarada con enlace externo está reservada a la implementación para su uso como firma de función con enlace externo C y externo C++, o como nombre del ámbito del espacio de nombres en el espacio de nombres global.*
- *Para cada tipo T de la biblioteca C estándar, los tipos ::T y std::T están reservados para la implementación y, cuando se defina, ::T será idéntico a std::T.*
- *Los identificadores de sufijos literales que no comienzan con un guión bajo están reservados para una futura estandarización.*"

Los identificadores y nombres de atributos a los que se hace referencia en el extracto anterior son override, final, alignas, carry_dependency, deprecated y noreturn. Ningunos otros identificadores están reservados.

Declarar o definir identificadores en un contexto en el que estén reservados derivará en un comportamiento indefinido o impredecible. Para evitar esto, siempre hay que evitar reservar o definir identificadores que estén reservados.

3.1.3. Aplicación del estándar al proyecto

Nuestro proyecto cumple perfectamente con el estándar, ya que no hace uso de ningún identificador reservado. Uno de los ejemplos de las reglas definidas en el apartado anterior es la nomenclatura de las cabeceras de los archivos, a continuación una captura de pantalla de una de ellas:

```
1  /**
2   * @file PedidoBiblioteca.h
3   * @brief Archivo cabecera donde se almacena toda la información relacionada con la clase PedidoBiblioteca.
4   */
5
6  #ifndef PEDIDOBIBLIOTECA_H
7  #define PEDIDOBIBLIOTECA_H
```

Figura 1: Captura de pantalla del cumplimiento del estándar DCL51-CPP.

Otro ejemplo de cumplimiento del estándar es el uso del identificador T en la implementación de las plantillas del programa. A continuación, una captura de pantalla de dicho uso:

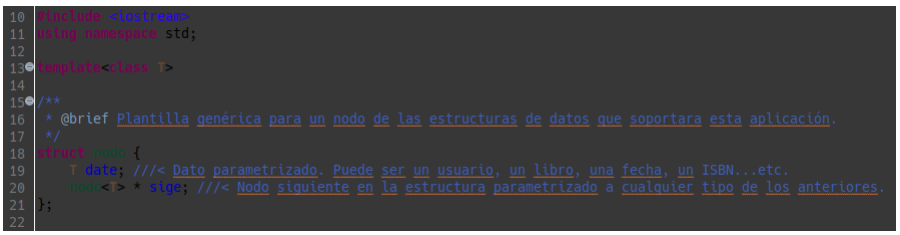


Figura 2: Captura de pantalla del cumplimiento del estándar DCL51-CPP.

Estos son los dos ejemplos más claros de cumplimiento del estándar. No podemos mostrar más capturas de pantalla ya que, al no hacer uso de identificadores reservados, no podemos mostrar las correcciones de los incumplimientos.

3.2. DCL52-CPP. Nunca califique un tipo de referencia con constante o volátil

3.2.1. Enlace al sitio web del estándar seleccionado

`https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL52-CPP.+Never+qualify+a+reference+type+with+const+or+volatile`

3.2.2. Explicación sobre su utilidad

Como sabemos de cursos anteriores, C++ no permite modificar el valor de una variable que haya sido calificada con constante (const).

En el caso de objetos que sea referenciados, es decir, utilizando el caracter reservado '&' se puede cometer el error de escribir la expresión de la siguiente forma: "*char const& p*", C++ ignora o prohibe la asignación de referencia a la palabra reservada çonst". Este hecho puede desencadenar en escrituras accidentales en la variable constante, probocando resultados no esperados en la ejecución del programa.

Para que la expresión fuera correcta, debería escribirse de una de las siguientes formas: "*const char &p*"ó "*char const &p*"

3.2.3. Aplicación del estándar al proyecto

En nuestro proyecto hacemos uso repetidas veces de expresiones con referencias de este tipo, y en todas ellas respetamos la sintaxis que se describe en el presente estándar.

A continuación, presentamos capturas de pantalla de alguna de las expresiones que se encuentran en el código fuente a modo de ejemplo:

```
/**
 * @brief Comparar fechas.
 * @param [in] f Fecha(dir, const).
 * @return bool. True en el caso de que la fecha actual sea menor que la fecha parámetro, false en cualquier otro caso.
 */
bool Fecha::operator<(const Fecha &f) {
    if (anio < f.anio)
        return true;
    else if (anio > f.anio)
        return false;

    if (mes < f.mes)
        return true;
    else if (mes > f.mes)
        return false;

    if (dia < f.dia)
        return true;
    else if (dia > f.dia)
        return false;

    if (hora < f.hora)
        return true;
    else if (hora > f.hora)
        return false;

    if (min < f.min)
        return true;

    return false;
}
```

Figura 3: Captura de pantalla del cumplimiento del estándar DCL52-CPP.

```
/**
 * @brief Función auxiliar de conversión desde estructura de tiempo tm de time.h.
 * @param [out] t tm (const, dir).
 */
void Fecha::leerTiempo(const tm &t) {
    dia = t.tm_mday;
    mes = t.tm_mon + 1;
    anio = t.tm_year + 1900;
    hora = t.tm_hour;
    min = t.tm_min;
}
```

Figura 4: Captura de pantalla del cumplimiento del estándar DCL52-CPP.

3.3. OOP53-CPP. Escribir los inicializadores de miembros de los constructores en el orden canónico.

3.3.1. Enlace al sitio web del estándar seleccionado

`https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP53-CPP.+Write+constructor+member+initializers+in+the+canonical+order`

3.3.2. Explicación sobre su utilidad

Este estándar nos dice que la lista de inicializadores de miembros para un constructor permite que los miembros se inicialicen a valores especificados y que los constructores de clases base sean llamados con argumentos específicos.

Es decir, en el caso de que un atributo miembro de la clase necesite el valor de otro atributo miembro, este debe de estar inicializado previamente. El atributo miembro dependiente irá después del miembro del que depende en la lista de atributos y en el constructor.

