



## PRÁCTICA 4

*Aplicación de Métricas Software*

Autores:

Andrés Rojas Ortega 77382127F  
Esteban Jódar Pozo 26003112W

## Índice

<b>1. Introducción</b>	<b>5</b>
<b>2. Selección de los estándares a seguir</b>	<b>6</b>
<b>3. Aplicación de estándares sobre el proyecto</b>	<b>6</b>
3.1. DCL51-CPP. No declare ni defina un identificador reservado . . . . .	6
3.1.1. Enlace al sitio web del estándar seleccionado . . . . .	6
3.1.2. Explicación sobre su utilidad . . . . .	7
3.1.3. Aplicación del estándar al proyecto . . . . .	8
3.2. DCL52-CPP. Nunca califique un tipo de referencia con constante o volátil . . . . .	9
3.2.1. Enlace al sitio web del estándar seleccionado . . . . .	9
3.2.2. Explicación sobre su utilidad . . . . .	9
3.2.3. Aplicación del estándar al proyecto . . . . .	9
3.3. DCL59-CPP. No defina un espacio de nombres sin nombre en un archivo de encabezado. . . . .	10
3.3.1. Enlace al sitio web del estándar seleccionado . . . . .	10
3.3.2. Explicación sobre su utilidad . . . . .	10
3.3.3. Aplicación del estándar al proyecto . . . . .	11
3.4. MEM51-CPP. Desasignar correctamente la memoria asignada a los objetos dinámicamente. . . . .	11
3.4.1. Enlace al sitio web del estándar seleccionado . . . . .	11
3.4.2. Explicación sobre su utilidad . . . . .	12
3.4.3. Aplicación del estándar al proyecto . . . . .	12
3.5. MEM52-CPP. Detectar errores de asignación de memoria . . . . .	13
3.5.1. Enlace al sitio web del estándar seleccionado . . . . .	13
3.5.2. Explicación sobre su utilidad . . . . .	13
3.5.3. Aplicación del estándar al proyecto . . . . .	13
3.6. FIO50-CPP. No ingrese y elabore alternativamente desde un flujo sin una llamada de posicionamiento. . . . .	14
3.6.1. Enlace al sitio web del estándar seleccionado . . . . .	14
3.6.2. Explicación sobre su utilidad . . . . .	14
3.6.3. Aplicación del estándar al proyecto . . . . .	15
3.7. FIO51-CPP. Cerrar los archivos cuando ya no sean necesarios.	15

3.7.1. Enlace al sitio web del estándar seleccionado . . . . .	15
3.7.2. Explicación sobre su utilidad . . . . .	15
3.7.3. Aplicación del estándar al proyecto . . . . .	16
3.8. ERR51-CPP. Manejar todas las excepciones. . . . .	17
3.8.1. Enlace al sitio web del estándar seleccionado . . . . .	17
3.8.2. Explicación sobre su utilidad . . . . .	17
3.8.3. Aplicación del estándar al proyecto . . . . .	17
3.9. OOP53-CPP. Escribir los inicializadores de miembros de los constructores en el orden canónico. . . . .	19
3.9.1. Enlace al sitio web del estándar seleccionado . . . . .	19
3.9.2. Explicación sobre su utilidad . . . . .	19
3.9.3. Aplicación del estándar al proyecto . . . . .	20
3.10. MSC52-CPP. Las funciones que devuelven un valor deben devolver un valor desde todas las rutas de salida. . . . .	30
3.10.1. Enlace al sitio web del estándar seleccionado . . . . .	30
3.10.2. Explicación sobre su utilidad . . . . .	30
3.10.3. Aplicación del estándar al proyecto . . . . .	30
<b>4. Análisis estático automatizado de código</b>	<b>33</b>
4.1. Error nº1: Memory leak . . . . .	33
4.1.1. Origen/explicación del error detectado . . . . .	33
4.1.2. Corrección realizada para solventar el error . . . . .	35
4.2. Error nº2: Mismatching allocation and deallocation . . . . .	36
4.2.1. Origen/explicación del error detectado . . . . .	36
4.2.2. Corrección realizada para solventar el error . . . . .	37
4.3. Error nº3: Class has a constructor with 1 argument that is not explicit . . . . .	38
4.3.1. Origen/explicación del error detectado . . . . .	38
4.3.2. Corrección realizada para solventar el error . . . . .	39
4.4. Error nº4: Parameter can be declared with const . . . . .	39
4.4.1. Origen/explicación del error detectado . . . . .	39
4.4.2. Corrección realizada para solventar el error . . . . .	41
4.5. Error nº5: The function is never used . . . . .	42
4.5.1. Origen/explicación del error detectado . . . . .	42
4.5.2. Corrección realizada para solventar el error . . . . .	43
4.6. Error nº6: The scope of the variable can be reduced . . . . .	43
4.6.1. Origen/explicación del error detectado . . . . .	43
4.6.2. Corrección realizada para solventar el error . . . . .	47
4.7. Error nº7: Unused varriable . . . . .	52
4.7.1. Origen/explicación del error detectado . . . . .	52
4.7.2. Corrección realizada para solventar el error . . . . .	53

4.8. Error nº8: Variable is reassigned a value before the old one has been used . . . . .	53
4.8.1. Origen/explicación del error detectado . . . . .	53
4.8.2. Corrección realizada para solventar el error . . . . .	54
4.9. Error nº9: Either the condition is redundant or there is possible null pointer dereference . . . . .	55
4.9.1. Origen/explicación del error detectado . . . . .	55
4.9.2. Corrección realizada para solventar el error . . . . .	55
4.10. Error nº10: Possible leak in public function. The pointer is not deallocated before is allocated . . . . .	56
4.10.1. Origen/explicación del error detectado . . . . .	56
4.10.2. Corrección realizada para solventar el error . . . . .	56
4.11. Warning Nº 1 y 2 . . . . .	57
4.11.1. Origen/explicación del warning detectado: . . . . .	57
4.11.2. Corrección realizada para solventar el warning: . . . . .	58
4.12. Warning Nº 5 . . . . .	58
4.12.1. Origen/explicación del warning detectado: . . . . .	58
4.12.2. Corrección realizada para solventar el warning: . . . . .	59
4.13. Warning Nº 6 . . . . .	59
4.13.1. Origen/explicación del warning detectado: . . . . .	59
4.13.2. Corrección realizada para solventar el warning: . . . . .	59
4.14. Warning Nº 7 . . . . .	60
4.14.1. Origen/explicación del warning detectado: . . . . .	60
4.14.2. Corrección realizada para solventar el warning: . . . . .	60
4.15. Warning Nº 13 . . . . .	61
4.15.1. Origen/explicación del warning detectado: . . . . .	61
4.15.2. Corrección realizada para solventar el warning: . . . . .	61
4.16. INFO N.º 1 . . . . .	61
4.16.1. Origen/explicación del info detectado: . . . . .	61
4.16.2. Corrección realizada para solventar el info: . . . . .	61
4.17. INFO N.º 2, 3, 4 y 5 . . . . .	61
4.17.1. Origen/explicación del info detectado: . . . . .	62
4.17.2. Corrección realizada para solventar el info: . . . . .	62
4.18. INFO N.º 6, 7 y 8 . . . . .	62
4.18.1. Origen/explicación del info detectado: . . . . .	63
4.18.2. Corrección realizada para solventar el info: . . . . .	63
4.19. INFO N.º 9 . . . . .	63
4.19.1. Origen/explicación del info detectado: . . . . .	63
4.19.2. Corrección realizada para solventar el info: . . . . .	64
4.20. INFO N.º 10 y 11 . . . . .	64
4.20.1. Origen/explicación del info detectado: . . . . .	64

4.20.2. Corrección realizada para solventar el info: . . . . .	64
4.21. INFO N. <sup>º</sup> 12 y N. <sup>º</sup> 13 . . . . .	65
4.21.1. Origen/explicación de los infos detectados: . . . . .	65
4.21.2. Corrección realizada para solventar los infos: . . . . .	65
4.22. INFO N. <sup>º</sup> 14 . . . . .	65
4.22.1. Origen/explicación de los infos detectados: . . . . .	66
4.22.2. Corrección realizada para solventar el info: . . . . .	66
4.23. INFO N. <sup>º</sup> 15 . . . . .	66
4.23.1. Origen/explicación de los infos detectados: . . . . .	66
4.23.2. Corrección realizada para solventar el info: . . . . .	66
4.24. INFOS N. <sup>º</sup> 16 y 17 . . . . .	67
4.24.1. Origen/explicación de los infos detectados: . . . . .	67
4.24.2. Corrección realizada para solventar los dos infos: . . . . .	67
4.25. INFOS N. <sup>º</sup> 18, 19, 20 y 21 . . . . .	67
4.25.1. Origen/explicación de los infos detectados: . . . . .	68
4.25.2. Corrección realizada para solventar los cuatro infos: . . . . .	68
4.26. INFOS N. <sup>º</sup> 22, 23 y 24 . . . . .	68
4.26.1. Origen/explicación de los infos detectados: . . . . .	69
4.26.2. Corrección realizada para solventar los tres infos: . . . . .	69
<b>5. Reducción de la complejidad ciclomática</b>	<b>69</b>
5.1. Estado del proyecto antes de las correcciones . . . . .	69
5.2. Corrección del módulo ” <i>void Application :: aplicacion_admin()</i> ”	70
5.2.1. Estado del módulo antes de la corrección . . . . .	70
5.2.2. Estado del módulo después de la corrección . . . . .	74
5.3. Corrección del módulo ” <i>void Application :: aplicacion_usuario()</i> ”	79
5.3.1. Estado del módulo antes de la corrección . . . . .	79
5.3.2. Estado del módulo después de la corrección . . . . .	83
5.4. Corrección del módulo ” <i>void Fecha :: comprobarFecha(unsigned aDia, unsigned aMes, unsigned aAnio, unsigned aHora, unsigned aMin) const</i> ”	85
5.4.1. Estado del módulo antes de la corrección . . . . .	85
5.4.2. Estado del módulo después de la corrección . . . . .	86
5.5. Estado del proyecto después de las correcciones . . . . .	86

## 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 desarrolló para una de las prácticas de la asignatura Estructuras de datos, impartida en el segundo curso del grado de Ingeniería informática ofrecido 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.
- **DCL59-CPP.** No defina un espacio de nombres sin nombre de encabezado.
- **MEM51-CPP.** Desasignar correctamente la memoria asignada a los objetos dinámicamente.
- **MEM52-CPP.** Detectar errores de asignación de memoria.
- **FIO50-CPP.** No ingrese y elabore alternativamente desde un flujo sin una llamada de posicionamiento interviniente.
- **FIO51-CPP.** Cerrar los archivos cuando ya no sean necesarios.
- **ERR51-CPP.** Manejar todas las excepciones.
- **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.

## 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:

```

10 #include <iostream>
11 using namespace std;
12
13 template<class T>
14
15 /**
16 * @brief Plantilla genérica para un nodo de las estructuras de datos que soportara esta aplicación.
17 */
18 struct nodo {
19     T date; //< Dato parametrizado. Puede ser un usuario, un libro, una fecha, un ISBN...etc.
20     nodo<> * sigue; //< Nodo siguiente en la estructura parametrizado a cualquier tipo de los anteriores.
21 };
22

```

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 carácter reservado '&' se puede cometer el error de escribir la expresión de la siguiente forma: "*char const& p*", C++ ignora o prohíbe la asignación de referencia a la palabra reservada *const*". 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:

```

68  /**
69   * @brief Comparar fechas.
70   * @param [in] f Fecha(dtr, const).
71   * @return bool. True en el caso de que la fecha actual sea menor que la fecha parámetro, false en cualquier otro caso.
72 */
73 bool Fecha::operator<(const Fecha &f) {
74     if (anio < f.anio)
75         return true;
76     else if (anio > f.anio)
77         return false;
78
79     if (mes < f.mes)
80         return true;
81     else if (mes > f.mes)
82         return false;
83
84     if (dia < f.dia)
85         return true;
86     else if (dia > f.dia)
87         return false;
88
89     if (hora < f.hora)
90         return true;
91     else if (hora > f.hora)
92         return false;
93
94     if (min < f.min)
95         return true;
96
97     return false;
98 }

```

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

```

221 /**
222  * @brief Función auxiliar de conversión desde estructura de tiempo tm de time.h.
223  * @param [out] t tm (const, dir).
224 */
225 void Fecha::leerTiempo(const tm &t) {
226     dia = t.tm_mday;
227     mes = t.tm_mon + 1;
228     anio = t.tm_year + 1900;
229     hora = t.tm_hour;
230     min = t.tm_min;
231 }
232

```

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

### 3.3. DCL59-CPP. No defina un espacio de nombres sin nombre en un archivo de encabezado.

#### 3.3.1. Enlace al sitio web del estándar seleccionado

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL59-CPP.+Do+not+define+an+unnamed+namespace+in+a+header+file>

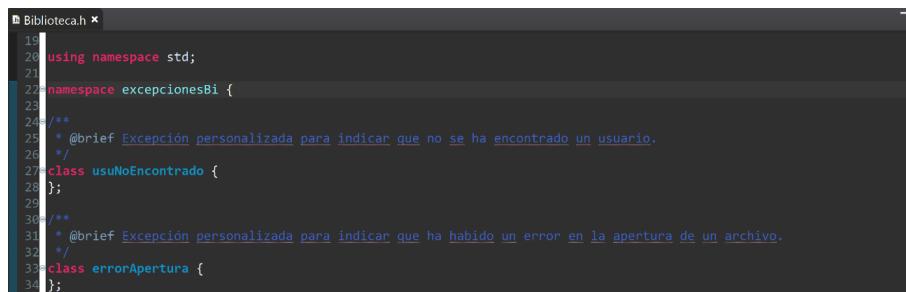
#### 3.3.2. Explicación sobre su utilidad

El código C++ de calidad de producción utiliza con frecuencia archivos de encabezado como medio para compartir código entre unidades de traducción. Un archivo de encabezado es cualquier archivo que se inserta en una unidad de traducción a través de una directiva. No defina un espacio de nombres sin nombre en un archivo de encabezado. Cuando se define un espacio de nombres sin nombre en un archivo de encabezado, puede dar lugar a resultados sorprendentes. Debido al vínculo interno predeterminado, cada unidad de

traducción definirá su propia instancia única de los miembros del espacio de nombres sin nombre que se usan en ODR dentro de esa unidad de traducción. Esto puede causar resultados inesperados, hinchar el ejecutable resultante o desencadenar inadvertidamente el comportamiento indefinido debido a infracciones de una regla de una definición.

### 3.3.3. Aplicación del estándar al proyecto

Los espacios de nombres sin nombre se usan para definir un espacio de nombres que es único para la unidad de traducción, donde los nombres contenidos tienen vinculación interna de forma predeterminada.



```
19
20 using namespace std;
21
22 namespace excepcionesBi {
23
24 /**
25 * @brief Excepción personalizada para indicar que no se ha encontrado un usuario.
26 */
27 class usuNoEncontrado {
28 };
29
30 /**
31 * @brief Excepción personalizada para indicar que ha habido un error en la apertura de un archivo.
32 */
33 class errorApertura {
34 };
35 }
```

Figura 5: Captura de pantalla del cumplimiento del estándar DCL59-CPP.

Aquí observamos como en un archivo cabecera al incluir un nuevo espacio de nombres, se le ha tenido que poner un nombre distinto para no confundir a la unidad de traducción. En este caso el nombre ofrecido ha sido el de “excepcionesBi”.



```
    }
} throw excepcionesBi::usuNoEncontrado(); }
```

Figura 6: Captura de pantalla del cumplimiento del estándar DCL59-CPP.

Así puede ser perfectamente referenciado desde el .cpp de dicho archivo Biblioteca como podemos ver en la imagen:

## 3.4. MEM51-CPP. Desasignar correctamente la memoria asignada a los objetos dinámicamente.

### 3.4.1. Enlace al sitio web del estándar seleccionado

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM51-CPP.+Properly+deallocate+dynamically+allocated+resources>

### 3.4.2. Explicación sobre su utilidad

Desasignar un puntero que no ha sido asignado con new () es un comportamiento indefinido porque el valor del puntero no se obtuvo mediante una función de asignación.

Al mismo tiempo, desasignar un puntero que ya se ha pasado a una función de desasignación es un comportamiento indefinido porque el valor del puntero ya no apunta a la memoria que se ha asignado dinámicamente. Puede estar apuntando a cualquier parte.

Cuando new () es invocado, el resultado es una llamada a un operador sobrecargable con el mismo nombre. Esta función se puede llamar directamente, pero tiene las mismas restricciones que su contraria dijéramos.

Es decir, delete () y pasarle un parámetro de puntero tiene las mismas restricciones que llamar al operador delete () en ese puntero. Además, las sobrecargas están sujetas a la resolución del alcance, por lo que es posible (pero no permitido) llamar a un operador específico de clase para asignar un objeto, pero a un operador global para desasignar el objeto. En resumen, y como siempre se nos ha enseñado, tras un new () debe de haber un delete (), con lo cual, aparte de evitar comportamientos inesperados, haremos una buena gestión de la memoria, recurso finito que hay que tratar con eficiencia.

### 3.4.3. Aplicación del estándar al proyecto

Aquí vemos como se llama al operador new ():

```
/** @brief Constructor por defecto.
 */
Application::Application() {
    bi = Biblioteca();
    usu = new Usuario();
    lusu = new lista_sin<Usuario>();
    li = Libro();
    pedbi = new PedidoBiblioteca();
    pedbi = new lista_sin<PedidoBiblioteca *>;
    pedusu = new lista_sin<PedidoUsuario *>;
    libro = new lista_sin<Libro *>;
    pedbipunt = new PedidoBiblioteca;
}
```

Figura 7: Captura de pantalla del cumplimiento del estándar MEM51-CPP.

Para desasignar correctamente la memoria asignada dinámicamente a estos objetos observamos como en el destructor se hacen las llamadas correspondientes al operador delete () que es la contraparte de new () y además a la

función de limpieza, que internamente hace también una llamada al operador delete () .

```
 /**
 * @brief Destructor de la clase Application.
 */
virtual ~Application() {
    delete usu;
    pedbi->limpia();
    pedusu->limpia();
}
```

Figura 8: Captura de pantalla del cumplimiento del estándar DCL59-CPP.

### 3.5. MEM52-CPP. Detectar errores de asignación de memoria

#### 3.5.1. Enlace al sitio web del estándar seleccionado

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors>

#### 3.5.2. Explicación sobre su utilidad

Como sabemos de cursos anteriores, el operador de asignación de memoria dinámica new (), produce una excepción si se produce un error en la asignación. Muchas veces nos hemos preguntado si el resultado de la llamada era nullptr o puntero nulo haciendo comprobaciones.

Pues bien, no es necesario hacer dichas comprobaciones debido a la excepción que se produce.

Además, también nos evita que el valor devuelto no lo sea antes de tener acceso al puntero o de tener a este apuntando a lo que conocemos como “basura” .