En el caso de que no se cumpliera este estándar y en el constructor el miembro dependiente se intentara inicializar antes de que se haya inicializado el miembro del que depende, daría lugar a un comportamiento indefinido, como por ejemplo leer memoria no inicializada (datos basura).

En definitiva, se deben de escribir siempre los inicializadores de miembros en un constructor en el orden canónico: primero las clases base directas en el orden en que aparecen en la lista de especificador-base para la clase, luego los miembros de datos no estáticos en el orden en que se declaran en la definición de la clase.

3.3.3. Aplicación del estándar al proyecto

En nuestro proyecto se encuentran las siguientes clases:

- Aplication
- Biblioteca
- Fecha
- Libro
- lista_sin
- PedidoBiblioteca
- PedidoUsuario
- Usuario

Para la clase Aplication, tenemos la siguiente lista de miembros:

```
/**
 * @brief Clase principal la cual derivará en sus variantes de admin y de usuario.
 */
class Aplication {
    Biblioteca bl;                          ///< Objeto de la clase Biblioteca.
    Usuario *usu;                           ///< Referencia a un usuario en concreto.
    lista_sin<Usuario> lusu;                ///< Lista de los usuarios registrados en la biblioteca.
    Libro li;                               ///< Objeto de la clase libro.
    PedidoBiblioteca *pedBi;                ///< Referencia a un pedido específico hecho por la biblioteca
    lista_sin<PedidoBiblioteca *> * pedbi;  ///< Lista de todos los libros pedidos en cada solicitud.
    lista_sin<PedidoUsuario *> * pedusu;    ///< Lista de los pedidos hechos por los usuarios.
    lista_sin<Libro *> * libro;             ///< Lista de todos los libros pedidos por todos los usuarios.
    PedidoBiblioteca *pedbipunt;            ///< Lista de los pedidos de la biblioteca no tramitados.
```

Figura 5: Captura de pantalla de los atributos miembros de la clase Aplication.

En la siguiente captura de pantalla se puede apreciar cómo no se cumple claramente con el estándar, ya que la inicialización de los atributos miembros de la clase no siguen el orden establecido en la declaración de la clase y hay atributos que no se inicializan.

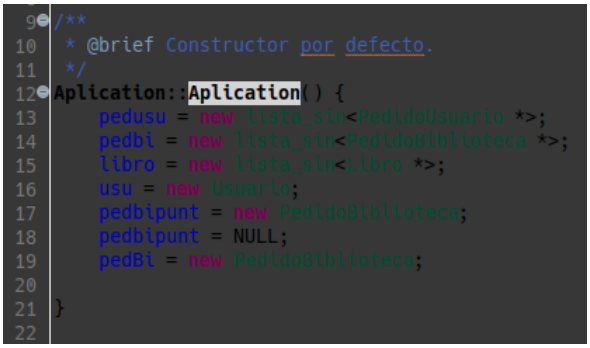


Figura 6: Captura de pantalla del incumplimiento del estándar OOP53-CPP.

A continuación, se ha incluido la inicialización de los atributos que no aparecían y se han inicializado todos los atributos en el orden con el que se declaran en la clase:

Figura 7: Captura de pantalla del cumplimiento del estándar OOP53-CPP.

Para la clase Biblioteca, tenemos la siguiente lista de miembros:

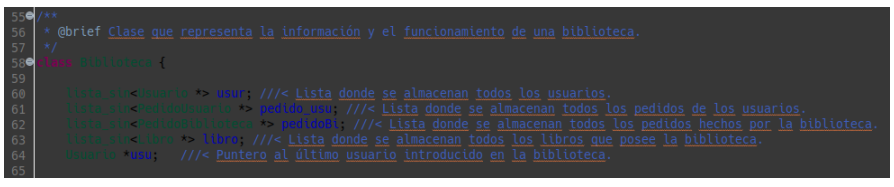


Figura 8: Captura de pantalla de los atributos miembros de la clase Biblioteca.

En la siguiente captura de pantalla se puede apreciar cómo sí se cumple claramente con el estándar, ya que la inicialización de los atributos miembros de la clase siguen el orden establecido en la declaración de la clase y se inicializan todos los atributos.

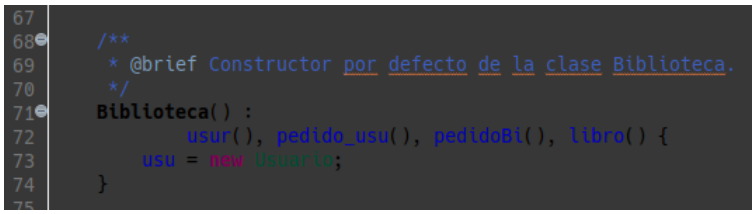


Figura 9: Captura de pantalla del cumplimiento del estándar OOP53-CPP.

Para la clase Fecha, tenemos la siguiente lista de miembros:

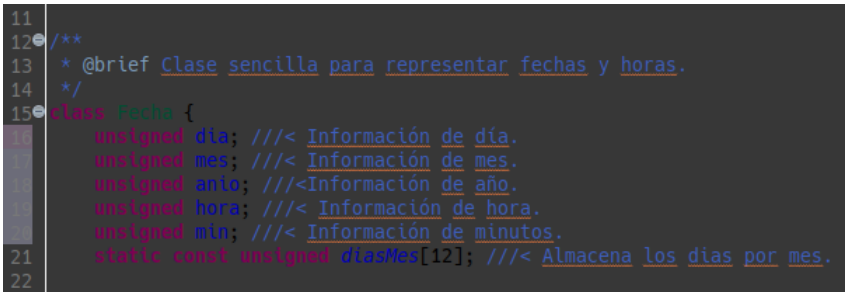


Figura 10: Captura de pantalla de los atributos miembros de la clase Fecha.

En la siguiente captura de pantalla se muestra el código fuente del constructor por defecto de la clase. Se puede observar que se delega la inicialización de las variables en una función externa, por lo que daremos por no cumplido el estándar.