Esta excepción es la (std::bad\_alloc), que detecta si no se puede asignar suficiente memoria.

#### 3.5.3. Aplicación del estándar al proyecto

Por ejemplo, si no se ha podido asignar memoria para usu con el operador new (), aparte de las excepciones personalizadas que nos mostrarían un mensaje, es correcto que en el primer bloque catch se atrapara también esta excepción si acaso fallase la asignación.

```

57     try {
58         usu = bi.buscaUsuario(alogin, aclave);
59         pedusu = bi.buscaPedidosUsuarioPendientes(usu);
60         while (i < pedusu->tamanio()) {
61             cout << *(pedusu->lee(i)) << endl;
62             i++;
63         }
64     } catch (const excepcionesBi::usuNoEncontrado& e) {
65         cerr << e.what();

```

Figura 9: Captura de pantalla del incumplimiento del estándar MEM52-CPP.

Una vez arreglado, se muestra ya en los catch la “bad\_alloc” si falla la asignación de memoria:

```

57     try {
58         usu = bi.buscaUsuario(alogin, aclave);
59         pedusu = bi.buscaPedidosUsuarioPendientes(usu);
60         while (i < pedusu->tamanio()) {
61             cout << *(pedusu->lee(i)) << endl;
62             i++;
63         }
64     } catch (bad_alloc& e) {
65     } catch (const excepcionesBi::usuNoEncontrado& e) {
66         cerr << e.what();
67     } catch (excepcionesBi::pedidoUsuarioNoencontrado& e) {
68         cerr << e.what();
69     }
70 }

```

Figura 10: Captura de pantalla del cumplimiento del estándar MEM52-CPP.

### 3.6. FIO50-CPP. No ingrese y elabore alternativamente desde un flujo sin una llamada de posicionamiento.

#### 3.6.1. Enlace al sitio web del estándar seleccionado

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/FI050-CPP.+Do+not+alternately+input+and+output+from+a+file+stream+without+an+intervening+positioning+call>

#### 3.6.2. Explicación sobre su utilidad

Cuando se abre un archivo, entre la secuencia de entrada al mismo y la de cierre, ha de situarse una función o llamada para un posicionamiento correcto dentro del mismo.

Si no nos situamos correctamente dentro de él podemos dar lugar a algunos comportamientos no deseados como:

- El puntero de lectura/escritura puede escribir datos al final de un archivo y a continuación leer del mismo. Si no hay una llamada para un posicionamiento intermedio, el comportamiento es indefinido.
- El puntero de lectura/escritura puede comenzar leer datos al comienzo de un archivo y a continuación escribir en el mismo.

De ahí que sea necesario una función intermedia, que posicione correctamente el puntero de lectura/escritura, situada entre la entrada al fichero y el cierre del mismo para evitar comportamientos indeseados. En este caso ha sido la función `std::basic_istream<T>::seekg()`.

### 3.6.3. Aplicación del estándar al proyecto

```

59  * @brief Devuelve una lista con los libros que contengan el título 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         entrada.seekg(0, ios::beg);
74         while (!entrada.eof()) {
75             getline(entrada, aTitulo);
76             getline(entrada, aAutores);
77             getline(entrada, aAnio); // Convierte a entero un string.
78             getline(entrada, aEditorial);
79             getline(entrada, aISBN);
80             getline(entrada, aPrecioActual); // Convierte a entero un string.
81             getline(entrada, espacio);
82             Libro *lib = new Libro(aTitulo, aAutores, aEditorial, aISBN,
83                                   atol(aAnio.c_str()), (float) atof(aPrecioActual.c_str()));
84             libro.aumenta(lib);
85         }
86     } else {
87         throw excepcionesBi::errorApertura();
88     }
89     entrada.close();
90 }
```

Figura 11: Captura de pantalla del cumplimiento del estándar FIO50-CPP.

En esta solución compatible, la `std::basic_istream<T>::seekg()` función se llama en la línea 73, entre la salida y la entrada, posicionando el puntero al comienzo de la lectura, eliminando el comportamiento indefinido .

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

### 3.7.1. Enlace al sitio web del estándar seleccionado

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/FI051-CPP.+Close+files+when+they+are+no+longer+needed>

### 3.7.2. 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 desperdiando memoria igualmente.

### 3.7.3. 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 título que se le pasa como parámetro.
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                                    atof(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 12: 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.

### 3.8. ERR51-CPP. Manejar todas las excepciones.

#### 3.8.1. Enlace al sitio web del estándar seleccionado

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions>

#### 3.8.2. Explicación sobre su utilidad

Cuando se produce una excepción, el control se transfiere al controlador más cercano con un tipo que coincide con el tipo de la excepción producida. Si no se encuentra ningún controlador coincidente directamente dentro de los controladores para un bloque try en el que se produce la excepción, la búsqueda de un controlador coincidente continúa buscando dinámicamente controladores en los bloques de prueba circundantes del mismo subproceso.

Todas las excepciones producidas por una aplicación deben ser detectadas por un controlador de excepciones coincidente. Incluso si la excepción no se puede recuperar correctamente, el uso del controlador de excepciones coincidente garantiza que la pila se desenrollará correctamente y proporciona la oportunidad de administrar correctamente los recursos externos antes de finalizar el proceso.

#### 3.8.3. Aplicación del estándar al proyecto

Aquí podemos ver como se lanzan excepciones, pero no son tratadas correctamente:

```
*Biblioteca.h  Biblioteca.cpp *
52     return usur.lee(i);
53 }
54 }
55 throw excepcionesBi::usuNoEncontrado();
56 }
```

Figura 13: Captura de pantalla del incumplimiento del estándar ERR51-CPP.

```
81     getline(entrada, espacio);
82     Libro *lib = new Libro(aTitulo, aAutores, aEditorial, aISBN,
83                           atoi(aAnio.c_str()), (float) atof(aPrecioActual.c_str()));
84     libro.aumenta(lib);
85   }
86 } else {
87   throw excepcionesBi::errorApertura();
88 }
```

Figura 14: Captura de pantalla del incumplimiento del estándar ERR51-CPP.

Ya que las clases no derivan de la clase exception ni tienen nada definido para ellas:

```

8 *Biblioteca.h *
9 #include "Libro.h"
10 #include "PedidoBiblioteca.h"
11 #include "PedidoUsuario.h"
12 #include "Usuario.h"
13 #include "Lista_sin.h"
14 #include <iostream>
15 #include <stdio.h>
16 #include <stdlib.h>
17 #include <string>
18 #include <iomanip>
19
20 using namespace std;
21
22 namespace excepcionesBi {
23
24 /**
25  * @brief Excepción personalizada para indicar que no se ha encontrado un usuario.
26 */
27 class usuNoEncontrado {
28 };
29
30 /**
31  * @brief Excepción personalizada para indicar que ha habido un error en la apertura de un archivo.
32 */
33 class errorApertura {
34 };

```

Figura 15: Captura de pantalla del incumplimiento del estándar ERR51-CPP.

Una vez le hemos dado definición a dichas excepciones y con la inclusión de la directiva `#include <exception>` en el fichero de cabecera, ya podemos controlar las excepciones, lo que garantiza que la pila se desenrolla hasta la función y permite una gestión elegante de los recursos externos.

```

#include <exception>
using namespace std;
namespace excepcionesBi {
    class usuNoEncontrado : public exception {
        public:
            const char* what() const throw () {
                return "\nError: El usuario no se pudo encontrar.";
            }
    };
    class errorApertura : public exception {
        public:
            const char* what() const throw () {
                return "\nError: El Fichero no se pudo abrir";
            }
    };
    class libroNoencontrado : public exception {
        public:
            const char* what() const throw () {
                return "\nError: El libro no se pudo encontrar.";
            }
    };
}

```

Figura 16: Captura de pantalla del cumplimiento del estándar ERR51-CPP.

Se atrapan las excepciones lanzadas por los throw en los bloques catch pero no se tratan adecuadamente.

```

57     try {
58         usu = bi.buscaUsuario(alogin, aclave);
59         pedusu = bi.buscaPedidosUsuarioPendientes(usu);
60         while (i < pedusu->tamanio()) {
61             cout << *(pedusu->lee(i)) << endl;
62             i++;
63         }
64     } catch (excepcionesBi::usuNoEncontrado&) {
65         cout << " Usuario no encontrado. " << endl;
66     } catch (excepcionesBi::pedidoUsuarioNoencontrado&) {
67         cout << " El usuario no tiene pedidos pendientes. " << endl;
68     }
69 }
```

Figura 17: Captura de pantalla del incumplimiento del estándar ERR51-CPP.

Y ya aquí se atrapan en el catch correspondiente y se tratan adecuadamente.

```

59     try {
60         usu = bi.buscaUsuario(alogin, aclave);
61         pedusu = bi.buscaPedidosUsuarioPendientes(usu);
62         while (i < pedusu->tamanio()) {
63             cout << *(pedusu->lee(i)) << endl;
64             i++;
65         }
66     } catch (const excepcionesBi::usuNoEncontrado& e) {
67         cerr << e.what();
68     } catch (excepcionesBi::pedidoUsuarioNoencontrado& e) {
69         cerr << e.what();
70     }
71 }
```

Figura 18: Captura de pantalla del cumplimiento del estándar ERR51-CPP.

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

#### 3.9.1. 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>

#### 3.9.2. 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.

### 3.9.3. Aplicación del estándar al proyecto

En nuestro proyecto se encuentran las siguientes clases:

- Application
- Biblioteca
- Fecha
- Libro
- lista\_sin
- PedidoBiblioteca
- PedidoUsuario
- Usuario

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

```

13  */
14 * @brief Clase principal la cual derivará en sus variantes de admin y de usuario.
15 */
16 class Application {
17     Biblioteca bi;           ///< Objeto de la clase Biblioteca.
18     Usuario *usu;           ///< Referencia a un usuario en concreto.
19     lista_siu<Usuario> lusu;  ///< Lista de los usuarios registrados en la biblioteca.
20     Libro li;               ///< Objeto de la clase libro.
21     PedidoBiblioteca *pedbi;  ///< Referencia a un pedido específico hecho por la biblioteca
22     lista_siu<PedidoBiblioteca *> * pedbi;  ///< Lista de todos los libros pedidos en cada solicitud.
23     lista_siu<PedidoUsuario *> * pedusu;  ///< Lista de los pedidos hechos por los usuarios.
24     lista_siu<Libro *> * libro;   ///< Lista de todos los libros pedidos por todos los usuarios.
25     PedidoBiblioteca *pedbipunt;  ///< Lista de los pedidos de la biblioteca no tramitados.

```

Figura 19: Captura de pantalla de los atributos miembros de la clase Application.

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.

```

9  /**
10  * @brief Constructor por defecto.
11 */
12 Application::Application() {
13     pedusu = new lista_sin<PedidoUsuario *>;
14     pedbi = new lista_sin<PedidoBiblioteca *>;
15     libro = new lista_sin<Libro *>;
16     usu = new Usuario();
17     pedbipunt = new PedidoBiblioteca;
18     pedbipunt = NULL;
19     pedBi = new PedidoBiblioteca;
20
21 }
22

```

Figura 20: 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:

```

8 /**
9  * @brief Constructor por defecto.
10 */
11
12 Application::Application() {
13     bi = Biblioteca();
14     usu = new Usuario();
15     lusu = new lista_sin<Usuario>();
16     li = Libro();
17     pedBi = new PedidoBiblioteca;
18     pedbi = new lista_sin<PedidoBiblioteca *>;
19     pedusu = new lista_sin<PedidoUsuario *>;
20     libro = new lista_sin<Libro *>;
21     pedbipunt = new PedidoBiblioteca;
22 }
23

```

Figura 21: Captura de pantalla del cumplimiento del estándar OOP53-CPP.

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

```

55 /**
56 * @brief Clase que representa la información y el funcionamiento de una biblioteca.
57 */
58 class Biblioteca {
59
60     lista_sin<Usuario *> usu; ///Lista donde se almacenan todos los usuarios.
61     lista_sin<PedidoUsuario *> pedido_usu; ///Lista donde se almacenan todos los pedidos de los usuarios.
62     lista_sin<PedidoBiblioteca *> pedidobi; ///Lista donde se almacenan todos los pedidos hechos por la biblioteca.
63     lista_sin<Libro *> libro; ///Lista donde se almacenan todos los libros que posee la biblioteca.
64     Usuario *usu; ///Puntero al último usuario introducido en la biblioteca.
65

```

Figura 22: 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.

```

67
68  /**
69   * @brief Constructor por defecto de la clase Biblioteca.
70   */
71  Biblioteca() :
72      usur(), pedido_usu(), pedidoBi(), libro() {
73      usu = new Usuario;
74  }
75

```

Figura 23: Captura de pantalla del cumplimiento del estándar OOP53-CPP.

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

```

11
12 /**
13  * @brief Clase sencilla para representar fechas y horas.
14 */
15 class Fecha {
16     unsigned dia; ///Información de día.
17     unsigned mes; ///Información de mes.
18     unsigned anio; ///Información de año.
19     unsigned hora; ///Información de hora.
20     unsigned min; ///Información de minutos.
21     static const unsigned diasMes[12]; ///Almacena los días por mes.
22

```

Figura 24: 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.

```

13 /**
14  * @brief Constructor por defecto de la clase Fecha.
15  * Crea una fecha con la hora actual.
16 */
17 Fecha::Fecha() {
18     time t tiempoActual;
19     struct tm *fechaActual;
20     time(&tiempoActual); ///Obtiene la hora actual del sistema.
21     fechaActual = localtime(&tiempoActual); ///Decodifica la hora en campos separados.
22     leerTiempo(*fechaActual);
23 }
24

```

Figura 25: 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 26: 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  * @brief Constructor parametrizado de la clase Fecha.
31  * Crea una fecha concreta. Devuelve una excepción ErrorFechaIncorrecta si la fecha introducida no es correcta.
32  */
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 27: 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
34 Fecha(const Fecha &f) :
35     dia(f.dia), mes(f.mes), anio(f.anio), hora(f.hora), min(f.min) {
36

```

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

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

```

13 /**
14  * @brief Clase que representa toda la información necesaria de un libro.
15  */
16 class Libro {
17     string titulo;    ///< Título del libro.
18     string autores;  ///< Autor/es del libro.
19     string editorial; ///< Editorial que publica el libro.
20     string ISBN;     ///< Código ISBN identificativo del libro (International Standard Book Number).
21     int anio;        ///< Año en el que se publica el libro.
22     float precioActual; ///< Precio actual del libro.
23

```

Figura 29: 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:

```

8  /**
9  * @brief Constructor por defecto de la clase Libro.
10 * @pre Al estar inicializando un objeto de la clase, todos los atributos aparecen vacíos o inicializados a cero.
11 */
12 Libro::Libro() {
13     titulo = "";
14     autores = "";
15     editorial = "";
16     ISBN = "";
17     precioActual = 0;
18     anio = 0;
19 }
20

```

Figura 30: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

Esto se soluciona reordenando la inicialización en el orden correcto:

```

8 /**
9  * @brief Constructor por defecto de la clase Libro.
10 * @pre Al estar inicializando un objeto de la clase, todos los atributos aparecen vacíos o inicializados a cero.
11 */
12 Libro::Libro() {
13     titulo = "";
14     autores = "";
15     editorial = "";
16     ISBN = "";
17     anio = 0;
18     precioActual = 0;
19 }
20

```

Figura 31: 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:

```

22 /**
23 * @brief Constructor parametrizado de la clase Libro.
24 * @param [in] aTitulo string. Título del libro.
25 * @param [in] aAutores string. Autor/es del libro.
26 * @param [in] aEditorial string. Editorial del libro.
27 * @param [in] aISBN string. ISBN del libro.
28 * @param [in] aAnio int. Año de publicación del libro.
29 * @param [in] aPrecioActual float. Precio del libro.
30 */
31 Libro::Libro(string aTitulo, string aAutores, string aEditorial, string aISBN, int aAnio, float aPrecioActual) {
32     titulo = aTitulo;
33     autores = aAutores;
34     editorial = aEditorial;
35     ISBN = aISBN;
36     anio = aAnio;
37     precioActual = aPrecioActual;
38 }
39

```

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

```

29
30  /**
31  * @brief Constructor por copia de la clase Libro.
32  * @param [in] lib Libro(dir). Instancia de la clase Libro de la cual se va realizar una copia.
33  */
34  Libro(const Libro &lib) {
35      this->titulo = lib.titulo;
36      this->autores = lib.autores;
37      this->editorial = lib.editorial;
38      this->ISBN = lib.ISBN;
39      this->anio = lib.anio;
40      this->precioActual = lib.precioActual;
41  }
42

```

Figura 33: 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:

```

22
23  template<class T>
24  /**
25  * @brief Plantilla genérica de estructura de datos Lista enlazada.
26  */
27  class lista_sin {
28      nodo<T> *nuevo;    ///Nodo nuevo en la Lista enlazada.
29      nodo<T> *primero;  ///Primer nodo en la Lista enlazada.
30      nodo<T> *ultimo;  ///Último nodo en la Lista enlazada.
31      unsigned numElem;  ///Número de nodos de la Lista enlazada.
32

```

Figura 34: 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 35: 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.tamano(); i++) {
125          this->aumenta(list.lee(i));
126      }
127  }
128

```

Figura 36: 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 * @brief Clase que representa de manera genérica un pedido hecho por la biblioteca.
14 */
15 class PedidoBiblioteca {
16     Fecha fecha;      //;< Queda registrada la fecha del pedido actualizada con la fecha y hora del sistema.
17     float importe;    //;< Importe total de todos los usuarios.
18     bool tramitado;  //;< Booleano a true si el pedido está tramitado, false en otro caso.
19     unsigned num;    //;< Número de pedido de biblioteca.
20     lista_sus<pedidoUsuario> pedido_usu; //;< Registro en la estructura de datos del pedido de un usuario.
21
22

```

Figura 37: 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 38: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

```

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

```

Figura 39: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

```

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

```

Figura 40: 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:

```

7  /**
8  * @brief Constructor por defecto de la clase PedidoBiblioteca.
9  */
10 PedidoBiblioteca::PedidoBiblioteca() :
11     fecha() {
12     importe = 0;
13     tramitado = false;
14     num = 0;
15     this->pedido_usu = pedido_usu;
16 }
17
18

```

Figura 41: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

```

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

```

Figura 42: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

```

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

```

Figura 43: 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 * @brief Clase que representa el pedido que un usuario hace a la biblioteca.
15 */
16 class PedidoUsuario {
17     Fecha fecha; //< Fecha que queda registrada al hacer un pedido.
18     int prioridad; //< Prioridad concedida al pedido del usuario.
19     float precio; //< Precio del pedido que ha hecho el usuario.
20     bool tramitado; //< Booleano que nos va a indicar si el pedido ha sido tramitado o no.
21     Usuario *usuario; //< Puntero que referencia a un usuario en concreto.
22     Libro *libro; //< Puntero que referencia a un libro en concreto.
23 }
24