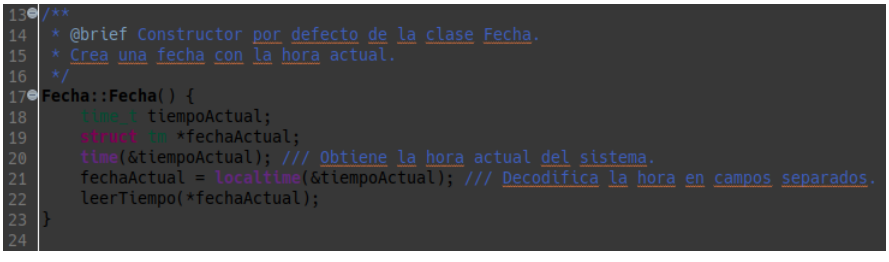
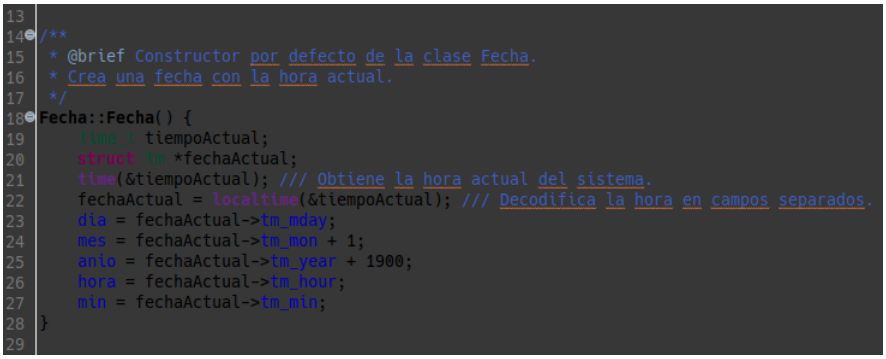


Figura 11: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

Para corregirlo, añadimos la inicialización de los atributos miembros de la clase al final, en el orden de declaración de los mismos:



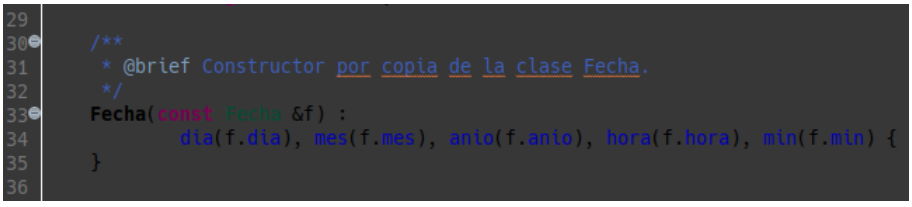
```
13
14  /**
15   * @brief Constructor por defecto de la clase Fecha.
16   * Crea una fecha con la hora actual.
17   */
18  Fecha::Fecha() {
19      time_t tiempoActual;
20      struct tm *fechaActual;
21      time(&tiempoActual); /// Obtiene la hora actual del sistema.
22      fechaActual = localtime(&tiempoActual); /// Decodifica la hora en campos separados.
23      dia = fechaActual->tm_mday;
24      mes = fechaActual->tm_mon + 1;
25      anio = fechaActual->tm_year + 1900;
26      hora = fechaActual->tm_hour;
27      min = fechaActual->tm_min;
28  }
29
```

Figura 12: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

Tanto para el constructor parametrizado como para el constructor por copia, se puede observar en las siguientes capturas de pantalla que se cumple con el estándar:

```
29
30  /**
31   * @brief Constructor parametrizado de la clase Fecha.
32   * Crea una fecha concreta. Devuelve una excepción ErrorFechaIncorrecta si la fecha introducida no es correcta.
33   */
34  Fecha::Fecha(unsigned aDia, unsigned aMes, unsigned aAnio, unsigned aHora,
35          unsigned aMin) {
36      comprobarFecha(aDia, aMes, aAnio, aHora, aMin); /// Filtra las fechas incorrectas.
37      dia = aDia;
38      mes = aMes;
39      anio = aAnio;
40      hora = aHora;
41      min = aMin;
42  }
43
```

Figura 13: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.



```
29
30      /**
31       * @brief Constructor por copia de la clase Fecha.
32       */
33      Fecha(const Fecha &f) :
34              dia(f.dia), mes(f.mes), anio(f.anio), hora(f.hora), min(f.min) {
35      }
36
```

Figura 14: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

Para la clase Libro, tenemos la siguiente lista de miembros:

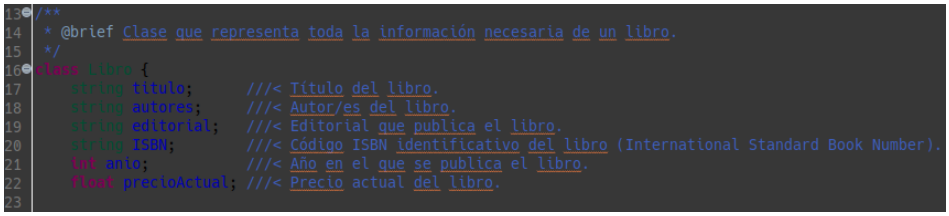


Figura 15: Captura de pantalla de los atributos miembros de la clase Libro.