```

Figura 44: 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:

```

8  /**
9  * @brief Constructor por defecto del pedido de un usuario en concreto.
10 * @return La inicialización de un pedido por parte del usuario con su fecha, precio, etc.
11 */
12 PedidoUsuario::PedidoUsuario() :
13     fecha()           ///<> Fecha que queda registrada al hacer un pedido.
14     prioridad = 0;    ///<> Prioridad concedida al pedido del usuario.
15     precio = 0;       ///<> Precio del pedido inicializado a cero.
16     tramitado = false; ///<> De entrada el pedido aun no ha sido tramitado.
17     libro = NULL;    ///<> De entrada no se esta apuntando a ningun libro en concreto.
18     usuario = NULL;  ///<> De entrada no se esta apuntando a ningun usuario en concreto.
19 }
20

```

Figura 45: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

```

20 /**
21 * @brief Constructor parametrizado de la clase PedidoUsuario.
22 * @param [in] libro Libro (ref).
23 * @param [in] usuario Usuario (ref).
24 * @param [in] aFecha Fecha.
25 * @param [in] aPrioridad int.
26 * @param [in] aPrecio float.
27 * @param [in] aTramitado bool.
28 */
29
30 PedidoUsuario::PedidoUsuario(Libro *libro, Usuario *usuario, Fecha aFecha, int aPrioridad, float aPrecio, bool aTramitado) {
31     fecha = aFecha;      ///<> Copia de la fecha que queda registrada al hacer un pedido.
32     prioridad = aPrioridad; ///<> Copia de la prioridad que queda registrada al hacer un pedido.
33     precio = aPrecio;    ///<> Copia del precio de un pedido.
34     tramitado = aTramitado; ///<> Copia de la tramitación de un pedido.
35     this->usuario = usuario; ///<> Referencia al usuario mediante el objeto this.
36     this->libro = libro;   ///<> Referencia al libro mediante el objeto this.
37 }
38

```

Figura 46: 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:

```

7
8 /**
9  * @brief Constructor por defecto del pedido de un usuario en concreto.
10 * @return La inicialización de un pedido por parte del usuario con su fecha, precio, etc.
11 */
12 PedidoUsuario::PedidoUsuario() :
13     fecha()           ///<> Fecha que queda registrada al hacer un pedido.
14     prioridad = 0;    ///<> Prioridad concedida al pedido del usuario.
15     precio = 0;       ///<> Precio del pedido inicializado a cero.
16     tramitado = false; ///<> De entrada el pedido aun no ha sido tramitado.
17     usuario = NULL;  ///<> De entrada no se esta apuntando a ningun usuario en concreto.
18     libro = NULL;    ///<> De entrada no se esta apuntando a ningun libro en concreto.
19 }
20

```

Figura 47: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

```

20 /**
21 * @brief Constructor parametrizado de la clase PedidoUsuario.
22 * @param [in] libro Libro (ref).
23 * @param [in] usuario Usuario (ref).
24 * @param [in] aFecha Fecha.
25 * @param [in] aPrioridad int.
26 * @param [in] aPrecio float.
27 * @param [in] aTramitado bool.
28 */
29
30 PedidoUsuario::PedidoUsuario(Fecha aFecha, int aPrioridad, float aPrecio, bool aTramitado, Usuario *usuario, Libro *libro) {
31     fecha = aFecha; //Copia de la fecha que queda registrada al hacer un pedido.
32     prioridad = aPrioridad; //Copia de la prioridad que queda registrada al hacer un pedido.
33     precio = aPrecio; //Copia del precio de un pedido.
34     tramitado = aTramitado; //Copia de la tramitación de un pedido.
35     this->usuario = usuario; //Referencia al usuario mediante el objeto this.
36     this->libro = libro; //Referencia al libro mediante el objeto this.
37 }
38

```

Figura 48: Captura de pantalla en la que se muestra el cumplimiento del estándar OOP53-CPP.

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

```

12 /**
13 * @brief Clase que representa a un usuario de la biblioteca.
14 */
15
16 class Usuario {
17     string nombre; //Nombre del usuario.
18     string clave; //Clave que lo autenticará ante el sistema.
19     string login; //Login del usuario.
20

```

Figura 49: 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:

```

8 /**
9 * @brief Constructor por defecto de la clase Usuario.
10 */
11
12 Usuario::Usuario() {
13     nombre = "";
14     clave = "";
15     login = "";
16 }
17

```

Figura 50: Captura de pantalla en la que se muestra el incumplimiento del estándar OOP53-CPP.

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

#### 3.10.1. 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>

#### 3.10.2. 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.

#### 3.10.3. 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.

```

6 /**
7 * @brief Introduce un nuevo Usuario en la biblioteca.
8 * @param [in] login string. Login del usuario. Login del nuevo Usuario.
9 * @param [in] nombre string. Nombre del Usuario, Nombre del nuevo Usuario.
10 * @param [in] clave string. Clave del Usuario. Clave del nuevo Usuario.
11 * @return bool. True si no se puede introducir el Usuario, falso en cualquier otro caso.
12 */
13
14 bool Biblioteca::nuevoUsuario(string login, string nombre, string clave) {
15     usu->rellena(login, nombre, clave);
16     unsigned i;
17     if (usu.tamano() == 0) { // Devuelve true si está el Usuario y ya no se puede introducir Usuario por motivos obvios.
18         usu.aumenta(usu);
19         return true;
20     } else {
21         for (i = 0; i < usu.tamano(); i++) {
22             if (usu->daClave(i) == usu.lee(i)->daClave()) {
23                 return false;
24             } else {
25                 usu.aumenta(usu);
26                 return true;
27             }
28         }
29     }
30     return false;
31 }
32

```

Figura 51: Captura de pantalla del cumplimiento del estándar MSC52-CPP.

```

67 /**
68 * @brief Comparar fechas.
69 * @param [in] f Fecha(dia, const).
70 * @return bool. True en el caso de que la fecha actual sea menor que la fecha parámetro, falso en cualquier otro caso.
71 */
72
73 bool Fecha::operator<(const Fecha &f) {
74     if (anio < f.anio)
75         return true;
76     else if (anio > f.anio)
77         return false;
78
79     if (mes < f.mes)
80         return true;
81     else if (mes > f.mes)
82         return false;
83
84     if (dia < f.dia)
85         return true;
86     else if (dia > f.dia)
87         return false;
88
89     if (hora < f.hora)
90         return true;
91     else if (hora > f.hora)
92         return false;
93
94     if (min < f.min)
95         return true;
96
97     return false;
98 }
99

```

Figura 52: 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 * @brief Devuelve una lista de las referencias de los pedidos de biblioteca tramitados.
232 * @return Lista de referencias a los pedidos de biblioteca tramitados.
233 */
234
235 list<PedidoBiblioteca *> * Biblioteca::buscaPedidosBibliotecaTramitados() {
236     unsigned i = 0;
237     list<PedidoBiblioteca *> * biTramitados = new list<PedidoBiblioteca *>;
238
239     while (i < pedidoBl.tamano()) { /// Busca pedidos tramitados de biblioteca y los devuelve en una lista.
240         if (pedidoBl.lee(i)->daTramit() == true) {
241             biTramitados->aumenta(pedidoBl.lee(i));
242         }
243         i++;
244     }
245     if (biTramitados->tamano() != 0)
246         return biTramitados;
247     else
248         throw excepcionesBi::pedidoBibliotecaNoencontrado();
249 }
250

```

Figura 53: Captura de pantalla del cumplimiento del estándar MSC52-CPP.

```

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

```

Figura 54: Captura de pantalla del cumplimiento del estándar MSC52-CPP.

## 4. Análisis estático automatizado de código

### 4.1. Error nº1: Memory leak

#### 4.1.1. Origen/explícacion del error detectado

En las siguientes dos capturas de pantalla se muestra el código fuente de las funciones *modifica* y *elimina\_dato* de la clase plantilla *lista\_sin* de nuestra aplicación.

```
174@template<class T>
175 void lista_sin<T>::modifica(T elem, unsigned pos) {
176     unsigned i = 0;
177     nodo<T> *iter;
178     iter = new struct nodo<T>;
179     iter = primero;
180     if (iter == NULL) {
181         throw ErrorElemento();
182     } else {
183         while (iter && i < pos) {
184             i++;
185             iter = iter->sige;
186         }
187         iter->date = elem;
188     }
189 }
```

Figura 55: Captura de pantalla de la función modifica

```

213
214@ template<class T>
215 T lista_sin<T>::elimina_dato(unsigned pos) {
216     unsigned i = 1;
217     T var;
218     nodo<T> *viejo;
219     viejo = new struct nodo<T>;
220     viejo = primero;
221     nuevo = primero;
222     if (numElem == 1) {
223         viejo = primero;
224         var = viejo->date;
225         delete viejo;
226         numElem--;
227         primero = NULL;
228         ultimo = NULL;
229         return var;
230     } else {
231         while (nuevo && i < pos) {
232             i++;
233             nuevo = nuevo->sige;
234         }
235         viejo = nuevo->sige;
236         nuevo->sige = viejo->sige;
237         var = viejo->date;
238         delete viejo;
239         numElem--;
240         return var;
241     }
242 }
243

```

Figura 56: Captura de pantalla de la función `elimina_dato`

En la línea 179 `cppcheck` nos indica que ha detectado un error de código, en concreto, se trata de un error de perdida de memoria para la variable `iter`. Esto es debido a que en la línea 177 se declara la variable (la cual es un puntero a una estructura llamada `nodo`, que hemos definido nosotros anteriormente en el mismo archivo), y en la línea 178 se inicializa con la palabra reservada `new`. Al realizar esto anterior, hemos reservado memoria dinámica para la estructura y la hemos inicializado.

Figura 57: Detalle del mensaje de error de `cppcheck`

Figura 58: Detalle del mensaje de error de `cppcheck`

El error se origina cuando, en la línea 179, hacemos que la variable `iter` (que es un puntero) apunte al atributo de la clase `primero` sin liberar la memoria de la estructura con la que inicializamos anteriormente. Esto hace que la memoria que ocupa dicha estructura no se libere y que, por lo tanto, se produzca una pérdida de memoria.

En el caso de la función *elimina\_dao*, la explicación del origen del error es exactamente la misma pero aplicada a la variable *viejo*.

#### 4.1.2. Corrección realizada para solventar el error

La manera de resolver esta incidencia es de lo más trivial. Tan solo deberemos realizar la declaración de la variable *y*, acto seguido, asignarle la dirección de memoria del atributo de la clase *primero*. A continuación, dejamos una captura de pantalla del estado de las funciones tras la resolución de los errores:

```
173
174@ template<class T>
175 void lista_sin<T>::modifica(T elem, unsigned pos) {
176     unsigned i = 0;
177     nodo<T> *iter;
178     iter = primero;
179     if (iter == NULL) {
180         throw ErrorElemento();
181     } else {
182         while (iter && i < pos) {
183             i++;
184             iter = iter->sige;
185         }
186         iter->date = elem;
187     }
188 }
```

Figura 59: Captura de pantalla de la función modifica tras la resolución del error

```

213
214@ template<class T>
215 T lista_sin<T>::elimina_dato(unsigned pos) {
216     unsigned i = 1;
217     T var;
218     nodo<T> *viejo;
219     viejo = primero;
220     nuevo = primero;
221     if (numElem == 1) {
222         viejo = primero;
223         var = viejo->date;
224         delete viejo;
225         numElem--;
226         primero = NULL;
227         ultimo = NULL;
228         return var;
229     } else {
230         while (nuevo && i < pos) {
231             i++;
232             nuevo = nuevo->sige;
233         }
234         viejo = nuevo->sige;
235         nuevo->sige = viejo->sige;
236         var = viejo->date;
237         delete viejo;
238         numElem--;
239     }
240 }
241 }
```

Figura 60: Captura de pantalla de la función `elimina_dato`

**nota:** Cabe destacar que también se podría haber resuelto este error realizando un ”`delete iter/viejo`” antes de asignarle la dirección de memoria del atributo `primero` pero en este caso hemos descartado esta opción ya que la estructura con la que se inicializa no se utiliza para nada y supondría la realización de operaciones adicionales innecesarias.

## 4.2. Error nº2: Mismatching allocation and deallocation

### 4.2.1. Origen/explicación del error detectado

A continuación presentamos una captura de pantalla de la función `limpia` de la clase plantilla `lista_sin`.

```

247
248@ template<class T>
249 void lista_sin<T>::limpia() {
250     while (primero) {
251         nuevo = primero;
252         primero = primero->sige;
253         delete[] nuevo;
254     }
255 }
256

```

Figura 61: Captura de pantalla de la función limpia

En esta función, en la línea 253, cppcheck nos indica que hay un error. Dicho error es originado dado que hacemos un ”*delete[]*”, es decir, una liberación de memoria para un vector/array de punteros. En el caso de la variable *nuevo*, podemos comprobar que se trata de un atributo de la clase *lista\_sin* y que se trata de un puntero a una estructura llamada *nodo*. Justo esto anterior es lo que genera el error, es decir, estamos intentando liberar la memoria asignada de un vector/array cuando la variable **no es ninguno de estos**.

Errors (6 items)				
↳ (cppcheck error) Mismatching allocation and deallocation: lista_sin < Libro * >::nuevo	lista_sin.h	/prac3G5	line 253	cppcheck Problem
↳ (cppcheck error) Mismatching allocation and deallocation: lista_sin < PedidoBiblioteca * >::nuevo	lista_sin.h	/prac3G5	line 253	cppcheck Problem
↳ (cppcheck error) Mismatching allocation and deallocation: lista_sin < PedidoUsuario * >::nuevo	lista_sin.h	/prac3G5	line 253	cppcheck Problem
↳ (cppcheck error) Mismatching allocation and deallocation: lista_sin < Usuario * >::nuevo	lista_sin.h	/prac3G5	line 253	cppcheck Problem
↳ (cppcheck error) Mismatching allocation and deallocation: lista_sin < Usuario >::nuevo	lista_sin.h	/prac3G5	line 253	cppcheck Problem
↳ (cppcheck error) Mismatching allocation and deallocation: lista_sin::nuevo	lista_sin.h	/prac3G5	line 253	cppcheck Problem

Figura 62: Detalle del mensaje de error de cppcheck

#### 4.2.2. Corrección realizada para solventar el error

Para corregir este error lo único que deberemos realizar es sustituir la sentencia ”*delete[] nuevo*” por la sentencia ”*delete nuevo*”. Una vez realizado esto el error habrá desaparecido, como se puede comprobar en la siguiente captura de pantalla:

```

247
248@ template<class T>
249 void lista_sin<T>::limpia() {
250     while (primero) {
251         nuevo = primero;
252         primero = primero->sige;
253         delete nuevo;
254     }
255 }
256

```

Figura 63: Captura de pantalla de la función limpia tras la resolución del error

### 4.3. Error nº3: Class has a constructor with 1 argument that is not explicit

#### 4.3.1. Origen/explícacion del error detectado

En la siguiente captura de pantalla se observa el constructor parametrizado de la clase *PedidoBiblioteca*.

```
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 64: Captura de pantalla del constructor parametrizado de la clase PedidoBiblioteca

Como se puede observar en la captura de pantalla, en este constructor se hace uso de una lista de inicializadores de miembro, la cual inicializa el valor de los atributos miembro antes de la ejecución del cuerpo del constructor.

En la lista observamos que se encuentra el atributo miembro *fecha* inicializado con su constructor por defecto. Esto es debido a que entre paréntesis no hemos escrito ningún nombre de variable, ya que no se le pasa ningún parámetro del tipo *fecha* al constructor.

No obstante, el parámetro *anum* no se usa en la lista, sino que se usa posteriormente en el cuerpo del constructor. Aquí es donde se haya el origen del error.

Como bien hemos estudiado en los primeros cursos de la carrera, cuando usamos la lista de inicializadores de miembro, es común asignar un valor por defecto a todos los parámetros del constructor. De este modo, si no se le pasara algún parámetro a este constructor, inicializaría el atributo miembro con el valor por defecto que tenga declarado el parámetro en cuestión.

El parámetro *anum* no tiene ningún valor por defecto y es por este motivo que la herramienta *cppcheck* nos genera el error.

```
(cppcheck style) Class 'PedidoBiblioteca' has a constructor with 1 argument that is not explicit. PedidoBiblioteca.h /prac3G5 line 31 cppcheck Problem
```

Figura 65: Detalle del mensaje de error de cppcheck

#### 4.3.2. Corrección realizada para solventar el error

La forma de corregir el error generado es muy sencilla. Nos limitaremos a asignarle un valor por defecto al parámetro *anum*, en este caso el mismo valor que se le asigna al atributo miembro *num* en el constructor por defecto. De este modo, podremos comprobar que el error habrá desaparecido.

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

Figura 66: Captura de pantalla del constructor parametrizado de la clase PedidoBiblioteca tras la corrección del error

**Nota:** Una segunda forma de corregir el error habría sido declarar el constructor como explícito, es decir, añadir *explicit* a la declaración del constructor. Hemos descartado esta opción ya que en algunos casos esto podría ocasionar un comportamiento indeseado en la ejecución del programa en ciertas situaciones.

### 4.4. Error nº4: Parameter can be declared with const

#### 4.4.1. Origen/explicación del error detectado

Este error tiene su origen en el constructor por copia de la clase *PedidoBiblioteca*. A continuación, se presenta una captura de pantalla de dicho constructor.

```
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 67: Captura de pantalla del constructor por copia de la clase PedidoBiblioteca

Como se puede observar en la captura, cppcheck nos indica que el error se encuentra en la línea 43 del archivo. Nos recomienda que el parámetro *pedbi* puede ser declarado como constante. Este parámetro es la instancia del objeto del cuál se hace una copia

Es bastante común (y recomendable) que en los constructores por copia y en los operadores de asignación se pasen como constante los objetos de los cuales queremos copiar el valor de sus atributos. Esto se realiza para evitar que, bien por descuido o bien por error de programación, se modifiquen los valores de los atributos miembros de estos parámetros.

Es por este motivo que cppcheck nos avisa de que podemos hacer este parámetro constante.

Este error también es detectado para el parámetro *pedbi*, en la línea 70 del archivo *PedidoBiblioteca.h*; y para el parámetro *pedido*, en la línea 49 del archivo *PedidoUsuario.h*. A continuación, presentamos capturas de pantalla de las funciones en las que aparecen estos errores.

Figura 68: Detalle del mensaje de error de cppcheck

Figura 69: Detalle del mensaje de error de cppcheck

```

64
65o  /**
66   * @brief Operador de asignación de la clase PedidoBiblioteca.
67   * @param pedbi [in] PedidoBiblioteca (ref). PedidoBiblioteca de la que se quiere realizar una copia.
68   * @return Instancia copia realizada.
69 */
70o PedidoBiblioteca& operator=(PedidoBiblioteca& pedbi) {
71     this->fecha = pedbi.fecha;
72     this->importe = pedbi.importe;
73     this->tramitado = pedbi.tramitado;
74     this->pedido_usu = pedbi.pedido_usu;
75     return *this;
76 }
77

```

Figura 70: Captura de pantalla del operador de asignación de la clase PedidoBiblioteca

```

43
44⊕ /**
45 * @brief Operador de asignación de la clase PedidoUsuario.
46 * @param [in] pedido PedidoUsuario (ref). Instancia de PedidoUsuario que se quiere copiar
47 * @return Instancia copia creada de la clase PedidoUsuario.
48 */
49⊕ PedidoUsuario& operator=(PedidoUsuario &pedido) {
50     this->fecha = pedido.fecha;
51     this->prioridad = pedido.prioridad;
52     this->precio = pedido.precio;
53     this->tramitado = pedido.tramitado;
54     this->usuario = pedido.usuario;
55     this->libro = pedido.libro;
56     return *this;
57 }
58

```

Figura 71: Captura de pantalla del operador de asignación de la clase PedidoUsuario

#### 4.4.2. Corrección realizada para solventar el error

Para corregir este error tan solo deberemos hacer constante el parámetro *pedbi*. Una vez hecho esto habrá desaparecido el aviso de cppcheck.