En la siguiente captura de pantalla se puede comprobar que en el constructor por defecto de la clase no se cumple el estándar, ya que no respeta el orden de los atributos miembros de la clase:

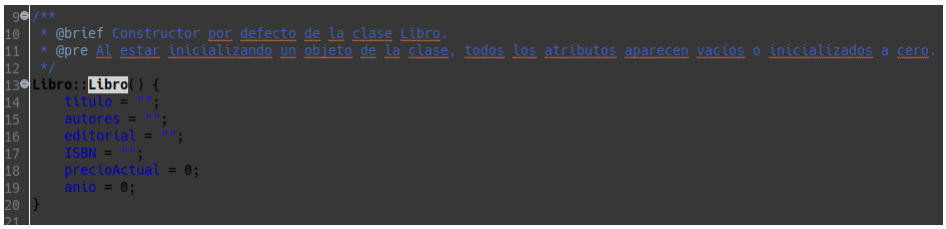


Figura 16: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

Esto se soluciona reordenando la inicialización en el orden correcto:

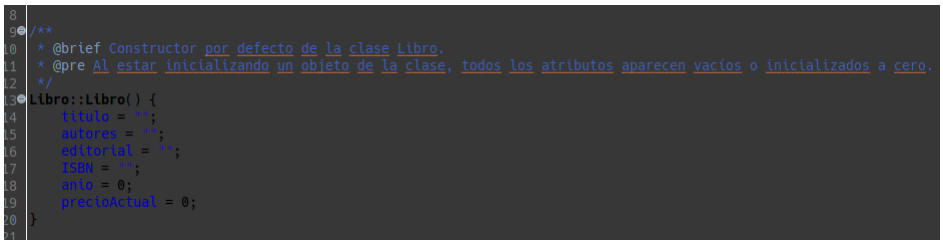


Figura 17: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

Para el constructor parametrizado y el constructor por copia observamos que sí se cumple el estándar:

```cpp
/**
 * @brief Constructor parametrizado de la clase Libro.
 * @param [in] aTitulo string. Título del libro.
 * @param [in] aAutores string. Autor/es del libro.
 * @param [in] aEditorial string. Editorial del libro.
 * @param [in] aISBN string. ISBN del libro.
 * @param [in] aAnio int. Año de publicación del libro.
 * @param [in] aPrecioActual float. Precio del libro.
 */
Libro::Libro(string aTitulo, string aAutores, string aEditorial, string aISBN, int aAnio, float aPrecioActual) {
    titulo = aTitulo;
    autores = aAutores;
    editorial = aEditorial;
    ISBN = aISBN;
    anio = aAnio;
    precioActual = aPrecioActual;
}

```

Figura 18: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

```cpp

    /**
     * @brief Constructor por copia de la clase Libro.
     * @param [in] lib Libro(dir). Instancia de la clase Libro de la cual se va realizar una copia.
     */
    Libro(const Libro &lib) {
        this->titulo = lib.titulo;
        this->autores = lib.autores;
        this->editorial = lib.editorial;
        this->ISBN = lib.ISBN;
        this->anio = lib.anio;
        this->precioActual = lib.precioActual;
    }

```

Figura 19: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

Para la clase plantilla lista_sin, tenemos la siguiente lista de miembros:

```cpp

template<class T>
/**
 * @brief Plantilla genérica de estrucutura de datos Lista enlazada.
 */
class lista_sin {
    nodo<T> *nuevo;   ///< Nuevo nodo en la Lista enlazada.
    nodo<T> *primero; ///< Primer nodo en la Lista enlazada.
    nodo<T> *ultimo;  ///< Último nodo en la Lista enlazada.
    unsigned numElem; ///< Número de nodos de la Lista enlazada.

```

Figura 20: Captura de pantalla de los atributos miembros de la clase lista_sin.

Podemos observar en las siguientes capturas de pantalla que tanto el constructor por defecto como el constructor por copia cumplen con el estándar:

```
109
110 template<class T>
111 lista_sin<T>::lista_sin() {
112     nuevo = NULL;
113     primero = NULL;
114     ultimo = NULL;
115     numElem = 0;
116 }
117
```

Figura 21: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

```
117
118 template<class T>
119 lista_sin<T>::lista_sin(lista_sin &list) {
120     nuevo = NULL;
121     primero = NULL;
122     ultimo = NULL;
123     numElem = 0;
124     for (unsigned i = 0; i < list.tamanio(); i++) {
125         this->aumenta(list.lee(i));
126     }
127 }
128
```

Figura 22: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

Para la clase clase PedidoBiblioteca, tenemos la siguiente lista de miembros:

```
12
13 /**
14  * @brief Clase que representa de manera genérica un pedido hecho por la biblioteca.
15  */
16 class PedidoBiblioteca {
17     Fecha fecha;        ///< Queda registrada la fecha del pedido actualizada con la fecha y hora del sistema.
18     float importe;      ///< Importe total de todos los usuarios.
19     bool tramitado;     ///< Booleano a true si el pedido esta tramitado, false en otro caso.
20     unsigned num;       ///< Número de pedido de biblioteca.
21     lista_sin<PedidoUsuario *> pedido_usu; ///< Registro en la estructura de datos del pedido de un usuario.
22
```

Figura 23: Captura de pantalla de los atributos miembros de la clase PedidoBiblioteca.

El constructor por defecto, parametrizado y por copia de la clase no respetan el orden establecido de los atributos miembros, no cumpliendo de este modo con el estándar:

```
8  /**
9   * @brief Constructor por defecto de la clase PedidoBiblioteca.
10  */
11 PedidoBiblioteca::PedidoBiblioteca() :
12     fecha() {
13     importe = 0;
14     tramitado = false;
15     this->pedido_usu = pedido_usu;
16     num = 0;
17 }
18
```

Figura 24: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

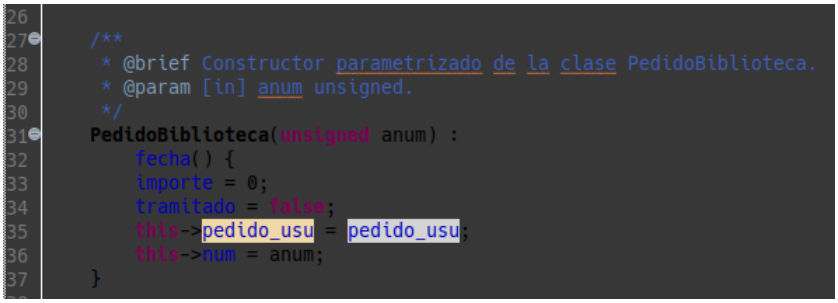


Figura 25: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

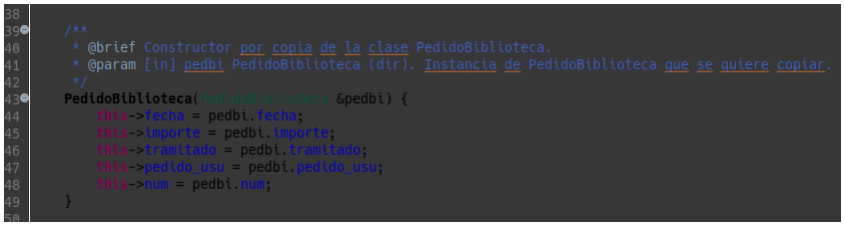


Figura 26: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

Para poder cumplir con el estándar lo único que tenemos que hacer es ordenar la inicialización de los atributos del siguiente modo:

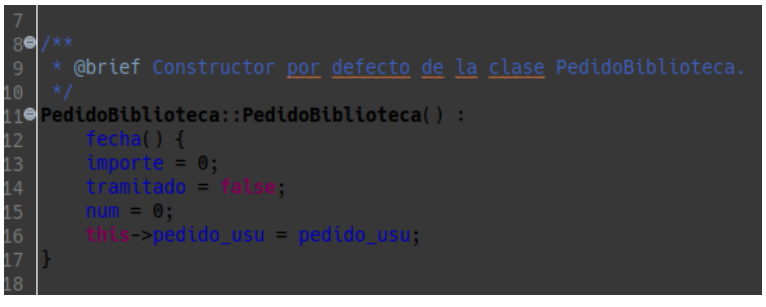


Figura 27: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

```
26
27      /**
28       * @brief Constructor parametrizado de la clase PedidoBiblioteca.
29       * @param [in] anum unsigned.
30       */
31      PedidoBiblioteca(unsigned anum) :
32          fecha() {
33          importe = 0;
34          tramitado = false;
35          this->num = anum;
36          this->pedido_usu = pedido_usu;
37      }
38
```

Figura 28: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

```
38
39      /**
40       * @brief Constructor por copia de la clase PedidoBiblioteca.
41       * @param [in] pedbi PedidoBiblioteca (dir). Instancia de PedidoBiblioteca que se quiere copiar.
42       */
43      PedidoBiblioteca(PedidoBiblioteca &pedbi) {
44          this->fecha = pedbi.fecha;
45          this->importe = pedbi.importe;
46          this->tramitado = pedbi.tramitado;
47          this->num = pedbi.num;
48          this->pedido_usu = pedbi.pedido_usu;
49      }
50
```

Figura 29: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

Para la clase clase PedidoUsuario, tenemos la siguiente lista de miembros:

```
13
14  /**
15   * @brief Clase que representa el pedido que un usuario hace a la biblioteca.
16   */
17  class PedidoUsuario {
18      Fecha fecha;       ///< Fecha que queda registrada al hacer un pedido.
19      int prioridad;     ///< Prioridad concedida al pedido del usuario.
20      float precio;      ///< Precio del pedido que ha hecho el usuario.
21      bool tramitado; ///< Booleano que nos va a indicar si el pedido ha sido tramitado o no.
22      Usuario *usuario; ///< Puntero que referencia a un usuario en concreto.
23      Libro *libro;      ///< Puntero que referencia a un libro en concreto.
24
```

Figura 30: Captura de pantalla de los atributos miembros de la clase PedidoUsuario.

Para los constructores por defecto y parametrizado podemos observar que no se cumple el estándar, como se puede observar en las siguientes capturas de pantalla:

```
/**
 * @brief Constructor por defecto del pedido de un usuario en concreto.
 * @return La inicialización de un pedido por parte del usuario con su fecha, precio, etc.
 */
PedidoUsuario::PedidoUsuario() :
    fecha() {              ///< Fecha que queda registrada al hacer un pedido.
    prioridad = 0;         ///< Prioridad concedida al pedido del usuario.
    precio = 0;            ///< Precio del pedido inicializado a cero.
    tramitado = false;     ///< De entrada el pedido aun no ha sido tramitado.
    libro = NULL;          ///< De entrada no se esta apuntando a ningun libro en concreto.
    usuario = NULL;        ///< De entrada no se esta apuntando a ningun usuario en concreto.
}
```

Figura 31: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

```
/**
 * @brief Constructor parametrizado de la clase PedidoUsuario.
 * @param [in] libro Libro (ref).
 * @param [in] usuario Usuario (ref).
 * @param [in] aFecha Fecha.
 * @param [in] aPrioridad int.
 * @param [in] aPrecio float.
 * @param [in] aTramitado bool.
 */
PedidoUsuario::PedidoUsuario(Libro *libro, Usuario *usuario, Fecha aFecha, int aPrioridad, float aPrecio, bool aTramitado) {
    fecha = aFecha;            ///< Copia de la fecha que queda registrada al hacer un pedido.
    prioridad = aPrioridad;    ///< Copia de la prioridad que queda registrada al hacer un pedido.
    precio = aPrecio;          ///< Copia del precio de un pedido.
    tramitado = aTramitado;    ///< Copia de la tramitación de un pedido.
    this->usuario = usuario;   ///< Referencia al usuario mediante el objeto this.
    this->libro = libro;       ///< Referencia al libro mediante el objeto this.
}
```

Figura 32: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

Para hacer que se cumpla con el estándar, debemos inicializar los atributos miembros según el orden establecido en la declaración de la clase, como se muestra en las siguientes capturas de pantalla:

```
/**
 * @brief Constructor por defecto del pedido de un usuario en concreto.
 * @return La inicialización de un pedido por parte del usuario con su fecha, precio, etc.
 */
PedidoUsuario::PedidoUsuario() :
    fecha() {              ///< Fecha que queda registrada al hacer un pedido.
    prioridad = 0;         ///< Prioridad concedida al pedido del usuario.
    precio = 0;            ///< Precio del pedido inicializado a cero.
    tramitado = false;     ///< De entrada el pedido aun no ha sido tramitado.
    usuario = NULL;        ///< De entrada no se esta apuntando a ningun usuario en concreto.
    libro = NULL;          ///< De entrada no se esta apuntando a ningun libro en concreto.
}
```