```

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(const 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 72: Captura de pantalla del constructor por copia de la clase PedidoBiblioteca tras la corrección del error

```

64
65⊕ /**
66 * @brief Operador de asignación de la clase PedidoBiblioteca.
67 * @param pedbi [in] PedidoBiblioteca (ref). PedidoBiblioteca de la que se quiere realizar una copia.
68 * @return Instancia copia realizada.
69 */
70⊕ PedidoBiblioteca& operator=(const PedidoBiblioteca& pedbi) {
71     this->fecha = pedbi.fecha;
72     this->importe = pedbi.importe;
73     this->tramitado = pedbi.tramitado;
74     this->pedido_usu = pedbi.pedido_usu;
75     return *this;
76 }
77

```

Figura 73: Captura de pantalla del operador de asignación de la clase PedidoBiblioteca tras la resolución del error

**Nota:** El aviso que se muestra en la captura de pantalla se debe a otro origen; el error de este apartado ha sido solucionado.

```

43
440    /**
45     * @brief Operador de asignación de la clase PedidoUsuario.
46     * @param [in] pedido PedidoUsuario (ref). Instancia de PedidoUsuario que se quiere copiar
47     * @return Instancia copia creada de la clase PedidoUsuario.
48     */
490 PedidoUsuario& operator=(const PedidoUsuario &pedido) {
50     this->fecha = pedido.fecha;
51     this->prioridad = pedido.prioridad;
52     this->precio = pedido.precio;
53     this->tramitado = pedido.tramitado;
54     this->usuario = pedido.usuario;
55     this->libro = pedido.libro;
56     return *this;
57 }

```

Figura 74: Captura de pantalla del operador de asignación de la clase PedidoUsuario tras la resolución del error

**Nota:** En el caso de que en nuestro constructor ó función necesitáramos por algún motivo modificar el valor de alguno de los atributos miembro del parámetro y, aún así, nos saltara el aviso de cppcheck deberíamos hacer caso omiso ya que de este modo nos daría un error al compilar el código.

## 4.5. Error nº5: The function is never used

### 4.5.1. Origen/explicación del error detectado

En este caso, cppcheck nos indica que las siguientes funciones no son utilizadas en nuestro código:

- *anadirAnios, anadirDias, anadirHoras, anadirMeses, anadirMin, asignarDia y asignarHora* del archivo *Fecha.cpp*.
- *daISBN* del archivo *Libro.cpp*.
- *daLBiblioteca* del archivo *Biblioteca.cpp*.
- *daLibro* del archivo *PedidoUsuario.cpp*.
- *daPedidoUsuario* del archivo *PedidoBiblioteca.cpp*.
- *validarClave* del archivo *Usuario.cpp*.

En el caso de este error explicar el origen de este error resulta de lo más trivial. Las funciones anteriormente enumerada se han definido en sus respectivos archivos y no se llegan a utilizar en el código del programa.

Por este motivo cppcheck nos alerta de este evento para, en el caso pertinente, eliminar el código fuente de estas funciones y reducir el número de líneas de código de nuestro programa.

↳ (cppcheck style) The function 'anadirAnios' is never used.	Fecha.cpp	/prac3G5	line 166	cppcheck Problem
↳ (cppcheck style) The function 'anadirDias' is never used.	Fecha.cpp	/prac3G5	line 142	cppcheck Problem
↳ (cppcheck style) The function 'anadirHoras' is never used.	Fecha.cpp	/prac3G5	line 130	cppcheck Problem
↳ (cppcheck style) The function 'anadirMeses' is never used.	Fecha.cpp	/prac3G5	line 154	cppcheck Problem
↳ (cppcheck style) The function 'anadirMin' is never used.	Fecha.cpp	/prac3G5	line 118	cppcheck Problem
↳ (cppcheck style) The function 'asignarDia' is never used.	Fecha.cpp	/prac3G5	line 50	cppcheck Problem
↳ (cppcheck style) The function 'asignarHora' is never used.	Fecha.cpp	/prac3G5	line 62	cppcheck Problem
↳ (cppcheck style) The function 'daISBN' is never used.	Libro.cpp	/prac3G5	line 44	cppcheck Problem
↳ (cppcheck style) The function 'daLBiblioteca' is never used.	Biblioteca.cpp	/prac3G5	line 37	cppcheck Problem
↳ (cppcheck style) The function 'daLibro' is never used.	PedidoUsuario.cpp	/prac3G5	line 51	cppcheck Problem
↳ (cppcheck style) The function 'daPedidoUsuario' is never used.	PedidoBiblioteca.cpp	/prac3G5	line 48	cppcheck Problem
↳ (cppcheck style) The function 'validarClave' is never used.	Usuario.cpp	/prac3G5	line 52	cppcheck Problem

Figura 75: Detalle del mensaje de error de cppcheck

#### 4.5.2. Corrección realizada para solventar el error

En nuestro caso particular resolvemos este error indicándole a cppcheck que ignore estos errores. El motivo de tomar tal decisión es debido a que, si bien en el programa no se utilizan estas funciones, en un futuro podrían ser de gran utilidad debido a la naturaleza de las mismas (modificar, obtener y validar atributos de las clases).

Si bien es cierto que se trata de un programa de granja realizado en el contexto de una práctica de asignatura de la carrera, hemos aplicado un punto de vista simulando que la aplicación se encuentra en producción, que está mantenida y que tendrá futuras actualizaciones de funcionalidades.

Una vez ignorados los errores no volverán a ser listados por cppcheck.

**Nota:** No hemos incluido capturas de pantalla ya que las funciones no presentan ningún error como tal y solo se mostrarían los avisos de cppcheck, haciendo aumentar considerablemente la extensión del presente documento.

### 4.6. Error nº6: The scope of the variable can be reduced

#### 4.6.1. Origen/explicación del error detectado

Para este error, cppcheck nos lista 11 incidencias, las cuales mostramos a continuación a modo de captura de pantalla.

(cppcheck style) The scope of the variable 'cpb' can be reduced.	Aplication.cpp	/prac3G5	line 30	cppcheck Problem
(cppcheck style) The scope of the variable 'i' can be reduced.	Aplication.cpp	/prac3G5	line 30	cppcheck Problem
(cppcheck style) The scope of the variable 'i' can be reduced.	Biblioteca.cpp	/prac3G5	line 16	cppcheck Problem
(cppcheck style) The scope of the variable 'i' can be reduced.	lista_sin.h	/prac3G5	line 161	cppcheck Problem
(cppcheck style) The scope of the variable 'i' can be reduced.	lista_sin.h	/prac3G5	line 176	cppcheck Problem
(cppcheck style) The scope of the variable 'i' can be reduced.	lista_sin.h	/prac3G5	line 194	cppcheck Problem
(cppcheck style) The scope of the variable 'i' can be reduced.	lista_sin.h	/prac3G5	line 216	cppcheck Problem
(cppcheck style) The scope of the variable 'num' can be reduced.	Aplication.cpp	/prac3G5	line 30	cppcheck Problem
(cppcheck style) The scope of the variable 'num_ped_bt' can be reduced.	Aplication.cpp	/prac3G5	line 30	cppcheck Problem
(cppcheck style) The scope of the variable 'opcion' can be reduced.	Aplication.cpp	/prac3G5	line 29	cppcheck Problem
(cppcheck style) The scope of the variable 'pos' can be reduced.	Biblioteca.cpp	/prac3G5	line 99	cppcheck Problem

Figura 76: Detalle del mensaje de error de cppcheck

Todos los errores que se generan tienen su origen en la declaración al inicio de la función de las variables para, más adelante, ser inicializadas cuando vayan a utilizarse.

Es recomendable que se declaren e inicialicen las variables locales de una función justo en el fragmento de código en el que vayan a ser utilizados. Por ejemplo, en vez de definir la variable iteradora 'i' de un bucle *for* al inicio de la función y asignarle el valor inicial en el bucle, se debe inicializar y asignar el valor en el mismo bucle.

En todas las capturas de pantalla siguientes se puede observar cómo cppcheck nos indica que cometemos este error en todas las variables resaltadas.

```

27 void Aplication::aplicacion_admin() {
28
29     int opcion;
30     unsigned i = 0, num, nume_ped_bt, cpb = 0;
31     string aclave, alogin, contra = "hola", contrase;
32     cout << " Introduzca contraseña: " << endl;
33     cin >> contrase;
34     if (contrase.compare(contra) == 0) {
35

```

Figura 77: Captura de pantalla del error cppcheck en el código fuente

```

6
78 /**
8 * @brief Introduce un nuevo Usuario en la biblioteca.
9 * @param [in] login string. Login del usuario. Login del nuevo Usuario.
10 * @param [in] nombre string. Nombre del Usuario. Nombre del nuevo Usuario.
11 * @param [in] clave string. Clave del Usuario. Clave del nuevo Usuario.
12 * @return bool. True si no se puede introducir el Usuario, false en cualquier otro caso.
13 */
149 bool Biblioteca::nuevoUsuario(string login, string nombre, string clave) {
15     usu->rellenaLogin(login, nombre, clave);
16     unsigned i;
17     if (usu->tamano() == 0) { // Devuelve true si está el Usuario y ya no se puede introducir
18         usur.aumenta(usu);
19         return true;
20     } else {
21         for (i = 0; i < usur->tamano(); i++) {
22             if (usu->daClave(i) == usur->lee(i)->daClave()) {
23                 return false;
24             } else {
25                 usur.aumenta(usu);
26                 return true;
27             }
28         }
29     }
30     return false;
31 }
32

```

Figura 78: Captura de pantalla del error cppcheck en el código fuente

```

173
17410 template<class T>
175 void lista_sln<T>::modifica(T elem, unsigned pos) {
176     unsigned i = 0;
177     nodo<T> *iter;
178     iter = primero;
179     if (iter == NULL) {
180         throw ErrorElemento();
181     } else {
182         while (iter && i < pos) {
183             i++;
184             iter = iter->sige;
185         }
186         iter->date = elem;
187     }
188 }
189

```

Figura 79: Captura de pantalla del error cppcheck en el código fuente

```

158
15911 template<class T>
160 T lista_sln<T>::lee(unsigned pos) {
161     unsigned i = 0;
162     nuevo = primero;
163     if (nuevo == NULL) {
164         throw ErrorElemento();
165     } else {
166         while (nuevo && i < pos) {
167             i++;
168             nuevo = nuevo->sige;
169         }
170     }
171 }
172

```

Figura 80: Captura de pantalla del error cppcheck en el código fuente

```

189
190# template<class T>
191 void lista_sin<T>::inserta_dato(const T &dato, unsigned pos) {
192     nodo<T> *iter;
193     iter = new struct nodo<T>;
194     unsigned i = 1;
195     if (ultimo == NULL) {
196         iter->sige = NULL;
197         iter->date = dato;
198         primero = ultimo = iter;
199     } else {
200         nuevo = primero;
201         while ((i < pos) && nuevo) {
202             nuevo = nuevo->sige;
203             i++;
204         }
205         if (nuevo != NULL) {
206             iter->sige = nuevo->sige;
207             iter->date = dato;
208             nuevo->sige = iter;
209         }
210     numElem++;
211 }
212 }
```

Figura 81: Captura de pantalla del error cppcheck en el código fuente

```

213
214# template<class T>
215 T lista_sin<T>::elimina_dato(unsigned pos) {
216     unsigned i = 1;
217     T var;
218     nodo<T> *viejo;
219     viejo = primero;
220     nuevo = primero;
221     if (numElem == 1) {
222         viejo = primero;
223         var = viejo->date;
224         delete viejo;
225         numElem--;
226         primero = NULL;
227         ultimo = NULL;
228         return var;
229     } else {
230         while (nuevo && i < pos) {
231             i++;
232             nuevo = nuevo->sige;
233         }
234         viejo = nuevo->sige;
235         nuevo->sige = viejo->sige;
236         var = viejo->date;
237         delete viejo;
238         numElem--;
239     }
240 }
241 }
```

Figura 82: Captura de pantalla del error cppcheck en el código fuente

```

91 /**
92 * @brief Devuelve una lista con los libros que contengan el título que se le pasa como parámetro.
93 * @param [in] titulo string. Título que deben contener los libros.
94 * @return Lista con todos los libros que coinciden con la búsqueda realizada.
95 */
96 lista_sin<Libro *> * Biblioteca::consultaLibros(string titulo) {
97     unsigned i;
98     int pos = -1;
99     lista_sin<Libro *> *libros = new lista_sin<Libro *>;
100    for (i = 0; i < libro.tamanio(); i++) {
101        pos = libro.lee(i)->datotitulo().find(titulo); // Devuelve la posición en la cadena.
102        if (pos != -1) { // Si no se modifica el -1 entonces es que no está.
103            libros->aumenta(libro.lee(i));
104        }
105    }
106    if (libros->tamano() == 0)
107        throw excepcionesBi::libroNoencontrado();
108    else
109        return libros;
110 }
111 }
```

Figura 83: Captura de pantalla del error cppcheck en el código fuente

#### 4.6.2. Corrección realizada para solventar el error

Para el error existente en la línea 29 de la figura 77 la solución consistirá en realizar la declaración dentro del bloque *if* de la función.

```

22 /**
23 * @brief Aquí se ha solicitado previamente una clave q.
24 * @pre La clave debe ser correcta.
25 */
26 void Application::aplicacion_admin() {
27
28     unsigned i = 0, num, nume_ped_bi, cpb = 0;
29     string aclave, alogin, contra = "hola", contrase;
30     cout << " Introduzca contraseña: " << endl;
31     cin >> contrase;
32     if (contrase.compare(contra) == 0) {
33
34         int opcion;
35
36         do {
37             cout
38                 << " _____ ##### Bienven:
```

Figura 84: Captura de pantalla del error cppcheck resuelto en el código fuente

Para el error que hace referencia a la variable *cpb* existente en la línea 30 de la figura 77 la solución consistirá en realizar la declaración e inicialización dentro del bloque *switch - case* de la función.

```

104     case 3: {
105         unsigned cpb = 0;
106         cout << " Se ha creado el pedido de biblioteca num: " << ++cpb
107             << endl;
108         pedipunt = bi.abrePedidoBiblioteca(--cpb);
109         cpb++;
110     }
111     break;
```

Figura 85: Captura de pantalla del error cppcheck resuelto en el código fuente

Para el error que hace referencia a la variable *i* existente en la línea 30 de la figura 77 la solución consistirá en realizar la declaración e inicialización dentro del bloque *switch - case* de la función.

```

60
61     unsigned i = 0;
62
63     case 1: {
64         i = 0;
65         cout

```

Figura 86: Captura de pantalla del error cppcheck resuelto en el código fuente

Para el error que hace referencia a la variable *i* existente en la línea 16 de la figura 78 la solución consistirá en realizar la declaración e inicialización dentro del bucle *for* de la función.

```

70 /**
8 * @brief Introduce un nuevo Usuario en la biblioteca.
9 * @param [in] login string. Login del usuario. Login del nuevo Usuario.
10 * @param [in] nombre string. Nombre del Usuario. Nombre del nuevo Usuario.
11 * @param [in] clave string. Clave del Usuario. Clave del nuevo Usuario.
12 * @return bool. True si no se puede introducir el Usuario, false en cualquier otro caso.
13 */
140 bool Biblioteca::nuevoUsuario(string login, string nombre, string clave) {
15     usu->rellena(login, nombre, clave);
16     if (usu.tamano() == 0) { // Devuelve true si está el Usuario y ya no se puede intro
17         usu.aumenta(usu);
18         return true;
19     } else {
20         for (unsigned i = 0; i < usu.tamano(); i++) {
21             if (usu->daClave() == usu.lee(i)->daClave()) {
22                 return false;
23             } else {
24                 usu.aumenta(usu);
25                 return true;
26             }
27         }
28     }
29 }
30 }
31

```

Figura 87: Captura de pantalla del error cppcheck resuelto en el código fuente

Para el error que hace referencia a la variable *i* existente en la línea 161 de la figura 80 la solución consistirá en realizar la declaración e inicialización dentro del bloque *else* de la función.

```
150
159@ template<class T>
160 T lista_sin<T>::lee(unsigned pos) {
161     nuevo = primero;
162     if (nuevo == NULL) {
163         throw ErrorElemento();
164     } else {
165         unsigned i = 0;
166         while (nuevo && i < pos) {
167             i++;
168             nuevo = nuevo->sige;
169         }
170         return nuevo->date;
171     }
172 }
173
```

Figura 88: Captura de pantalla del error cppcheck resuelto en el código fuente

Para el error que hace referencia a la variable *i* existente en la línea 176 de la figura 79 la solución consistirá en realizar la declaración e inicialización dentro del bloque *else* de la función.

```
1/3
174@ template<class T>
175 void lista_sin<T>::modifica(T elem, unsigned pos) {
176     nodo<T> *iter;
177     iter = primero;
178     if (iter == NULL) {
179         throw ErrorElemento();
180     } else {
181         unsigned i = 0;
182         while (iter && i < pos) {
183             i++;
184             iter = iter->sige;
185         }
186         iter->date = elem;
187     }
188 }
```

Figura 89: Captura de pantalla del error cppcheck resuelto en el código fuente

Para el error que hace referencia a la variable *i* existente en la línea 194 de la figura 81 la solución consistirá en realizar la declaración e inicialización dentro del bloque *else* de la función.

```
189
190@template<class T>
191 void lista_sinc<T>::inserta_dato(const T &dato, unsigned pos) {
192     nodo<T> *iter;
193     iter = new struct nodo<T>;
194     if (ultimo == NULL) {
195         iter->sige = NULL;
196         iter->date = dato;
197         primero = ultimo = iter;
198     } else {
199         unsigned i = 1;
200         nuevo = primero;
201         while ((i < pos) && nuevo) {
202             nuevo = nuevo->sige;
203             i++;
204         }
205         if (nuevo != NULL) {
206             iter->sige = nuevo->sige;
207             iter->date = dato;
208             nuevo->sige = iter;
209         }
210     }
211     numElem++; // Incremento "numElem" porque hay un dato mas en la lista.
212 }
213 }
```