Figura 33: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

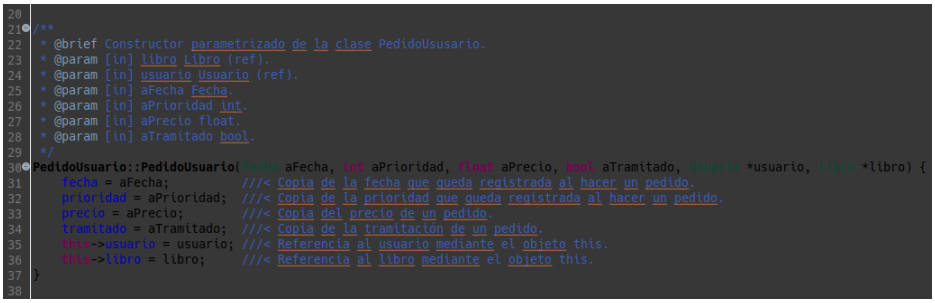


Figura 34: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

Para la clase clase Usuario, tenemos la siguiente lista de miembros:

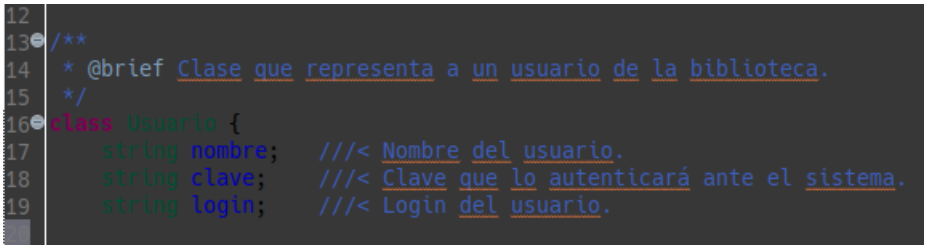


Figura 35: Captura de pantalla de los atributos miembros de la clase Usuario.

En el caso de esta clase, solo se dispone de un constructor por defecto. En este, no se respeta el estándar ya que no se inicializan los atributos miembro en el orden canónico, como se puede observar en la siguiente captura de pantalla:

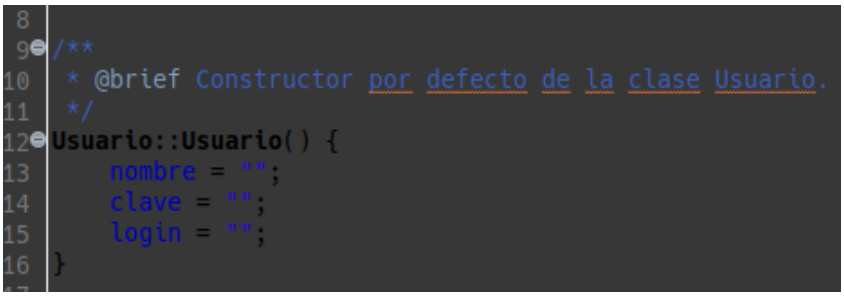


Figura 36: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

3.4. MSC52-CPP. Las funciones que devuelven un valor deben devolver un valor desde todas las rutas de salida.

3.4.1. Enlace al sitio web del estándar seleccionado

`https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC52-CPP.+Value-returning+functions+must+return+a+value+from+all+exit+paths`

3.4.2. Explicación sobre su utilidad

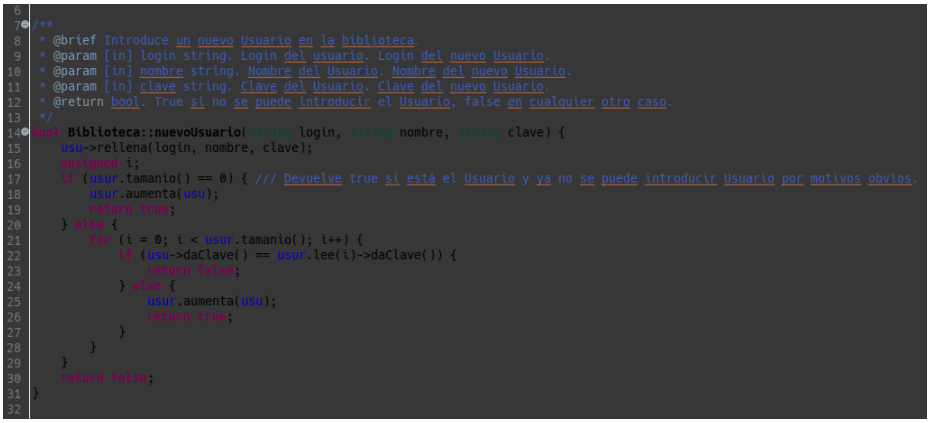
La utilidad de la aplicación de este estándar es bastante conocida. Este estándar nos dice que, siempre que una función devuelve un valor, cada ruta de ejecución de la misma debe de devolver siempre un valor. En el caso de que en algún caso no se devolviera nada, esto podría desembocar en un comportamiento indefinido de la aplicación.

3.4.3. Aplicación del estándar al proyecto

En nuestro proyecto, todas las funciones que devuelven un valor cumplen con este estándar. Dentro de las funciones que devuelven un valor, en nuestro proyecto hay tres tipos diferenciados: las funciones que solo tienen una ruta de ejecución, las funciones que tienen más de una ruta de ejecución y devuelven un valor en cada ruta, y las funciones que tienen más de una ruta de ejecución pero no devuelven un valor en todas.

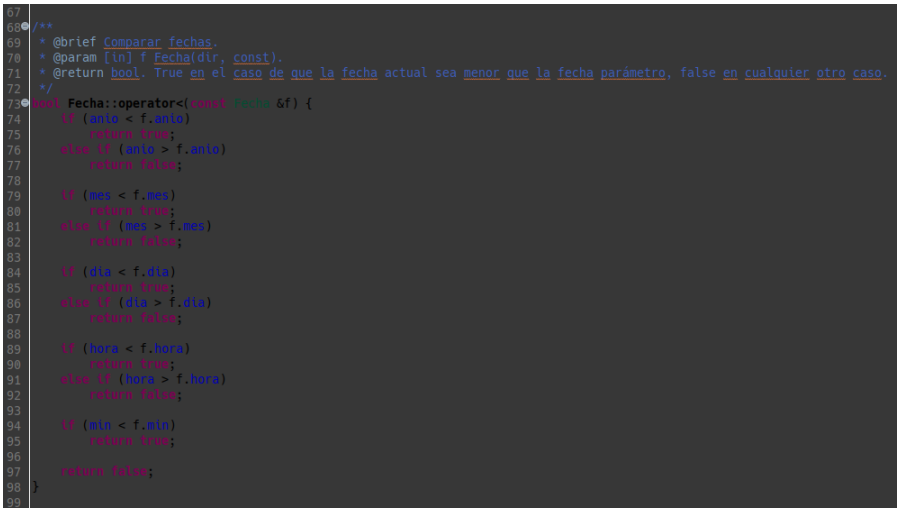
Las funciones que solo tienen una ruta de ejecución las obviaremos en este informe, ya que el cumplimiento de este estándar resulta algo trivial de comprobar.

Para el segundo tipo de funciones pondremos un par de ejemplos a continuación en los que se puede comprobar claramente que no existe ninguna ruta de ejecución en la que no se devuelva ningún valor.



```
/**
 * @brief Introduce un nuevo Usuario en la biblioteca.
 * @param [in] login string. Login del usuario. Login del nuevo Usuario.
 * @param [in] nombre string. Nombre del Usuario. Nombre del nuevo Usuario.
 * @param [in] clave string. Clave del Usuario. Clave del nuevo Usuario.
 * @return bool. True si no se puede introducir el Usuario, false en cualquier otro caso.
 */
bool Biblioteca::nuevoUsuario(string login, string nombre, string clave) {
    usu->rellena(login, nombre, clave);
    unsigned i;
    if (usur.tamanio() == 0) { /// Devuelve true si está el Usuario y ya no se puede introducir Usuario por motivos obvios.
        usur.aumenta(usu);
        return true;
    } else {
        for (i = 0; i < usur.tamanio(); i++) {
            if (usu->daClave() == usur.lee(i)->daClave()) {
                return false;
            } else {
                usur.aumenta(usu);
                return true;
            }
        }
    }
    return false;
}
```

Figura 37: Captura de pantalla del cumplimiento del estándar MSC52-CPP.

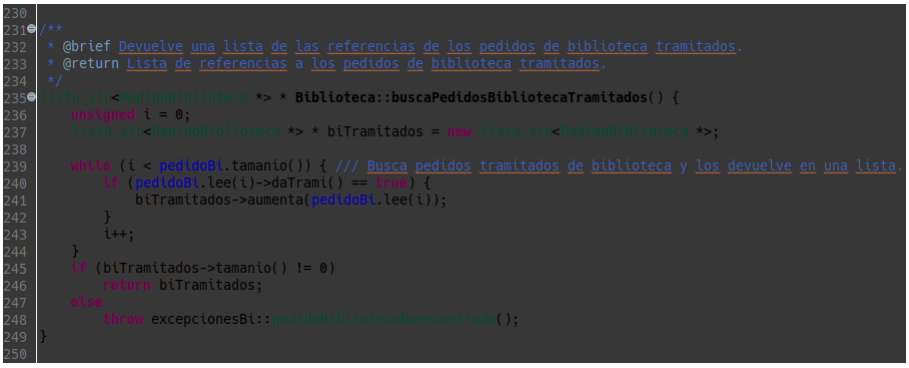


```
/**
 * @brief Comparar fechas.
 * @param [in] f Fecha(dir, const).
 * @return bool. True en el caso de que la fecha actual sea menor que la fecha parámetro, false en cualquier otro caso.
 */
bool Fecha::operator<(const Fecha &f) {
    if (anio < f.anio)
        return true;
    else if (anio > f.anio)
        return false;

    if (mes < f.mes)
        return true;
    else if (mes > f.mes)
        return false;

    if (dia < f.dia)
        return true;
    else if (dia > f.dia)
        return false;

    if (hora < f.hora)
        return true;
    else if (hora > f.hora)
        return false;

    if (min < f.min)
        return true;

    return false;
}
```

Figura 38: Captura de pantalla del cumplimiento del estándar MSC52-CPP.

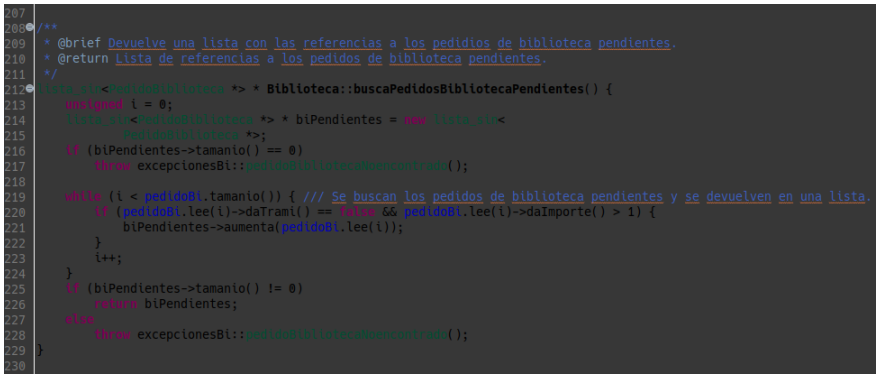
Para el tercer tipo de funciones, si bien no se devuelve un valor en alguna de las rutas de ejecución, esto es debido a que la devolución de un valor no válido produciría un error de ejecución. Por este motivo, no se devuelve un valor sino que se ejecuta una excepción que detiene la ejecución para evitar comportamientos impredecibles.