Figura 90: Captura de pantalla del error cppcheck resuelto en el código fuente

Para el error que hace referencia a la variable *i* existente en la línea 216 de la figura 82 la solución consistirá en realizar la declaración e inicialización dentro del bloque *else* de la función.

```

213
214④ template<class T>
215 T lista_sin<T>::elimina_dato(unsigned pos) {
216     T var;
217     nodo<T> *viejo;
218     viejo = primero;
219     nuevo = primero;
220     if (numElem == 1) {
221         viejo = primero;
222         var = viejo->date;
223         delete viejo;
224         numElem--;
225         primero = NULL;
226         ultimo = NULL;
227         return var;
228     } else {
229         unsigned i = 1;
230         while (nuevo && i < pos) {
231             i++;
232             nuevo = nuevo->sige;
233         }
234         viejo = nuevo->sige;
235         nuevo->sige = viejo->sige;
236         var = viejo->date;
237         delete viejo;
238         numElem--;
239         return var;
240     }
241 }
242

```

Figura 91: Captura de pantalla del error cppcheck resuelto en el código fuente

Para el error que hace referencia a la variable *num* existente en la línea 30 de la figura 77 la solución consistirá en realizar la declaración e inicialización dentro del bloque *switch - case* de la función.

```

90
91     case 2: {
92         unsigned num;
93         cout
94             << " Introduzca la numeración del pedido de la biblioteca que quiere tramitar: "
95             << endl;
96         cin >> num;
97         num--;
98         try {
99             pedBi = bi.dalistaPedBiblioteca(num);
100            bi.cierraPedidoBiblioteca(pedBi, num);
101        } catch (bad_alloc&) {
102        } catch (const excepcionesBi::pedidoBibliotecaNoencontrado& e) {
103            cerr << e.what();
104        }
105    }
106    break;
107

```

Figura 92: Captura de pantalla del error cppcheck resuelto en el código fuente

Para el error que hace referencia a la variable *nume\_ped.bi* existente en la línea 30 de la figura 77 la solución consistirá en realizar la declaración e inicialización dentro del bloque *switch - case* de la función.

```

115     case 4: {
116         unsigned nume_ped_bi;
117         PedidoUsuario * min = new PedidoUsuario;
118         cout
119             << " Introduzca los datos del usuario del que quiere tramitar sus pedidos: "
120             << endl;
121         cout << " Introduzca la clave del usuario: " << endl;
122         cin >> aclave;
123         cout << " Introduzca el login: " << endl;
124         cin >> alogin;

```

Figura 93: Captura de pantalla del error cppcheck resuelto en el código fuente

Para el error que hace referencia a la variable *pos* existente en la línea 98 de la figura 83 la solución consistirá en realizar la declaración e inicialización dentro del bucle *for* de la función.

```

90
91 /**
92 * @brief Devuelve una lista con los libros que contengan el título que se le pasa como parámetro.
93 * @param [in] título string. Título que deben contener los libros.
94 * @return Lista con todos los libros que coinciden con la búsqueda realizada.
95 */
96 lista_sin<Libro *> * Biblioteca::consultaLibros(string título) {
97     unsigned i;
98     lista_sin<Libro *> *libros = new lista_sin<Libro *>;
99     for (i = 0; i < libro.tamano(); i++) {
100         int pos = libro.lee(i)->daTitulo().find(título); // Devuelve la posición en la cadena.
101         if (pos != -1) { // Si no se modifica el -1 entonces es que no está.
102             libros->aumenta(libro.lee(i));
103         }
104     }
105     if (libros->tamano() == 0)
106         throw excepcionesBi::libroNoencontrado();
107     else
108         return libros;
109 }
110

```

Figura 94: Captura de pantalla del error cppcheck resuelto en el código fuente

## 4.7. Error nº7: Unused variable

### 4.7.1. Origen/explicación del error detectado

Cppcheck nos indica en este caso que el error se origina en la línea 232 del archivo *Application.cpp*, más concretamente en la variable *basura*.

```

229 void Application::aplicacion_usuario() {
230
231     int opcion;
232     string alogin, aclave, anombre, aISBN, atitulo, claven, basura;
233     bool var;
234     unsigned i;
235     try {
236         bi.cargaLibros("../libros.txt");
237     } catch (bad_alloc& ) {
238     } catch (const excepcionesBi::errorApertura& e) {
239         cerr << e.what();
240     }
241

```

Figura 95: Captura de pantalla del error cppcheck en el código fuente

El error se origina debido a que declaramos esta variable pero no llegamos a utilizarla en toda la función, generando en consecuencia un desperdicio

de la memoria local de la función. Por este motivo, cppcheck nos alerta al respecto.

Figura 96: Detalle del mensaje de error de cppcheck

#### 4.7.2. Corrección realizada para solventar el error

La resolución de este error consiste en la eliminación de dicha variable de nuestro código. Una vez realizado esto, el error habrá sido eliminado.

```
229@ void Application::aplicacion_usuario() {
230
231     int opcion;
232     string alogin, clave, nombre, aISBN, atitulo, claven;
233     bool var;
234     unsigned i;
235     try {
236         bi.cargaLibros("../libros.txt");
237     } catch (bad_alloc& ) {
238     } catch (const excepcionesBi::errorApertura& e) {
239         cerr << e.what();
240     }
241 }
```

Figura 97: Captura de pantalla del error cppcheck resuelto en el código fuente

### 4.8. Error nº8: Variable is reassigned a value before the old one has been used

#### 4.8.1. Origen/explicación del error detectado

Este error tiene su origen en la línea 221 del archivo *lista\_sin.h*.

```
214@ template<class T>
215 T lista_sin<T>::elimina_dato(unsigned pos) {
216     T var;
217     nodo<T> *viejo;
218     viejo = primero;
219     nuevo = primero;
220     if (numElem == 1) {
221         viejo = primero;
222         var = viejo->date;
223         delete viejo;
224         numElem--;
225         primero = NULL;
226         ultimo = NULL;
227         return var;
228     } else {
229         unsigned i = 1;
230         while (nuevo && i < pos) {
231             i++;
232             nuevo = nuevo->sige;
233         }
234         viejo = nuevo->sige;
235         nuevo->sige = viejo->sige;
236         var = viejo->date;
237         delete viejo;
238         numElem--;
239         return var;
240     }
241 }
```

Figura 98: Captura de pantalla del error cppcheck en el código fuente

Este error se origina cuando, tras declarar la variable *viejo* en la línea 217, se le asigna la dirección de memoria que hay guardada en la variable *primero* (línea 218).

Una vez que hemos hecho esto, en la línea 221 volvemos a asignarle el valor de *primero*. En nuestro código no habría problema, pero si en la primera asignación le hubiéramos pasado otro valor no habríamos llegado a realizar ninguna operación con este primer valor. Es decir, hemos guardado una información que no hemos utilizado para nada. En consecuencia, habremos realizado una operación innecesaria (primera asignación). Este es el porqué de que cppcheck nos lance una advertencia al respecto.

(cppcheck style) Variable 'viejo' is reassigned a value before the old one has been used.

lista\_sin.h

/prac3GS

line 221

cppcheck Problem

Figura 99: Detalle del mensaje de error de cppcheck

#### 4.8.2. Corrección realizada para solventar el error

La solución a este error pasa por eliminar la segunda asignación del código fuente (línea 221), ya que la primera asignación es válida al entrar en la ejecución del bloque *if*. De este modo, el error habrá sido solventado.

```

213
214@ template<class T>
215 T lista_sin<T>::elimina_dato(unsigned pos) {
216     T var;
217     nodo<T> *viejo;
218     viejo = primero;
219     nuevo = primero;
220     if (numElem == 1) {
221         var = viejo->date;
222         delete viejo;
223         numElem--;
224         primero = NULL;
225         ultimo = NULL;
226         return var;
227     } else {
228         unsigned i = 1;
229         while (nuevo && i < pos) {
230             i++;
231             nuevo = nuevo->sige;
232         }
233         viejo = nuevo->sige;
234         nuevo->sige = viejo->sige;
235         var = viejo->date;
236         delete viejo;
237         numElem--;
238         return var;
239     }
240 }
```

Figura 100: Captura de pantalla del error cppcheck resuelto en el código fuente

## 4.9. Error nº9: Either the condition is redundant or there is possible null pointer dereference

### 4.9.1. Origen/explicación del error detectado

Este error se encuentra en la línea 186 del archivo *lista\_sin.h*, en la siguiente captura de pantalla se muestra la función en la que se encuentra dicho error.

```

173
1740 template<class T>
175     void lista_sin<T>::modifica(T elem, unsigned pos) {
176         nodo<T> *iter;
177         iter = primero;
178         if (iter == NULL) {
179             throw ErrorElemento();
180         } else {
181             unsigned i = 0;
182             while (iter && i < pos) {
183                 i++;
184                 iter = iter->sige;
185             }
186             iter->date = elem;
187         }
188     }
189 }
190

```

Figura 101: Captura de pantalla del error cppcheck en el código fuente

Este error es causado porque no comprobamos que el puntero *iter* no esté apuntando a NULL. En otra situación, estando *iter* inicializado correctamente, no hubiera sido necesario realizar esta comprobación pero, como se puede observar en la captura de pantalla, dentro del bucle *while* se va actualizando la dirección a la que apunta con el valor de *sige*.

Al final de la ejecución del bucle *while* es posible que *iter* esté apuntando a NULL, de hecho una de las condiciones de parada del bucle es que esté apuntando a NULL. Por este motivo cppcheck nos alerta de este hecho.

```
(cppcheck warning) Either the condition 'iter' is redundant or there is possible null pointer dereference: iter. lista_sin.h /prac3GS line 186 cppcheck Problem
```

Figura 102: Detalle del mensaje de error de cppcheck

### 4.9.2. Corrección realizada para solventar el error

La manera de subsanar este error es realizar la comprobación de que no esté apuntando a NULL justo antes de acceder al objeto apuntado. En la captura de pantalla que se encuentra a continuación se puede observar con detalle la solución aplicada.

```

1/3
174@ template<class T>
175 void lista_sin<T>::modifica(T elem, unsigned pos) {
176     nodo<T> *iter;
177     iter = primero;
178     if (iter == NULL) {
179         throw ErrorElemento();
180     } else {
181         unsigned i = 0;
182         while (iter && i < pos) {
183             i++;
184             iter = iter->sige;
185         }
186         if (iter != NULL) {
187             iter->date = elem;
188         }
189     }
190 }
```

Figura 103: Captura de pantalla del error cppcheck subsanado en el código fuente

## 4.10. Error nº10: Possible leak in public function. The pointer is not deallocated before is allocated

### 4.10.1. Origen/explicación del error detectado

En la siguiente captura de pantalla se muestra la función que contiene el error de cppcheck.

```

144@ template<class T>
145 void lista_sin<T>::aumenta(T elem) {
146     nuevo = new struct nodo<T>;
147     nuevo->date = elem;
148     nuevo->sige = NULL;
149     numElem++;
150     if (primero == NULL) {
151         primero = nuevo;
152         ultimo = nuevo;
153     } else {
154         ultimo->sige = nuevo;
155         ultimo = nuevo;
156     }
157 }
```

Figura 104: Captura de pantalla del error cppcheck en el código fuente

Se puede observar que en la línea 146 se le asigna la dirección de memoria de un nuevo *nodo* al puntero *nuevo*. *nuevo* es un atributo miembro de la clase y es posible que ya estuviera apuntando a otra dirección de memoria y, por lo tanto, es posible que se esté produciendo una fuga de memoria. Es por esto que cppcheck nos lanza la advertencia.

(cppcheck warning) Possible leak in public function. The pointer 'nuevo' is not deallocated before it is allocated. lista\_sin.h /prac3GS line 146 cppcheck Problem

Figura 105: Detalle del mensaje de error de cppcheck

### 4.10.2. Corrección realizada para solventar el error

En la captura de pantalla siguiente se puede observar la solución aplicada al código fuente para solventar el error. Esta solución consiste en liberar la

memoria de *nuevo*, previa comprobación de que no esté apuntando a NULL para evitar posibles errores.

```

144  template<class T>
145  void lista_sinc<T>::aumenta(T elem) {
146      if(nuevo==NULL){
147          delete nuevo;
148      }
149      nuevo = new struct nodo<T>;
150      nuevo->date = elem;
151      nuevo->sige = NULL;
152      numElem++;
153      if (primero == NULL) {
154          primero = nuevo;
155          ultimo = nuevo;
156      } else {
157          ultimo->sige = nuevo;
158          ultimo = nuevo;
159      }
160  }
161

```

Figura 106: Captura de pantalla del error cppcheck subsanado en el código fuente

#### 4.11. Warning N° 1 y 2

[cppcheck warning] Class 'Aplication' does not have a copy constructor which is recommended since it has dynamic memory/resource allocation(s).

[cppcheck warning] Class 'Aplication' does not have a operator= which is recommended since it has dynamic memory/resource allocation(s).

##### 4.11.1. Origen/explicación del warning detectado:

```

Application::Application() {
    pedusu = new lista_sinc<PedidoUsuario *>;
    peddi = new lista_sinc<PedidoBiblioteca *>;
    libro = new lista_sinc<Libro *>;
    libro = new Libro();
    peddipunt = new PedidoBiblioteca;
    peddipunt = NULL;
    peddbi = new PedidoBiblioteca;
}
```
* @brief Aquí se ha solicitado previamente una clave que, si coincide, da entrada al esquema de admin.
* @param La clave debe ser correcta.
```
void Aplication::aplicacion_admin() {
    string clave, login, contra = "Hola", contrase;
    cout << "Introduzca contraseña: " << endl;
    cin >> contrase;
    if (contrase.compare(contra) == 0) {
        int opcion;

```

Se observa el warning en la línea 16 que nos informa que sería recomendable para la clase Aplicación el contar con un constructor de copia debido a que cuenta con recursos en memoria dinámica, asimismo, se nos ofrece en esta misma línea otro warning diciendo que sería recomendable la implementación de un operador de igualdad por el mismo motivo..

#### 4.11.2. Corrección realizada para solventar el warning:

```

37     Application::Application& operator=(const Application& apl) {
38         this->bi = apl.bi;
39         this->usu = apl.usu;
40         this->lusu = apl.lusu;
41         this->li = apl.li;
42         this->pedBi = apl.pedBi;
43         this->pedbi = apl.pedbi;
44         this->pedusu = apl.pedusu;
45         this->libro = apl.libro;
46         this->pedbipunt = apl.pedbipunt;
47         return *this;
48     }

```

Este mismo warning nos aparece, pero para la clase Biblioteca que nos dice que:

```

138     Biblioteca::Biblioteca& operator=(const Biblioteca& bib) {
139         this->usu = bib.usu;
140         this->pedido_usu = bib.pedido_usu;
141         this->pedidoBi = bib.pedidoBi;
142         this->libro = bib.libro;
143         this->usu = bib.usu;
144         return *this;
145     }
146

```

[cppcheck warning] Class 'Aplication' does not have a operator= which is recommended since it has dynamic memory/resource allocation(s).

#### 4.12. Warning N° 5

[cppcheck warning] Member variable 'Libro::anio' is not assigned a value in 'Libro::operator='.

##### 4.12.1. Origen/explicación del warning detectado:

```

/*
 * @brief Operador de asignación de la clase Libro.
 * @param [in] lib Libro(dir). Instancia de la clase Libro de la cual se va a realizar una copia.
 * @return Devuelve una nueva instancia de la clase Libro con la información de lib.
 */
Libro& operator=(const Libro &lib) {
    this->título = lib.título;
    this->autores = lib.autores;
    this->editorial = lib.editorial;
    this->ISBN = lib.ISBN;
    return *this;
}


```

Se observa el warning en la línea seleccionada que nos informa que la variable miembro anio no tiene asignado valor en el operador de igualdad.

#### 4.12.2. Corrección realizada para solventar el warning:

```
/*
 * @brief Operador de asignación de la clase Libro.
 * @param [in] lib Libro(dir). Instancia de la clase Libro de la cual se va a realizar una copia.
 * @return Devuelve una nueva instancia de la clase Libro con la información de lib.
 */
Libro& operator=(const Libro &lib) {
    this->titulo = lib.titulo;
    this->autores = lib.autores;
    this->editorial = lib.editorial;
    this->ISBN = lib.ISBN;
    return *this;
}
```

Se coloca dentro del operador de igualdad para que la variable miembro autores pueda tener un valor asignado.

### 4.13. Warning Nº 6

[cppcheck warning] Member variable 'Libro::precioActual' is not assigned a value in 'Libro::operator='.

#### 4.13.1. Origen/explicación del warning detectado:

```
/*
 * @brief Operador de asignación de la clase Libro.
 * @param [in] lib Libro(dir). Instancia de la clase Libro de la cual se va a realizar una copia.
 * @return Devuelve una nueva instancia de la clase Libro con la información de lib.
 */
Libro& operator=(const Libro &lib) {
    this->titulo = lib.titulo;
    this->autores = lib.autores;
    this->editorial = lib.editorial;
    this->ISBN = lib.ISBN;
    return *this;
}
```

Se observa el warning en la línea seleccionada que nos informa que la variable miembro precioActual no tiene asignado valor en el operador de igualdad.

#### 4.13.2. Corrección realizada para solventar el warning:

```
/*
 * @brief Operador de asignación de la clase Libro.
 * @param [in] lib Libro(dir). Instancia de la clase Libro de la cual se va a realizar una copia.
 * @return Devuelve una nueva instancia de la clase Libro con la información de lib.
 */
Libro& operator=(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;
    return *this;
}
```

Se coloca dentro del operador de igualdad para que precioActual pueda tener un valor asignado.

## 4.14. Warning N° 7

[cppcheck warning] Member variable 'lista.sin < Libro \* >::numElem' is not assigned a value in 'lista.sin < Libro \* >::operator='.

### 4.14.1. Origen/explicación del warning detectado:

```
template<class T>
Lista_sin<T>& Lista_sin<T>::operator=(Lista_sin &list) {
    this->limpia();
    for (unsigned i = 0; i < list.tamano(); i++) {
        this->aumenta(list.lee(i)); // Gracias a "aumenta" se reserva memoria para cada nodo.
    }
    return (*this);
}
```

En este warning en la línea seleccionada que nos informa que la variable miembro numElem no tiene asignado valor en el operador de igualdad.

### 4.14.2. Corrección realizada para solventar el warning:

```
template<class T>
Lista_sin<T>& Lista_sin<T>::operator=(Lista_sin &list) {
    this->limpia();
    for (unsigned i = 0; i < list.tamano(); i++) {
        this->aumenta(list.lee(i)); // Gracias a "aumenta" se reserva memoria para cada nodo.
    }
    this->numElem = list.numElem;
    return (*this);
}
```

Se coloca dentro del operador de igualdad para que numElem pueda tener un valor asignado.

**Nota:** Al arreglar los dos últimos warnings, Cppcheck ha obviado los siguientes warnings, siempre referentes a la misma variable miembro 'numElem'... a saber:

[cppcheck warning] Member variable 'lista.sin < PedidoUsuario \* >::numElem' is not assigned a value in 'lista.sin < PedidoUsuario \* >::operator='.

[cppcheck warning] Member variable 'lista.sin < Usuario \* >::numElem' is not assigned a value in 'lista.sin < Usuario \* >::operator='.

[cppcheck warning] Member variable 'lista.sin < Usuario >::numElem' is not assigned a value in 'lista.sin < Usuario >::operator='.

[cppcheck warning] Member variable 'lista.sin::numElem' is not assigned a value in 'lista.sin::operator='.

## 4.15. Warning N° 13

[cppcheck warning] Member variable 'PedidoBiblioteca::num' is not assigned a value in 'PedidoBiblioteca::operator='.

### 4.15.1. Origen/explicación del warning detectado:

En este warning en la línea seleccionada que nos informa que la variable miembro num de la clase PedidoBiblioteca no tiene asignado valor en el operador de igualdad.

```
PedidoBiblioteca& operator=(const PedidoBiblioteca& pedbi) {
    this->fecha = pedbi.fecha;
    this->importe = pedbi.importe;
    this->tramitado = pedbi.tramitado;
    this->pedido_usu = pedbi.pedido_usu;
    return *this;
}
```

### 4.15.2. Corrección realizada para solventar el warning:

```
PedidoBiblioteca& operator=(const PedidoBiblioteca& pedbi) {
    this->fecha = pedbi.fecha;
    this->importe = pedbi.importe;
    this->tramitado = pedbi.tramitado;
    this->num = pedbi.num;
    this->pedido_usu = pedbi.pedido_usu;
    return *this;
}
```

Se coloca dentro del operador de igualdad para que num pueda tener un valor asignado.

## 4.16. INFO N.º 1

[cppcheck information] Cppcheck cannot find all the include files (use --check-config for details)

### 4.16.1. Origen/explicación del info detectado:

En este info se nos informa que Cppcheck no puede encontrar todos los archivos Include.

### 4.16.2. Corrección realizada para solventar el info:

## 4.17. INFO N.º 2, 3, 4 y 5

[cppcheck performance] Function parameter 'aTitulo' should be passed by const reference.

[cppcheck performance] Function parameter 'aAutores' should be passed by const reference.

[cppcheck performance] Function parameter 'aEditorial' should be passed by const reference.

[cppcheck performance] Function parameter 'aISBN' should be passed by const reference.

#### 4.17.1. Origen/explicación del info detectado:

```
1 libro::Libro(string aTitulo, string aAutores, string aEditorial, string aISBN, int aAnio, float aPrecioActual) {
2     titulo = aTitulo;
3     autores = aAutores;
4     editorial = aEditorial;
5     ISBN = aISBN;
6     anio = aAnio;
7     precioActual = aPrecioActual;
8 }
```

En este info se nos informa que el parámetro de la función aTitulo debería ser pasado por referencia constante, al igual que aAutores, aEditorial y aISBN.

#### 4.17.2. Corrección realizada para solventar el info:

```
1 libro::Libro(const string aTitulo, const string aAutores, const string aEditorial, const string aISBN, int aAnio, float aPrecioActual) {
2     titulo = aTitulo;
3     autores = aAutores;
4     editorial = aEditorial;
5     ISBN = aISBN;
6     anio = aAnio;
7     precioActual = aPrecioActual;
8 }
```

Le hemos añadido la palabra reservada const antes del tipo de las variables.

**NOTA** Haciendo esto mismo, hemos logrado eliminar 4 Infos.

### 4.18. INFO N.º 6, 7 y 8

[cppcheck performance] Function parameter 'anombre' should be passed by const reference.

[cppcheck performance] Function parameter 'alogin' should be passed by const reference.

[cppcheck performance] Function parameter 'aclave' should be passed by const reference.

#### 4.18.1. Origen/explicación del info detectado:

```

40 void Usuario::rellena(string anombre, string alogin, string aclave) {
41     nombre = anombre;
42     clave = aclave;
43     login = alogin;
44 }
```

En este info se nos informa que el parámetro de la función anombre, al igual que alogin y aclave, deberían ser pasados por referencia constante, al igual la info anterior.

#### 4.18.2. Corrección realizada para solventar el info:

```

34 /**
35  * @brief Función que asigna a un Usuario su nombre, su Clave, y por supuesto su Login.
36  * @param [in] anombre string. Nombre del Usuario.
37  * @param [in] alogin string. Login del Usuario.
38  * @param [in] aclave string. Clave del Usuario.
39 */
40 void Usuario::rellena(const string anombre, const string alogin, const string aclave) {
41     nombre = anombre;
42     clave = aclave;
43     login = alogin;
44 }
```

Le hemos añadido la palabra reservada const antes del tipo de las variables.

**NOTA** Haciendo esto mismo, hemos logrado eliminar otras 2 Infos.

### 4.19. INFO N.<sup>º</sup> 9

[cppcheck performance] Function parameter 'aFecha' should be passed by const reference.

#### 4.19.1. Origen/explicación del info detectado:

```

38 PedidoUsuario::PedidoUsuario(Fecha aFecha, int aPrioridad, float aPrecio, bool aTramitado, Usuario *usuario, Libro *libro) {
39     fecha = aFecha;           ///Copia de la fecha que queda registrada al hacer un pedido.
40     prioridad = aPrioridad;  ///Copia de la prioridad que queda registrada al hacer un pedido.
41     precio = aPrecio;        ///Copia del precio de un pedido.
42     tramitado = aTramitado;  ///Copia de la tramitación de un pedido.
43     this->usuario = usuario; ///Referencia al usuario mediante el objeto this.
44     this->libro = libro;     ///Referencia al libro mediante el objeto this.
45 }
```

En este info se nos informa que el parámetro de la función aFecha debería ser pasado por referencia constante, al igual las infos anteriores.

#### 4.19.2. Corrección realizada para solventar el info:

```

46 PedidoUsuario::PedidoUsuario(const Fecha &fecha, int &prioridad, float &precio, bool &enviado, Usuario *&usuario, Libro *&libro) {
47     fecha = fecha; // Copia de la fecha que queda registrada al hacer un pedido.
48     prioridad = prioridad; // Copia de la prioridad que queda registrada al hacer un pedido.
49     precio = precio; // Copia del precio de un pedido.
50     enviado = enviado; // Copia de si el pedido ha sido enviado o no.
51     this->usuario = usuario; // Referencia al usuario mediante el objeto this.
52     this->libro = libro; // Referencia al libro mediante el objeto this.
53 }

```

Le hemos añadido la palabra reservada const antes del tipo de la variable.

#### 4.20. INFO N.º 10 y 11

[cppcheck performance] Function parameter 'login' should be passed by const reference.

[cppcheck performance] Function parameter 'clave' should be passed by const reference.

#### 4.20.1. Origen/explicación del info detectado:

```

46 Usuario* Biblioteca::buscaUsuario(string login, string clave) {
47
48     for (unsigned i = 0; i < usur.tamanio(); i++) {
49         if (usur.lee(i)->daLogin() == login
50             && usur.lee(i)->daClave() == clave) {
51             return usur.lee(i);
52         }
53     }
54     throw excepcionesBi::usuNoEncontrado();
55 }

```

En este info se nos informa que el parámetro de la función login y clave deberían ser pasado s por referencia constante, al igual las infos anteriores.

#### 4.20.2. Corrección realizada para solventar el info:

```

46 Usuario* Biblioteca::buscaUsuario(const string login, const string clave) {
47
48     for (unsigned i = 0; i < usur.tamanio(); i++) {
49         if (usur.lee(i)->daLogin() == login
50             && usur.lee(i)->daClave() == clave) {
51             return usur.lee(i);
52         }
53     }
54     throw excepcionesBi::usuNoEncontrado();
55 }

```

Le hemos añadido la palabra reservada const antes del tipo de las variables.

**NOTA** Haciendo esto mismo, hemos logrado eliminar otras Info.

## 4.21. INFO N.<sup>º</sup> 12 y N.<sup>º</sup> 13

[cppcheck performance] Function parameter 'claven' should be passed by const reference.

[cppcheck performance] Function parameter 'claven' should be passed by const reference.

### 4.21.1. Origen/explicación de los infos detectados:

```
51 */
52 bool Usuario::validarClave(string claven) {
53
54     if (this->clave == claven)
55         return true;
56     else
57         return false;
58 }
59 /**
60 * @brief Cambia el atributo clave con el string pasado como parámetro.
61 * @param [in] nuevaClave string.
62 * @pre Antes ha de ser utilizada la función anterior.
63 */
64 void Usuario::cambiarClave(string claven) {
65     this->clave = claven;
66 }
67 }
```

En estos info se nos informa que los parámetros claven de ambas funciones, deberían ser pasados por referencia constante, al igual que los infos anteriores.

### 4.21.2. Corrección realizada para solventar los infos:

```
52 bool Usuario::validarClave(const string claven) {
53
54     if (this->clave == claven)
55         return true;
56     else
57         return false;
58 }
59 /**
60 * @brief Cambia el atributo clave con el string pasado como parámetro.
61 * @param [in] nuevaClave string.
62 * @pre Antes ha de ser utilizada la función anterior.
63 */
64 void Usuario::cambiarClave(const string claven) {
65     this->clave = claven;
66 }
67 }
```

Le hemos añadido la palabra reservada const antes del tipo de las variables.

## 4.22. INFO N.<sup>º</sup> 14

[cppcheck performance] Function parameter 'elem' should be passed by const reference.

#### 4.22.1. Origen/explicación de los infos detectados:

```

145 template<class T>
146 void lista_sinc<T>::aumenta(T elem) {
147     if(nuevo==NULL){
148         delete nuevo;
149     }

```

En este info se nos informa que el parámetro elem, debería ser pasado por referencia constante, al igual que los infos anteriores.

#### 4.22.2. Corrección realizada para solventar el info:

```

template<class T>
void lista_sinc<T>::aumenta(T const &elem) {
    if(nuevo==NULL){
        delete nuevo;
    }
}

```

Le hemos añadido la palabra reservada const antes del tipo de la variable y la referencia.

### 4.23. INFO N.<sup>º</sup> 15

[cppcheck performance] Function parameter 'elem' should be passed by const reference.

#### 4.23.1. Origen/explicación de los infos detectados:

```

template<class T>
void lista_sinc<T>::modifica(T elem, unsigned pos) {
    nodo<T> *iter;
    iter = primero;
    if (iter == NULL) {
        throw ErrorElemento();
    }
}

```

En este info se nos informa que el parámetro elem, debería ser pasado por referencia constante, al igual que los infos anteriores.

#### 4.23.2. Corrección realizada para solventar el info:

```

template<class T>
void lista_sinc<T>::modifica(T const &elem, unsigned pos) {
    nodo<T> *iter;
    iter = primero;
    if (iter == NULL) {
        throw ErrorElemento();
    } else {

```

Le hemos añadido la palabra reservada const antes del tipo de la variable y la referencia.

#### 4.24. INFOS N.<sup>o</sup> 16 y 17

[cppcheck performance] Function parameter 'aFecha' should be passed by const reference.

[cppcheck performance] Variable 'fecha' is assigned in constructor body. Consider performing initialization in initialization list.

##### 4.24.1. Origen/explicación de los infos detectados:

```
39 PedidoUsuario::PedidoUsuario(const Fecha &fecha, int aPrioridad, float aPrecio, bool aTramitado, Usuario *usuario, Libro *libro) {
40     fecha = aFecha; // Copia de la Fecha que queda registrada al hacer un pedido.
41     prioridad = aPrioridad; // Copia de la prioridad que queda registrada al hacer un pedido.
42     precio = aPrecio; // Copia del precio de un pedido.
43     tramitado = aTramitado; // Copia de la tramitación de un pedido.
44     this->usuario = usuario; // Referencia al usuario mediante el objeto this.
45     this->libro = libro; // Referencia al libro mediante el objeto this.
46 }
```

En este info se nos informa que el parámetro aFecha, debería ser pasado por referencia constante, al igual que los infos anteriores, del mismo modo, al solventar este info nos damos cuenta que nos desaparece otro más, el de que la variable fecha es asignada en el cuerpo del constructor y que deberíamos considerar inicializarla en la lista.

##### 4.24.2. Corrección realizada para solventar los dos infos:

```
39 PedidoUsuario::PedidoUsuario(const Fecha &fecha, int aPrioridad, float aPrecio, bool aTramitado, Usuario *usuario, Libro *libro) {
40     fecha = aFecha; // Copia de la Fecha que queda registrada al hacer un pedido.
41     prioridad = aPrioridad; // Copia de la prioridad que queda registrada al hacer un pedido.
42     precio = aPrecio; // Copia del precio de un pedido.
43     tramitado = aTramitado; // Copia de la tramitación de un pedido.
44     this->usuario = usuario; // Referencia al usuario mediante el objeto this.
45     this->libro = libro; // Referencia al libro mediante el objeto this.
46 }
```

Le hemos añadido la palabra reservada const antes del tipo de la variable aFecha junto con su referencia. De este modo, Cppcheck ha considerado que la variable fecha es inicializada en la lista en lugar del cuerpo del constructor y nos ha eliminado dicha info.

#### 4.25. INFOS N.<sup>o</sup> 18, 19, 20 y 21

[cppcheck performance] Variable 'titulo' is assigned in constructor body. Consider performing initialization in initialization list.

[cppcheck performance] Variable 'autores' is assigned in constructor body. Consider performing initialization in initialization list.

[cppcheck performance] Variable 'editorial' is assigned in constructor body. Consider performing initialization in initialization list.

[cppcheck performance] Variable 'ISBN' is assigned in constructor body. Consider performing initialization in initialization list.

#### 4.25.1. Origen/explicación de los infos detectados:

```
1  /**
2  * @brief Constructor por defecto de la clase Libro.
3  * @pre Al estar inicializando un objeto de la clase, todos los atributos aparecen vacíos o inicializados a cero.
4  */
5 Libro::Libro() {
6     titulo = "";
7     autores = "";
8     editorial = "";
9     ISBN = "";
10    año = 0;
11    precioActual = 0;
12 }
```

En estos cuatro infos se nos informa que las variables título, autores, editorial y ISBN son asignados en el cuerpo del constructor y que deberíamos considerar inicializarlos en la lista.

#### 4.25.2. Corrección realizada para solventar los cuatro infos:

```
9 /**
10 * @brief Constructor por defecto de la clase Libro.
11 * @pre Al estar inicializando un objeto de la clase, todos los atributos aparecen vacíos o inicializados a cero.
12 */
13 Libro::Libro(string atitulo = "", string aautores = "", string aeditorial = "", string aISBN = "") : titulo(atitulo), autores(aautores),
14     editorial(aeditorial), ISBN(aISBN) {
15     año = 0;
16     precioActual = 0;
17 }
```

De este modo, Cppcheck ha considerado que las variables anteriores quedan inicializadas en la lista en lugar del cuerpo del constructor y nos ha eliminado dichas infos.

#### 4.26. INFOS N.<sup>º</sup> 22, 23 y 24

[cppcheck performance] Variable 'nombre' is assigned in constructor body. Consider performing initialization in initialization list.

[cppcheck performance] Variable 'clave' is assigned in constructor body. Consider performing initialization in initialization list.

[cppcheck performance] Variable 'login' is assigned in constructor body. Consider performing initialization in initialization list.

#### 4.26.1. Origen/explicación de los infos detectados:

```

9 /**
10 * @brief Constructor por defecto de la clase Usuario.
11 */
12 Usuario::Usuario() {
13     nombre = "";
14     clave = "";
15     login = "";
16 }

```

En estos tres infos se nos informa que las variables nombre, clave y login son asignados en el cuerpo del constructor y que deberíamos considerar inicializarlos en la lista.

#### 4.26.2. Corrección realizada para solventar los tres infos:

```

9 /**
10 * @brief Constructor por defecto de la clase Usuario.
11 */
12 Usuario::Usuario(string anombre = "", string aclave = "", string alogin = "") : nombre(anombre), clave(aclave), login(alogin){}
13

```

Al igual que hemos procedido anteriormente, Cppcheck ha considerado que las variables anteriores quedan inicializadas en la lista en lugar del cuerpo del constructor y nos ha eliminado dichas infos.

## 5. Reducción de la complejidad ciclomática

### 5.1. Estado del proyecto antes de las correcciones

Antes de empezar de trabajar en esta práctica, comprobaremos cuál es el estado del proyecto antes de empezar a trabajar. Para ello, ejecutaremos los test de cppcheck y metriculator. En las siguientes capturas de pantalla observamos los resultados arrojados.

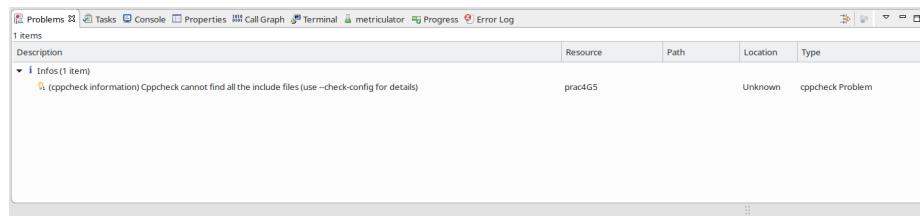


Figura 107: Captura de pantalla de los resultados arrojados tras la ejecución de cppcheck.

Scope (199 items)	LSLOC	EfferentCouplir	McCabe	NbParams	NbMembers
• Application::aplicacion_admin()	138	0	34	0	0
• Application::aplicacion_usuario()	74	0	18	0	0
• Fecha::comprobarFecha(unsigned aDia, unsigned aMes, unsigned aAnio,unsigned aHora, unsigned aMin) const	9	0	12	5	0
• Fecha::operator<(const Fecha &f)	16	0	10	1	0
• * Biblioteca::buscaPedidosBibliotecaPendientes()	11	0	6	0	0

Figura 108: Captura de pantalla de los resultados arrojados tras la ejecución de metriculator.

En los resultados arrojados por cppcheck, podemos observar que tenemos un info: no se encuentran todos los includes del proyecto. Tras buscar información, descubrimos que este error es bastante común ya que cppcheck no es capaz de identificar todas las dependencias que son externas a los archivos del proyecto. Por este motivo, y tras revisar que no falla la sintaxis de ninguno de los includes, la forma de resolver este info es simplemente ignorarlo.

Tras solucionar este info, volvemos a ejecutar el análisis de cppcheck y esta vez no nos devuelve ninguna incidencia. A continuación, se muestra una captura de pantalla como justificación.

Description	Resource	Path	Location	Type

Figura 109: Captura de pantalla de los resultados arrojados tras la ejecución de cppcheck.

Respecto a los resultados arrojados por metriculator, podemos observar que aparecen tres módulos que presentan un  $V(G)>10$  en la métrica de McCabe:

- void Application::aplicacion\_admin()
- void Application::aplicacion\_usuario()
- void Fecha::comprobarFecha(unsigned aDia, unsigned aMes, unsigned aAnio, unsigned aHora, unsigned aMin) const

## 5.2. Corrección del módulo ”void Application :: aplicacion\_admin()”

### 5.2.1. Estado del módulo antes de la corrección

Este módulo se encuentra implementado en la línea 38 del archivo Application.cpp del proyecto. Su complejidad ciclomática es de  $V(G)=34$  en la

métrica de McCabe. En las siguientes capturas de pantalla se puede observar el estado de dicho módulo antes de realizar las correcciones oportunas.

```

34@ /**
35 * @brief Aquí se ha solicitado previamente una clave que, si coincide, da entrada al esquema de admin.
36 * @pre La clave debe ser correcta.
37 */
38@ void Application::aplicacion_admin() {
39     string aclave, alogin, contra = "hola", contrase;
40     cout << " Introduzca contraseña: " << endl;
41     cin >> contrase;
42     if (contrase.compare(contra) == 0) {
43
44         int opcion;
45
46         do {
47             cout
48                 << " ##### Bienvenido a la Administración de la biblioteca. #####\n\n"
49                 << endl;
50             cout
51                 << "De entre las siguientes opciones indique la que quiera elegir, para salir pulse 0: "
52                 << endl << endl;
53             cout << "      1.- Muestra lista de pedidos pendientes de un usuario."
54                 << endl;
55             cout << "      2.- Cierra pedido biblioteca. " << endl;
56             cout << "      3.- Crear pedido biblioteca. " << endl;
57             cout << "      4.- Tramitar pedidos de un usuario. " << endl;
58             cout
59                 << "      5.- Muestra una lista de pedidos de un usuario tramitados. "
60                 << endl;
61             cout
62                 << "      6.- Muestra una lista de pedidos tramitados de la biblioteca. "
63                 << endl;
64             cout
65                 << "      7.- Muestra una lista de pedidos pendientes de la biblioteca. "
66                 << endl;
67             cin >> opcion;
68             switch (opcion) {
69
70             case 1: {
71
72                 cout
73                     << " Introduzca el usuario del cual quiere saber sus pedidos pendientes: "
74                     << endl;
75                 cout << " Introduzca el login: " << endl;
76                 cin >> alogin;
77                 cout << " Introduzca la clave del usuario: " << endl;
78                 cin >> aclave;
79
80             try {
81                 unsigned i = 0;
82                 usu = bi.buscaUsuario(alogin, aclave);
83                 pedusu = bi.buscaPedidosUsuarioPendientes(usu);
84                 while (i < pedusu->tamanio()) {
85                     cout << *(pedusu->lee(i)) << endl;

```

Figura 110: Detalle del módulo a corregir.

```

85             cout << *(pedusu->lee(i)) << endl;
86             i++;
87         }
88     } catch (bad_alloc& ) {
89     } catch (const excepcionesBi::usuNoEncontrado& e) {
90         cerr << e.what();
91     } catch (const excepcionesBi::pedidoUsuarioNoencontrado& e) {
92         cerr << e.what();
93     } catch (const ErrorElemento& e) {
94         cerr << e.what();
95     }
96 }
97 break;
98
99 case 2: {
100     unsigned num;
101     cout
102         << " Introduzca la numeración del pedido de la biblioteca que quiere tramitar: "
103         << endl;
104     cin >> num;
105     num--;
106     try {
107         pedBi = bi.daListaPedBiblioteca(num);
108         bi.cierraPedidoBiblioteca(pedBi, num);
109     } catch (bad_alloc& ) {
110     } catch (const excepcionesBi::pedidoBibliotecaNoencontrado& e) {
111         cerr << e.what();
112     }
113     break;
114
115 case 3: {
116     unsigned cpb = 0;
117     cout << " Se ha creado el pedido de biblioteca num: " << ++cpb
118         << endl;
119     pedipunt = bi.abrePedidoBiblioteca(--cpb);
120     cpb++;
121
122     break;
123
124 case 4: {
125
126     unsigned nume_ped_bi;
127     PedidoUsuario * min = new PedidoUsuario;
128     cout
129         << " Introduzca los datos del usuario del que quiere tramitar sus pedidos: "
130         << endl;
131     cout << " Introduzca la clave del usuario: " << endl;
132     cin >> aclave;
133     cout << " Introduzca el login: " << endl;
134     cin >> alogin;
135     cout
136         << " Introduzca el nombre del usuario: " << endl;

```

Figura 111: Detalle del módulo a corregir.

```

136         cout
137             << " Introduzca el número del pedido de la biblioteca a la que quiere dirigir el pedido del usuario: "
138             << endl;
139             cin >> nume_ped_b1;
140             nume_ped_b1--;
141             try {
142                 unsigned i = 0;
143                 pedBi = bi.dalistaPedBiblioteca(nume_ped_b1); // Obtengo el pedido biblioteca con el número especificado.
144                 usu = bi.buscaUsuario(alogin, aclave);
145                 pedusu = bi.buscaPedidosUsuarioPendientes(usu);
146                 i = 0;
147                 min = pedusu->lee(0);
148                 while (i < pedusu->tamano()) {
149                     if (min>daPrioridad()
150                         > pedusu->lee(i)->daPrioridad()) {
151                         min = pedusu->lee(i);
152                     }
153                     i++;
154                 }
155                 bi.tramitaPedidoUsuario(min, pedBi);
156             } catch (bad_alloc& ) {
157             } catch (const excepcionesBi::usuNoEncontrado& e) {
158                 cerr << e.what();
159             } catch (const excepcionesBi::pedidoBibliotecaNoencontrado& e) {
160                 cerr << e.what();
161             } catch (const ErrorElemento& e) {
162                 cerr << e.what();
163             }
164         }
165         break;
166     case 5: {
167         unsigned i = 0;
168         i = 0;
169         cout
170             << " Introduzca el usuario del que quiere saber sus pedidos pendientes: "
171             << endl;
172         cout << " Introduzca la clave del usuario: " << endl;
173         cin >> aclave;
174         cout << " Introduzca el login: " << endl;
175         cin >> alogin;
176         try {
177             usu = bi.buscaUsuario(alogin, aclave);
178             pedusu = bi.buscaPedidosUsuarioTramitados(usu);
179
180             while (i < pedusu->tamano()) {
181                 cout << *(pedusu->lee(i)) << endl;
182                 i++;
183             }
184         } catch (bad_alloc& ) {
185         } catch (const excepcionesBi::usuNoEncontrado& e) {
186             cerr << e.what();
187         }
    }

```

Figura 112: Detalle del módulo a corregir.

```

187         cerr << e.what();
188     } catch (const ErrorElemento& e) {
189         cerr << e.what();
190     }
191     break;
192 }
193
194 case 6: {
195
196     try {
197         unsigned i = 0;
198         i = 0;
199         pedbi = bi.buscaPedidosBibliotecaTramitados();
200         cout
201             << " La lista de pedidos de la biblioteca tramitados es la siguiente: "
202             << endl;
203         while (i < pedbi->tamanio()) {
204             cout << *(pedbi->lee(i)) << endl;
205             i++;
206         }
207     } catch (bad_alloc& ) {
208     } catch (const excepcionesBi::pedidoBibliotecaNoencontrado& e) {
209         cerr << e.what();
210     } catch (const ErrorElemento& e) {
211         cerr << e.what();
212     }
213     break;
214 }
215
216 case 7: {
217     unsigned i = 0;
218     i = 0;
219     try {
220         pedbi = bi.buscaPedidosBibliotecaPendientes();
221         cout
222             << " La lista de pedidos de la biblioteca pendientes es la siguiente: "
223             << endl;
224         while (i < pedbi->tamanio()) {
225             cout << *(pedbi->lee(i)) << endl;
226             i++;
227         }
228     } catch (bad_alloc& ) {
229     } catch (const excepcionesBi::pedidoBibliotecaNoencontrado& e) {
230         cerr << e.what();
231     } catch (const ErrorElemento& e) {
232         cerr << e.what();
233     }
234 }
235
236 } while (opcion != 0);
237 } else
238     cout << " Contraseña introducida no valida. " << endl;
239 }
```

Figura 113: Detalle del módulo a corregir.

### 5.2.2. Estado del módulo después de la corrección

Como se puede observar en las capturas de pantalla del módulo en el apartado anterior se trata de un módulo con un altísimo número de líneas de código. Una primera estrategia que seguiremos será la de modularizar parte de este código.

En concreto, modularizaremos el código que se encuentra en cada uno de los *case* para reducir la complejidad y la legibilidad del código. Una vez hecho esto, se crearán las siguientes funciones:

- void Application::mostrarMenu()
- void Application::mostrarListaPedidosPendientes(string alogin, string aclave)

- void Application::mostrarTramitarPedido()
- void Application::mostrarTramitarPedidosUsuario(string alogin, string aclave)
- void Application::mostrarConsultarPedidosUsuario(string alogin, string aclave)
- void Application::mostrarListaPedidosTramitados()
- void Application::mostrarListaPedidosPendientesBiblioteca()

A continuación, presentamos capturas de pantalla de todos los módulos enumerados anteriormente.

```

192© /**
193 * @brief Muestra el menú de interacción de la aplicación por pantalla
194 */
195© void Application::mostrarMenu() {
196     cout
197         << " ##### Bienvenido a la Administración de la biblioteca. #####\n\n"
198         << endl;
199     cout
200         << "De entre las siguientes opciones indique la que quiera elegir, para salir pulse 0: "
201         << endl;
202     cout << " 1.- Muestra lista de pedidos pendientes de un usuario." << endl;
203     cout << " 2.- Cierra pedido biblioteca. " << endl;
204     cout << " 3.- Crear pedido biblioteca. " << endl;
205     cout << " 4.- Tramitar pedidos de un usuario. " << endl;
206     cout << " 5.- Muestra una lista de pedidos de un usuario tramitados. "
207         << endl;
208     cout << " 6.- Muestra una lista de pedidos tramitados de la biblioteca. "
209         << endl;
210     cout << " 7.- Muestra una lista de pedidos pendientes de la biblioteca. "
211         << endl;
212 }
213

```

Figura 114: Detalle del módulo mostrarMenu.

```

214© /**
215 * @brief Muestra por pantalla una lista con todos los pedidos pendientes de un usuario determinado
216 * @param [in] alogin string. Login del usuario
217 * @param [in] aclave string. Clave de acceso del usuario
218 */
219© void Application::mostrarListaPedidosPendientes(string alogin, string aclave) {
220     cout
221         << " Introduzca el usuario del cual quiere saber sus pedidos pendientes: "
222         << endl;
223     cout << " Introduzca el login: " << endl;
224     cin >> alogin;
225     cout << " Introduzca la clave del usuario: " << endl;
226     cin >> aclave;
227
228     try {
229         unsigned i = 0;
230         usu = bi.buscaUsuario(alogin, aclave);
231         pedusu = bi.buscaPedidosUsuarioPendientes(usu);
232         while (i < pedusu->tamano()) {
233             cout << *(pedusu->lee(i)) << endl;
234             i++;
235         }
236     } catch (bad_alloc& ) {
237     } catch (const excepcionesBi::usuNoEncontrado& e) {
238         cerr << e.what();
239     } catch (const excepcionesBi::pedidoUsuarioNoencontrado& e) {
240         cerr << e.what();
241     } catch (const ErrorElemento& e) {
242         cerr << e.what();
243     }
244 }
245

```

Figura 115: Detalle del módulo mostrarListaPedidosPendientes.

```

245 /**
246 * @brief Permite tramitar un pedido de biblioteca determinado
247 */
248 void Application::mostrarTramitarPedido() {
249     unsigned num;
250     cout
251         << " Introduzca la numeración del pedido de la biblioteca que quiere tramitar: "
252         << endl;
253     cin >> num;
254     num--;
255     try {
256         pedBi = bi.dalistaPedBiblioteca(num);
257         bi.cierraPedidoBiblioteca(pedBi, num);
258     } catch (bad_alloc& ) {
259     } catch (const excepcionesBi::pedidoBibliotecaNoencontrado& e) {
260         cerr << e.what();
261     }
262 }
263
264

```

Figura 116: Detalle del módulo mostrarTramitarPedido.

```

265 /**
266 * @brief Tramita un pedido de un usuario determinado
267 * @param [in] alogin string. Login del usuario.
268 * @param [in] aclave string. Clave de acceso del usuario.
269 */
270 void Application::mostrarTramitarPedidosUsuario(string alogin, string aclave) {
271     unsigned nume_ped_bi;
272     PedidoUsuario * mtln = new PedidoUsuario;
273     cout
274         << " Introduzca los datos del usuario del que quiere tramitar sus pedidos: "
275         << endl;
276     cout << " Introduzca la clave del usuario: " << endl;
277     cin >> aclave;
278     cout << " Introduzca el login: " << endl;
279     cin >> alogin;
280     cout
281         << " Introduzca el número del pedido de la biblioteca a la que quiere dirigir el pedido del usuario: "
282         << endl;
283     cin >> nume_ped_bi;
284     nume_ped_bi--;
285     try {
286         unsigned i = 0;
287         pedBi = bi.dalistaPedBiblioteca(nume_ped_bi); // Obtengo el pedido biblioteca con el número especificado.
288         usu = bi.buscaUsuario(alogin, aclave);
289         pedusu = bi.buscaPedidosUsuarioPendientes(usu);
290         i = 0;
291         min = pedusu->lee(0);
292         while (i < pedusu->tamano()) {
293             if (min->daPrioridad() > pedusu->lee(i)->daPrioridad()) {
294                 min = pedusu->lee(i);
295             }
296             i++;
297         }
298         bi.tramitaPedidoUsuario(min, pedBi);
299     } catch (bad_alloc& ) {
300     } catch (const excepcionesBi::usuNoEncontrado& e) {
301         cerr << e.what();
302     } catch (const excepcionesBi::pedidoBibliotecaNoencontrado& e) {
303         cerr << e.what();
304     } catch (const ErrorElemento& e) {
305         cerr << e.what();
306     }
307 }
308

```

Figura 117: Detalle del módulo mostrarTramitarPedidosUsuario.

```

308 /**
309 * @brief Muestra por pantalla los pedidos pendientes de un usuario determinado
310 * @param [in] alogin string. Login del usuario
311 * @param [in] aclave string. Clave de acceso del usuario
312 */
313
314 void Application::mostrarConsultarPedidosUsuario(string alogin, string aclave) {
315     unsigned i = 0;
316     i = 0;
317     cout
318         << " Introduzca el usuario del que quiere saber sus pedidos pendientes: "
319         << endl;
320     cout << " Introduzca la clave del usuario: " << endl;
321     cin >> aclave;
322     cout << " Introduzca el login: " << endl;
323     cin >> alogin;
324     try {
325         usu = bi.buscaUsuario(alogin, aclave);
326         pedusu = bi.buscaPedidosUsuarioTramitados(usu);
327
328         while (i < pedusu->tamanio()) {
329             cout << *(pedusu->lee(i)) << endl;
330             i++;
331         }
332     } catch (bad_alloc& ) {
333     } catch (const excepcionesBi::usuNoEncontrado& e) {
334         cerr << e.what();
335     } catch (const ErrorElemento& e) {
336         cerr << e.what();
337     }
338 }
339

```

Figura 118: Detalle del módulo mostrarConsultarPedidosUsuario.

```

340 /**
341 * @brief Muestra por pantalla todos los pedidos de biblioteca que han sido tramitados
342 */
343 void Application::mostrarListaPedidosTramitados() {
344     try {
345         unsigned i = 0;
346         i = 0;
347         pedbi = bi.buscaPedidosBibliotecaTramitados();
348         cout
349             << " La lista de pedidos de la biblioteca tramitados es la siguiente: "
350             << endl;
351         while (i < pedbi->tamanio()) {
352             cout << *(pedbi->lee(i)) << endl;
353             i++;
354         }
355     } catch (bad_alloc& ) {
356     } catch (const excepcionesBi::pedidoBibliotecaNoencontrado& e) {
357         cerr << e.what();
358     } catch (const ErrorElemento& e) {
359         cerr << e.what();
360     }
361 }
362

```

Figura 119: Detalle del módulo mostrarListaPedidosTramitados.

```

363 /**
364 * @brief Muestra por pantalla todos los pedidos pendientes de Biblioteca.
365 */
366 void Application::mostrarListaPedidosPendientesBiblioteca() {
367     unsigned i = 0;
368     i = 0;
369     try {
370         pedbi = bi.buscaPedidosBibliotecaPendientes();
371         cout
372             << " La lista de pedidos de la biblioteca pendientes es la siguiente: "
373             << endl;
374         while (i < pedbi->tamanio()) {
375             cout << *(pedbi->lee(i)) << endl;
376             i++;
377         }
378     } catch (bad_alloc& ) {
379     } catch (const excepcionesBi::pedidoBibliotecaNoencontrado& e) {
380         cerr << e.what();
381     } catch (const ErrorElemento& e) {
382         cerr << e.what();
383     }
384 }
385

```

Figura 120: Detalle del módulo mostrarListaPedidosPendientesBiblioteca.

Una vez creados todos los anteriores módulos el estado del módulo a corregir es el que se muestra en la siguiente captura de pantalla.

```

35
34@ /**
35 * @brief Aquí se ha solicitado previamente una clave que, si coincide, da entrada al esquema de admin.
36 * @pre La clave debe ser correcta.
37 */
38@ void Application::aplicacion_admin() {
39     string aclave, alogin, contra = "hola", contrase;
40     cout << " Introduzca contraseña: " << endl;
41     cin >> contrase;
42     if (contrase.compare(contra) == 0) {
43         int opcion;
44
45         do {
46             mostrarMenu();
47             cin >> opcion;
48             switch (opcion) {
49
50             case 1: {
51                 mostrarListaPedidosPendientes(alogin, aclave);
52             }
53             break;
54
55             case 2: {
56                 mostrarTramitarPedido();
57             }
58             break;
59
60             case 3: {
61                 unsigned cpb = 0;
62                 cout << " Se ha creado el pedido de biblioteca num: " << ++cpb
63                 << endl;
64                 pedibipunt = bi.abrePedidoBiblioteca(--cpb);
65                 cpb++;
66             }
67             break;
68
69             case 4: {
70                 mostrarTramitarPedidosUsuario(alogin, aclave);
71             }
72             break;
73
74             case 5: {
75                 mostrarConsultarPedidosUsuario(alogin, aclave);
76             }
77             break;
78
79             case 6: {
80                 mostrarListaPedidosTramitados();
81             }
82             break;
83
84         }
85     }
86 }
```

Figura 121: Detalle del módulo corregido.

```

87         case 7: {
88             mostrarListaPedidosPendientesBiblioteca();
89         }
90     }
91 } while (opcion != 0);
92 } else
93     cout << " Contraseña introducida no valida. " << endl;
94 }
```

Figura 122: Detalle del módulo corregido.

A continuación, realizamos un nuevo análisis de metriculator para comprobar en qué grado hemos disminuido  $V(G)$ . Obtenemos los siguientes resultados.

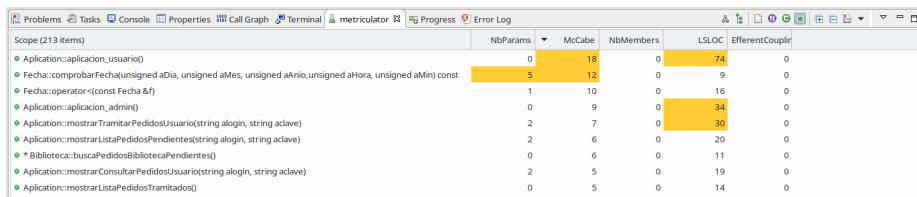


Figura 123: Resultados de metriculator tras las correcciones realizadas.

Podemos comprobar que hemos conseguido obtener un  $V(G)=9$  en la métrica de McCabe, por lo que hemos conseguido cumplir con el objetivo de la corrección.

Para terminar, ejecutamos un nuevo test de cppcheck para comprobar que no arrastremos ningún error o warning en las funciones nuevas. Obtenemos los siguientes resultados.

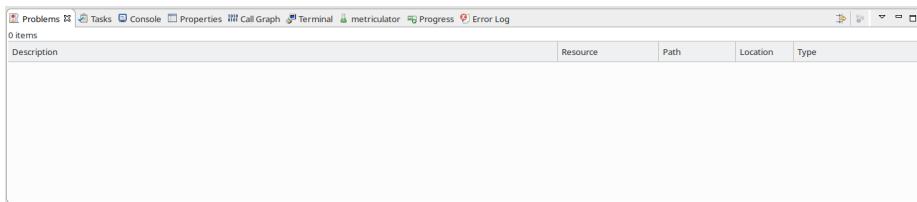


Figura 124: Resultados de cppcheck tras las correcciones realizadas.

Podemos observar que no obtenemos ningún error, warning o aviso; por lo tanto, damos por concluida la corrección del módulo.

### 5.3. Corrección del módulo ”void Application :: aplicacion\_usuario()”

#### 5.3.1. Estado del módulo antes de la corrección

Este módulo se encuentra implementado en la línea 100 del archivo Application.cpp del proyecto. Su complejidad ciclomática es de  $V(G) = 18$  en la métrica de McCabe. En las siguientes capturas de pantalla se puede observar el estado de dicho módulo antes de realizar las correcciones oportunas.

The screenshot shows the Eclipse IDE interface with the Application.cpp file open. The code implements a user application with four options: login, register, change password, and consult a book. It uses a library bl to handle books and users. The metriculator view shows the following metrics for the application:

Scope (213 items)	LLOC	McCabe
Aplicación:aplicacion_usuario()	74	18
Fecha:comprobarFecha(unsigned aDía, unsigned aMes, unsigned aAño)	9	12
Fecha:operator<(const Fecha &d)	16	10
Aplicación:main	54	0

Figura 125: Detalles del módulo a corregir.

The screenshot shows the Eclipse IDE interface with the Application.cpp file open. The code handles user registration and password changes. It includes a try-catch block for memory allocation errors and a loop for consulting books. The metriculator view shows the following metrics for the application:

Scope (213 items)	LLOC	McCabe
Aplicación:aplicacion_usuario()	74	18
Fecha:comprobarFecha(unsigned aDía, unsigned aMes, unsigned aAño)	9	12
Fecha:operator<(const Fecha &d)	16	10
Aplicación:main	54	0

Figura 126: Detalles del módulo a corregir.

The screenshot shows the Eclipse IDE interface with the Application.cpp file open in the editor. The code implements a menu system for a library application. Below the editor, a metrics analysis tool window titled 'metriculator' displays data for three functions:

Scope (213 items)	LLOC	McCabe
Application::mostrarMenuUsuario()	74	18
Fecha::comprobarFecha(unsigned alDia, unsigned aMes, unsigned aAño)	9	12
Fecha::operator<(const Fecha &f)	16	10

Figura 127: Detalles del módulo a corregir.

Como hemos procedido anteriormente, modularizaremos el código que se encuentra en cada uno de los case para reducir la complejidad y la legibilidad del código. Una vez hecho esto, se crearán las siguientes funciones:

- void Application::mostrarMenuUsuario();

```

305 /**
306  * @brief Muestra el menú de interacción de la aplicación del usuario por pantalla
307 */
308 void Application::mostrarMenuUsuario() {
309     cout << "Bienvenido a nuestra Biblioteca.\n\n" << endl << endl;
310     cout << "Elija una opción, para salir pulse 0." << endl;
311     cout << "1. Registrarse en la aplicación. " << endl;
312     cout << "2. Cambiar contraseña. " << endl;
313     cout << "3. Consultar un libro. " << endl;
314     cout << "4. Realizar un pedido. " << endl;
315 }
```

Figura 128: Detalles del módulo mostrarMenuUsuario.

- void Application::registro(string alogin, string aclave, string aclave);

```

317 /**
318 * @brief Registra un usuario en la biblioteca.
319 */
320 void Application::registro(string alogin, string anombre, string aclave) {
321     bool var;
322     cout << " Introduzca un login. " << endl;
323     cin >> alogin;
324     cout << " Introduzca un nombre. " << endl;
325     cin >> anombre;
326     cout << " Introduzca una contraseña. " << endl;
327     cin >> aclave;
328     var = bi.nuevoUsuario(ologin, anombre, aclave);
329     if (var == false)
330         cout << "El login o la clave estan ya en uso. " << endl;
331     else {
332         cout << " Registro correcto. " << endl;
333     }
334 }
```

Figura 129: Detalles del módulo registro.

- void Application::cambiaClave(string alogin, string aclave, string claven);

```

336 /**
337 * @brief Cambia la contraseña de un usuario ofreciendo la anterior que tenia.
338 */
339 void Application::cambiaClave(string alogin, string aclave, string claven) {
340     cout << " Introduzca login actual. " << endl;
341     cin >> alogin;
342     cout << " Introduzca contraseña actual. " << endl;
343     cin >> aclave;
344     try {
345         usu = bi.buscaUsuario(ologin, aclave);
346         cout << " Introduzca clave nueva. " << endl;
347         cin >> claven;
348         usu->cambiarClave(claven);
349     } catch (bad_alloc&) {
350     } catch (const excepcionesBi::usuNoEncontrado& e) {
351         cerr << e.what();
352     }
353 }
```

Figura 130: Detalles del módulo cambiaClave.

- void Application::consultaLibro(string atitulo);

```

355 /**
356 * @brief Hace la consulta de un libro en concreto.
357 */
358 void Application::consultaLibro(string atitulo) {
359     cout << " Introduzca una palabra clave del libro: " << endl;
360     cin >> atitulo;
361     try {
362         libro = bi.consultaLibros(atitulo);
363         for (unsigned i = 0; i < libro->tamanio(); i++) {
364             cout << *(libro->lee(i)) << endl;
365             ;
366             if (i % 5 == 0)
367                 system("pause");
368         }
369     } catch (bad_alloc& ) {
370     } catch (const excepcionesBi::libroNoencontrado& e) {
371         cerr << e.what();
372     } catch (const ErrorElemento& e) {
373         cerr << e.what();
374     }
375 }
```

Figura 131: Detalles del módulo consultaLibro.

- void Application::hazPedido(string aISBN, string atitulo);

```

377 /**
378 * @brief Hace un pedido a la Biblioteca.
379 */
380 void Application::hazPedido(string aISBN, string atitulo) {
381     cout << " Introduzca el ISBN: " << endl;
382     cin >> aISBN;
383     cout << " Introduzca el título: " << endl;
384     cin >> atitulo;
385     try {
386         libro = bi.consultaLibros(atitulo);
387         bi.creaPedidoUsuario(usu, libro->lee(0), 0);
388     } catch (bad_alloc& ) {
389     } catch (const excepcionesBi::libroNoencontrado& e) {
390         cerr << e.what();
391     } catch (const ErrorElemento& e) {
392         cerr << e.what();
393     }
394 }
```

Figura 132: Detalles del módulo hazPedido.

### 5.3.2. Estado del módulo después de la corrección

Una vez creados todos los anteriores módulos el estado del módulo a corregir es el que se muestra en la siguiente captura de pantalla.

```

266 void Application::aplicacion_usuario() {
267     string alogin, aclave, anombre, aISBN, atitulo, claven;
268     try {
269         bi.cargaLibros("../libros.txt");
270     } catch (bad_alloc& ) {
271     } catch (const excepcionesBi::errorApertura& e) {
272         cerr << e.what();
273     }
274     int op;
275     do {
276         mostrarMenuUsuario();
277         cin >> op;
278         switch (op) {
279
280             case 1: {
281                 registro(alogin, anombre, aclave);
282             }
283             break;
284             case 2: {
285                 cambiaClave(alogin, aclave, claven);
286             }
287             break;
288             case 3: {
289                 consultaLibro(atitulo);
290             }
291             break;
292             case 4: {
293                 hazPedido(aISBN, atitulo);
294             }
295             break;
296             default: {
297                 cout << "Has de pulsar una opcion válida o 0 para salir.";
298             }
299             break;
300         }
301     } while (op != 0);

```

Figura 133: Detalles del módulo corregido.

A continuación, realizamos un nuevo análisis de metriculator para comprobar en qué grado hemos disminuido V(G). Obtenemos los siguientes resultados.

Scope (223 items)	LSLOC	McCabe
* Application::mostrarTramitarPedidosUsuario(string alogin, ...)	30	7
* Application::aplicacion_usuario()	27	7
* Application::mostrarListaPedidosPendientes(string alogin, s...	20	6
* Application::consultaLibro(string atitulo)	15	6
* * Biblioteca::buscaPedidosBibliotecaPendientes()	11	6
* Application::mostrarConsultarPedidosUsuario(string alogin...	19	5

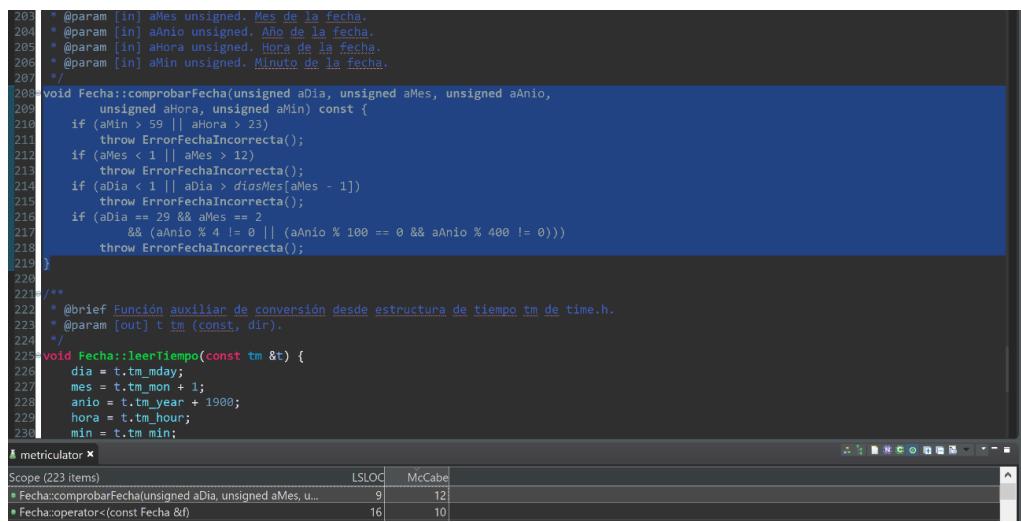
Figura 134: Resultados de metriculator tras las correcciones realizadas.

Con lo cual hemos reducido  $V(g)$  de 18 a 7 en void Application :: aplicacion usuario();

## 5.4. Corrección del módulo ”void Fecha :: comprobarFecha(unsigned aDia, unsigned aMes, unsigned aAnio, unsigned aHora, unsigned aMin) const”

### 5.4.1. Estado del módulo antes de la corrección

Este módulo se encuentra implementado en la línea 208 del archivo Fecha.cpp del proyecto. Su complejidad ciclomática es de  $V(G) = 12$  en la métrica de McCabe. En la siguiente captura de pantalla se puede observar el estado de dicho módulo antes de realizar las correcciones oportunas.



```

203 * @param [in] aMes unsigned. Mes de la fecha.
204 * @param [in] aAnio unsigned. Año de la fecha.
205 * @param [in] aHora unsigned. Hora de la fecha.
206 * @param [in] aMin unsigned. Minuto de la fecha.
207 */
208 void Fecha::comprobarFecha(unsigned aDia, unsigned aMes, unsigned aAnio,
209     unsigned aHora, unsigned aMin) const {
210     if (aMin > 59 || aHora > 23)
211         throw ErrorFechaIncorrecta();
212     if (aMes < 1 || aMes > 12)
213         throw ErrorFechaIncorrecta();
214     if (aDia < 1 || aDia > diasMes[aMes - 1])
215         throw ErrorFechaIncorrecta();
216     if (aDia == 29 && aMes == 2
217         && (aAnio % 4 != 0 || (aAnio % 100 == 0 && aAnio % 400 != 0)))
218         throw ErrorFechaIncorrecta();
219 }
220 /**
221 * @brief Función auxiliar de conversión desde estructura de tiempo tm de time.h.
222 * @param [out] t tm (const, dir).
223 */
224 void Fecha::leerTiempo(const tm &t) {
225     dia = t.tm_mday;
226     mes = t.tm_mon + 1;
227     anio = t.tm_year + 1900;
228     hora = t.tm_hour;
229     min = t.tm_min;
230 }
```

Scope (223 items)	LSLOC	McCabe
Fecha::comprobarFecha(unsigned aDia, unsigned aMes, u...	9	12
Fecha::operator<(const Fecha &f)	16	10

Figura 135: Detalle del módulo a corregir.

La complejidad de este módulo es debido a las numerosas comprobaciones que han de hacerse. Además, la gran cantidad de operadores tales como `||`, `&&`, `<`, `>`, `==`, `!=` aumentan este valor.

Observamos que hace la comprobación de si el año es bisiesto para añadirle un día más a febrero.

Se ha decidido obviar tal comprobación al parecer excesiva para el carácter de la práctica y hemos logrado reducir la  $V(g)$  del módulo de 12 hasta 10 como podemos observar en las siguientes imágenes.

### 5.4.2. Estado del módulo después de la corrección

```

199 /**
200 * @brief Comprobación de la validez de una fecha.
201 * @param [in] aDia unsigned. Día de la fecha.
202 * @param [in] aMes unsigned. Mes de la fecha.
203 * @param [in] aAnio unsigned. Año de la fecha.
204 * @param [in] aHora unsigned. Hora de la fecha.
205 * @param [in] aMin unsigned. Minuto de la fecha.
206 */
207 void Fecha::comprobarFecha(unsigned aDia, unsigned aMes, unsigned aAnio, unsigned aHora, unsigned aMin) const {
208     if (aMin > 59 || aHora > 23)
209         throw ErrorFechaIncorrecta();
210     if (aMes < 1 || aMes > 12)
211         throw ErrorFechaIncorrecta();
212     if (aDia < 1 || aDia > diasMes[aMes - 1])
213         throw ErrorFechaIncorrecta();
214     if (aDia == 29 && aMes == 2 && aAnio % 4 != 0)
215         throw ErrorFechaIncorrecta();
216 }

```

Figura 136: Detalle del módulo después de aplicar las correcciones.

Scope (223 items)	LSLOC	McCabe
Fecha::operator<(const Fecha &f)	16	10
Fecha::comprobarFecha(unsigned aDia, u...	9	10
Application::aplicacion_admin()	35	9
Application::mostrarTramitarPedidosUsuar...	30	7
Application::aplicacion_usuario()	27	7

Figura 137: Resultados de metriculator tras las correcciones realizadas.

### 5.5. Estado del proyecto después de las correcciones

Tras realizar todas las correcciones que se exponen en los apartados anteriores, volvemos a ejecutar los tests de metriculator y cppcheck para comprobar el estado final del proyecto. Los resultados arrojados se exponen a continuación:

Scope (223 items)	NbParams	EfferentCouplir	NbMembers	McCabe	LSLOC
Fecha::operator<(const Fecha &f)	1	0	0	10	16
Fecha::comprobarFecha(unsigned aDia, unsigned aMes, unsigned aAnio, unsigned aHora, unsigned aMin) const	5	0	0	10	9
Application::aplicacion_admin()	0	0	0	9	35
Application::mostrarTramitarPedidosUsuario(string alogin, string clave)	2	0	0	7	30
Application::aplicacion_usuario()	0	0	0	7	27
Application::mostrarListaPedidosPendientes(string alogin, string clave)	2	0	0	6	20
Application::consultarLibro(string atitulo)	1	0	0	6	15
* Biblioteca::buscaPedidosBibliotecaPendientes()	0	0	0	6	11
Application::mostrarConsultarPedidosUsuario(string alogin, string clave)	2	0	0	5	19
Application::mostrarListaPedidosTramitados()	0	0	0	5	14
Application::mostrarListaPedidosPendientesBiblioteca()	0	0	0	5	14
* Biblioteca::buscaPedidosUsuarioPendientes(Usuario *usuario)	1	0	0	5	9

Figura 138: Captura de pantalla de los resultados arrojados tras la ejecución de metriculator.

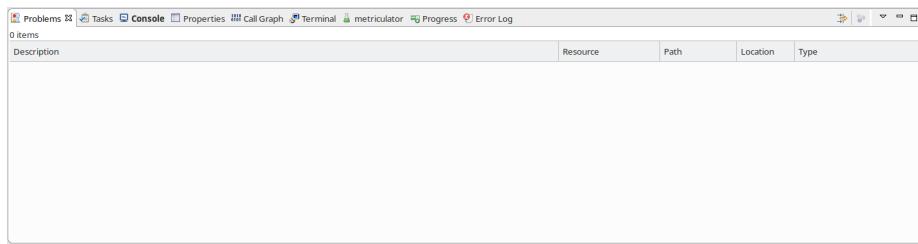


Figura 139: Captura de pantalla de los resultados arrojados tras la ejecución de cppcheck.

Los resultados devueltos por metriculator están ordenados por el valor de McCabe, de mayor a menor, y se puede observar como no existe ningún módulo con un  $V(G) > 10$ .

Tras ejecutar cppcheck, este no nos devuelve ninguna incidencia.

Estos resultados nos permiten considerar como cumplidos satisfactoriamente los objetivos fijados para esta práctica.

## 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/FI051-CPP.+Close+files+when+they+are+no+longer+needed>
- [10] <https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL59-CPP.+Do+not+define+an+unnamed+namespace+in+a+header+file>
- [11] <https://wiki.sei.cmu.edu/confluence/display/cplusplus/FI050-CPP.+Do+not+alternately+input+and+output+from+a+file+stream+without+an+intervening+positioning+call>
- [12] <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions>
- [13] <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM51-CPP.+Properly+deallocate+dynamically+allocated+resources>

[14] <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors>

[15] <http://cppcheck.sourceforge.net/>