Teniendo en cuenta lo anterior, en el resto de rutas de ejecución que no causan excepciones sí se devuelve siempre un valor. Por este motivo, se considera cumplido el estándar. A continuación, dejamos un par de capturas de funciones de este tipo:



```
230
231 /**
232  * @brief Devuelve una lista de las referencias de los pedidos de biblioteca tramitados.
233  * @return Lista de referencias a los pedidos de biblioteca tramitados.
234  */
235 lista_sin<PedidoBiblioteca *> * Biblioteca::buscaPedidosBibliotecaTramitados() {
236     unsigned i = 0;
237     lista_sin<PedidoBiblioteca *> * biTramitados = new lista_sin<PedidoBiblioteca *>;
238
239     while (i < pedidoBi.tamanio()) { /// Busca pedidos tramitados de biblioteca y los devuelve en una lista.
240         if (pedidoBi.lee(i)->daTrami() == true) {
241             biTramitados->aumenta(pedidoBi.lee(i));
242         }
243         i++;
244     }
245     if (biTramitados->tamanio() != 0)
246         return biTramitados;
247     else
248         throw excepcionesBi::pedidoBibliotecaNoencontrado();
249 }
250
```

Figura 39: Captura de pantalla del cumplimiento del estándar MSC52-CPP.



```
207
208 /**
209  * @brief Devuelve una lista con las referencias a los pedidos de biblioteca pendientes.
210  * @return Lista de referencias a los pedidos de biblioteca pendientes.
211  */
212 lista_sin<PedidoBiblioteca *> * Biblioteca::buscaPedidosBibliotecaPendientes() {
213     unsigned i = 0;
214     lista_sin<PedidoBiblioteca *> * biPendientes = new lista_sin<
215             PedidoBiblioteca *>;
216     if (biPendientes->tamanio() == 0)
217         throw excepcionesBi::pedidoBibliotecaNoencontrado();
218
219     while (i < pedidoBi.tamanio()) { /// Se buscan los pedidos de biblioteca pendientes y se devuelven en una lista.
220         if (pedidoBi.lee(i)->daTrami() == false && pedidoBi.lee(i)->daImporte() > 1) {
221             biPendientes->aumenta(pedidoBi.lee(i));
222         }
223         i++;
224     }
225     if (biPendientes->tamanio() != 0)
226         return biPendientes;
227     else
228         throw excepcionesBi::pedidoBibliotecaNoencontrado();
229 }
230
```

Figura 40: Captura de pantalla del cumplimiento del estándar MSC52-CPP.

3.5. FIO51-CPP. Cerrar los archivos cuando ya no sean necesarios.

3.5.1. Enlace al sitio web del estándar seleccionado

`https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed`

3.5.2. Explicación sobre su utilidad

Una llamada a la función $std :: basic\_filebuf < T >:: open()$ siempre debe ir acompañada de otra llamada a la función $std :: basic_filebuf < T >:: close()$ antes de la finalización del ciclo de vida del último puntero que almacenase el valor devuelto por la llamada de la primera función ó antes de la finalización del programa, lo que ocurriese antes.

La mala praxis de este estándar puede provocar la utilización innecesaria de memoria estática durante toda la ejecución del programa. En el peor de los casos

si se abrieran muchos archivos y no se cerrara ninguno durante la ejecución del programa, podría llegar a provocar un desbordamiento de la memoria estática. Aún utilizando memoria dinámica, si no cerramos el archivo cuando ya no sea necesario, seguiríamos desperdiciando memoria igualmente.

3.5.3. Aplicación del estándar al proyecto

En el proyecto elegido para la realización de las prácticas solo se hace lectura de un fichero en una única función en todo el programa, esta función es "$voidBiblioteca :: cargaLibros(string fichero)$".

```
58 /**
59  * @brief Devuelve una lista con los libros que contengan el titulo que se le pasa como parametro.
60  * @param [in] fichero string. Fichero donde se encuentra almacenada la información de los libros.
61  */
62 void Biblioteca::cargaLibros(string fichero) {
63     ifstream entrada;
64     entrada.open(fichero.c_str(), ios::in);
65     string aTitulo;
66     string aAutores;
67     string aEditorial;
68     string aISBN;
69     string aAnio;
70     string aPrecioActual;
71     string espacio;
72     if (entrada) {
73         while (!entrada.eof()) {
74             getline(entrada, aTitulo);
75             getline(entrada, aAutores);
76             getline(entrada, aAnio); /// Convierte a entero un string.
77             getline(entrada, aEditorial);
78             getline(entrada, aISBN);
79             getline(entrada, aPrecioActual); /// Convierte a entero un string.
80             getline(entrada, espacio);
81             Libro *lib = new Libro(aTitulo, aAutores, aEditorial, aISBN,
82                     atoi(aAnio.c_str()), (float) atof(aPrecioActual.c_str()));
83             libro.aumenta(lib);
84         }
85     } else {
86         throw excepcionesBi::errorApertura();
87     }
88     entrada.close();
89 }
```

Figura 41: Captura de pantalla del cumplimiento del estándar FIO51-CPP.

En la captura de pantalla anterior observamos que en la línea número 64 se abre el archivo "fichero", cuando ya se han realizado todas las operaciones de lectura del contenido del mismo, se procede a su cierre en la línea 88, quedando verificado el cumplimiento del estándar.

Referencias

[1] https://www.sei.cmu.edu/about/divisions/cert/index.cfm

[2] https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards

[3] https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard

[4] https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682

[5] https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL51-CPP.+Do+not+declare+or+define+a+reserved+identifier

[6] https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL52-CPP.+Never+qualify+a+reference+type+with+const+or+volatile

[7] https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP53-CPP.+Write+constructor+member+initializers+in+the+canonical+order

[8] https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC52-CPP.+Value-returning+functions+must+return+a+value+from+all+exit+paths

[9] https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed