
PRÁCTICA 4

Calidad del Software



GRUPO 7

ÍNDICE

Integrantes del grupo:	5
Asignatura para la que se desarrolló:	5
Breve temática del proyecto:	5
EXP53-CPP. No lea la memoria no inicializada	6
FIO51-CPP. Cerrar archivos cuando ya no sean necesarios	9
STR52-CPP. Utilice referencias, punteros e iteradores válidos para hacer referencia a elementos de una cadena básica	13
MSC52-CPP. Las funciones de devolución de valor deben devolver un valor de todas las rutas de salida	15
CTR53-CPP. Utilice rangos de iteradores válidos	17
STR53-CPP. Compruebe el rango del elemento de acceso	20
STR51-CPP. No intente crear un string desde un puntero nulo	22
OOP58-CPP. Las operaciones de copia no deben mutar el objeto origen	24
INT33-C. Asegúrese de que las operaciones de división no resulten en errores de divisiones por cero.	26
CTR50-CPP. Garantizar que los índices de los contenedores e iteradores están en un rango válido	27
Errores que aparecen en la práctica 3	29
Corrección de los problemas	31
Corrección de las advertencias	34
Corrección de los infos	55
Apartado 4: Completar tipos de errores	59
Práctica 4	60

Integrantes del grupo:

Adrián Pérez Ortega apo00015@red.ujaen.es y Juan Bautista Muñoz Ruiz
jbmr0001@red.ujaen.es.

Asignatura para la que se desarrolló:

El proyecto usado se desarrolló para la asignatura de segundo año Estructura de Datos durante el curso 2020-2021.

Breve temática del proyecto:

Se trata de la práctica 3 en la que se pedía implementar la estructura de datos *Árbol AVL* para albergar objetos de tipo *Palabra*.

Además de la creación de la estructura de datos, la finalidad de esta práctica era comprobar una serie de textos mediante la clase *GestorTextos*. Cada *GestorTextos* trabajaba sobre varios documentos y para *Documento* se guardaban en una lista las palabras inexistentes en base a la comprobación de las estructuras de datos de las clases *Diccionario* y *VerbosConjugados*.

Por otro lado, se implementó una prueba de búsquedas, que se encuentra comentada, con las diferentes estructuras de datos.

EXP53-CPP. No lea la memoria no inicializada

Enlace:

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory>

Explicación:

El objetivo de esta regla es garantizar que todas las variables locales declaradas en nuestras funciones, sean inicializadas antes de ser llamadas o utilizadas.

Esto se debe a que, si no se inicializa el objeto, este tomará un valor indeterminado, pudiendo generar un comportamiento indefinido.

Aplicación en el proyecto:

Aquí mostramos la función `insertaFinal()` de la clase `ListaEnlazada`.

```
/**
 * @brief Función para insertar en el final de la Lista Enlazada
 *
 * @param [in] dato Dato de tipo T a insertar
 */
template<typename T>
void ListaEnlazada<T>::insertaFinal(T& dato) {
    Nodo<T> *nuevo = new Nodo<T>(dato, 0);
    if (_cola != 0) {
        _cola->sig = nuevo;
    }
    if (_cabecera == 0) {
        _cabecera = nuevo;
    }
    _cola = nuevo;
    _tam++;
}
```

Ilustración 1: Ejemplo de código bien hecho

Como podemos ver en esta función utilizamos una variable local del tipo `Nodo`, `nuevo`, antes de llamar a esta variable es inicializada correctamente para evitar un comportamiento anómalo de la función.

Un ejemplo de nuestro código que no cumple esta regla, sería la función `revés` de la clase `Palabra`.

```

/**
 * @brief Método para poner al revés la palabra
 *
 * @return Devuelve un objeto de tipo palabra con la palabra al revés
 */
Palabra Palabra::reves() {
    std::string pal;
    Palabra palabra;
    int i = _palabra.length() - 1;
    while (i >= 0) {
        pal += _palabra[i];
        i--;
    }
    palabra.setPalabra(pal);
    return palabra;
}

```

Ilustración 2: Ejemplo de código que no cumple la regla

En ella podemos ver como estamos llamando a la variable `pal`, sin que esta este inicializada correctamente, para solucionar este problema deberemos inicializar la variable a un valor determinado.

```

/**
 * @brief Método para poner al revés la palabra
 *
 * @return Devuelve un objeto de tipo palabra con la palabra al revés
 */
Palabra Palabra::reves() {
    std::string pal = "";
    Palabra palabra;
    int i = _palabra.length() - 1;
    while (i >= 0) {
        pal += _palabra[i];
        i--;
    }
    palabra.setPalabra(pal);
    return palabra;
}

```

Ilustración 3: Corrección de la función `reves()`

Otro ejemplo mal hecho de nuestro código sería la función `getCombinaciones()` de la clase `Palabra`, en esta función tampoco cumplimos con la regla.

```
/**
 * @brief Método para calcular todas las combinaciones con las letras de una palabra
 *
 * @return Devuelve un Vector Dinámico de palabras con todas las posibles combinaciones
 */
VDinamico<Palabra> Palabra::getCombinaciones() { //metodo mas eficiente
    VDinamico<Palabra> vPalabra;
    for (unsigned int i = 0; i < _palabra.length(); i++) { //pasamos cada letra de la palabra al vector
        std::string aux;
        aux.push_back(_palabra[i]); //conversion de char a string
        Palabra letra(aux); //creamos la palabra y la insertamos en el vectordinamico de palabras
        vPalabra.insertar(letra);
    }
    vPalabra.ordenar(); //ordenamos el vector, necesario para los anagramas
    return vPalabra;
}
```

Ilustración 4: Ejemplo de la función `getCombinaciones()` que no cumple la regla

En ella podemos ver como utilizamos la variable `aux`, sin haberla inicializado correctamente. Para solucionarlo lo único que tenemos que hacer es declarar la variable fuera del bucle e inicializarla correctamente.

```
/**
 * @brief Método para calcular todas las combinaciones con las letras de una palabra
 *
 * @return Devuelve un Vector Dinámico de palabras con todas las posibles combinaciones
 */
VDinamico<Palabra> Palabra::getCombinaciones() { //metodo mas eficiente
    VDinamico<Palabra> vPalabra;
    std::string aux = "";
    for (unsigned int i = 0; i < _palabra.length(); i++) { //pasamos cada letra de la palabra al vector
        aux.push_back(_palabra[i]); //conversion de char a string
        Palabra letra(aux); //creamos la palabra y la insertamos en el vectordinamico de palabras
        vPalabra.insertar(letra);
    }
    vPalabra.ordenar(); //ordenamos el vector, necesario para los anagramas
    return vPalabra;
}
```

Ilustración 5: Solución de la función `getCombinaciones()`

FIO51-CPP. Cerrar archivos cuando ya no sean necesarios

Enlace:

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

Explicación:

El objetivo de esta regla es garantizar que, al abrir un archivo, este archivo que es cerrado correctamente antes de la finalización de la función o del programa. Esta regla es importante en el tratamiento de entradas y salidas de flujo de datos ya que el no cerrar bien estos archivos puede provocar que diferentes atacantes intenten agotar los recursos del sistema y aumenta el riesgo de que los datos escritos en los buffers de archivos, no se vacíen correctamente en caso de producirse una finalización anómala del programa.

Aplicación en el proyecto:

```

/**
 * @brief Método para leer el diccionario
 *
 * El diccionario se lee mediante un buffer de lectura de ficheros carga cada palabra en el VDin
 */
void Diccionario::leerDiccionario() {
    std::ifstream fe;
    string linea;
    int total = 0;
    std::string palabra;

    fe.open(getNombreFich()); //abrimos fichero
    if (fe.good()) {
        while (!fe.eof()) { //bucle hasta llegue a la ultima linea
            getline(fe, linea); //leemos la línea
            std::stringstream ss;

            ++total;
            ss << linea;
            getline(ss, palabra); //extraemos la palabra en forma de string de la línea leída
            Palabra pal(palabra); //creamos una Palabra con el string leído
            _terminos.insertar(pal);
        }
        std::cout << "Palabras leídas diccionario: " << total << std::endl;
        std::cout << "Tamaño diccionario: " << _terminos.tam() << std::endl;
        fe.close();
    }
}

```

Ilustración 6: Ejemplo de código que si cumple la regla

Como podemos ver en nuestra función leerDiccionario(), abrimos un flujo de entrada para poder operar con el archivo que le pasamos como parámetro. Una vez que terminamos de leer todo el archivo y llegamos al final del archivo llamamos al método close() para cerrar el archivo correctamente.

Otro ejemplo sería el que realizamos en la función de chequearTexto() de la clase Documento.

```

/**
 * @brief Método para chequear el texto del Documento
 */
void Documento::chequearTexto() {
    _dicc.setNombreFich("dicc-espanol-sin.txt");
    _dicc.leerDiccionario(); //función para cargar la palabras del diccionario
    VerbosConjugados v; //variable auxiliar para asignarle los verbos
    std::fstream fe;
    string palabra;
    fe.open(getNombreFich()); //abrimos flujo de lectura de fichero

    while (fe >> palabra) { //leemos cada palabra en formato de string
        Palabra pal(palabra);
        pal.limpiar(); //la mandamos a limpiarse de signos no alfabéticos solo en caso de leer qu
        v = _dicc.getVerbos();
        if (!_dicc.buscarDicotomica(pal) && !v.buscar(pal)) { //en caso de no encontrar la palab
            addInexistente(pal);
        }
    }
    _inexistentes.borrarRepetidos(); //funcion imlementada para borrar repetidos
    //MostrarInexistentes();
    std::cout << std::endl << "Tamaño inexistentes: " << _inexistentes.tama()
        << std::endl; //Mostramos el tamaño de la lista
    v.mostrarAltura();
    borrarNombresPropios();
}

```

Ilustración 7: Ejemplo de código que no cumple la regla.

Como podemos en este caso cometemos el error de no cerrar correctamente el flujo de entrada, vamos a solucionarlo cerrando correctamente el flujo de entrada una vez que se ha terminado de operar con el archivo.


```

//
void Documento::chequearTexto() {
    _dicc.setNombreFich("dicc-espanol-sin.txt");
    _dicc.leerDiccionario(); //función para cargar la palabras del diccionario
    VerbosConjugados v; //variable auxiliar para asignarle los verbos
    std::fstream fe;
    string palabra;
    fe.open(getNombreFich()); //abrimos flujo de lectura de fichero

    while (fe >> palabra) { //leemos cada palabra en formato de string
        Palabra pal(palabra);
        pal.limpiar(); //la mandamos a limpiarse de signos no alfabéticos solo en caso de leer
        v = _dicc.getVerbos();
        if (!_dicc.buscarDicotomica(pal) && !v.buscar(pal)) { //en caso de no encontrar la palabra
            addInexistente(pal);
        }
    }
    // Cerramos el archivo
    fe.close();
    _inexistentes.borrarRepetidos(); //funcion implementada para borrar repetidos
    //MostrarInexistentes();
    std::cout << std::endl << "Tamaño inexistentes: " << _inexistentes.tama()
        << std::endl; //Mostramos el tamaño de la lista
    v.mostrarAltura();
    borrarNombresPropios();
}

```

Ilustración 8: Solución de la función chequearTexto().

Por último, otro ejemplo de nuestro proyecto en el que cumplimos la regla sería el que realizamos en la función leerVerbos() de la clase VerbosConjugados.

```

/**
 * @brief Método para leer el fichero de verbos e insertarlo en el AVL
 */
void VerbosConjugados::leerVerbos() {
    std::ifstream fe;
    string linea;
    int total = 0;
    std::string palabra;

    fe.open(getNombreFich()); //abrimos fichero
    if (fe.good()) {
        while (!fe.eof()) { //bucle hasta llegue a la ultima linea
            getline(fe, linea); //leemos la linea
            std::stringstream ss;

            ++total;
            ss << linea;
            getline(ss, palabra); //extraemos la palabra en forma de string de la linea leida
            Palabra pal(palabra); //creamos una Palabra con el string leido
            _vconjugados.insertar(pal);
        }

        fe.close();
    }
}

```

Ilustración 9: Ejemplo de código bien hecho.

Como podemos apreciar se cierra correctamente el recurso abierto una vez que se ha terminado de operar con él.

STR52-CPP. Utilice referencias, punteros e iteradores válidos para hacer referencia a elementos de una cadena básica

Enlace:

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR52-CPP.+Use+valid+references%2C+pointers%2C+and+iterators+to+reference+elements+of+a+basic+string>

Explicación:

El objetivo de esta regla es hacernos entender que al utilizar variables del tipo string que están inicializadas no debemos modificarlas en el caso que estemos iterando sobre ellas, ya que en ese caso el comportamiento de las llamadas al iterador puede no estar definidas, es por ello que deberíamos utilizar variables locales en las que almacenemos los cambios e iteremos sobre la variable inicializada sin modificarla.

Aplicación en el proyecto:

Para ejemplificar esta regla se ha decidido sobrescribir una función ya creada anteriormente.

```
/**
 * @brief Método para poner al reves una palabra pasada como parametro y asignarla
 *
 * @param pal Palabra que queremos poner como al revés
 * @return Devuelve el resultado de la palabra
 */
void Palabra::reves(std::string pal) {
    std::string aux = pal;
    int i = pal.length() - 1;
    while (i >= 0) {
        aux += pal[i];
        i--;
    }
    this->setPalabra(pal);
}
```

Ilustración 10: Ejemplo de código bien hecho.

Como podemos ver se pasa como parámetro una cadena, como el objetivo es alterar el orden de la palabra, no podemos trabajar directamente con ella, es por eso que declaramos una variable auxiliar que será la cadena que obtenga el resultado final. Por tanto, cada vez que llamamos a la cadena 'pal' esta se mantiene íntegra y no cambia en ningún momento durante la ejecución de la función.

MSC52-CPP. Las funciones de devolución de valor deben devolver un valor de todas las rutas de salida

Enlace:

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/MS52-CPP.+Value-returning+functions+must+return+a+value+from+all+exit+paths>

Explicación:

El objetivo de esta regla es asegurarnos de que todas las funciones declaradas en nuestro código que devuelvan algún tipo de valor, devuelven algún valor siempre en cualquier situación y en todas las rutas posibles de las funciones. Esta es una regla importante ya que en el caso de que una función que debería de devolver algún tipo de valor no devuelva nada, podría generar un comportamiento indefinido.

Aplicación en el proyecto:

Aquí mostramos un ejemplo de cómo se accede siempre a un return en la función de anagrama(Palabra pal) de la clase Palabra. En concreto esta función devolverá true si la palabra que pasamos como parámetro es un anagrama o false en caso contrario.

```
/**
 * @brief Método que devuelve si dos palabras son anagramas
 *
 * @param [in] pal Palabra con la que comparar
 * @return Boolean que indica si son anagramas
 */
bool Palabra::anagrama(Palabra pal) { //comparamos dos vectores de letras ordenados alfabéticamente
    VDinamico<Palabra> aux1 = this->getCombinaciones(); //se obtienen los vectores de palabras ordenados
    VDinamico<Palabra> aux2 = pal.getCombinaciones();
    bool anagrama = true;
    if (aux1.tam() == aux2.tam()) { //se comparan las dos palabras comparando cada posición del vector ordenado
        for (unsigned int i = 0; i < this->getCombinaciones().tam(); i++) {
            if (aux1[i]._palabra != aux2[i]._palabra) {
                return false; //si todas las palabras de cada posición son iguales-> anagrama
            }
        }
    } else {
        return false;
    }
    return anagrama;
}
```

Ilustración 11: Ejemplo de código bien hecho.

Como podemos ver, en caso de que alguna condición falle se devuelve false, pero en el caso que todo se cumpla al final de la función se realizará un return, devolviendo un true.

Otro ejemplo que podemos mostrar es el de la función palindromo(Palabra pal) de la clase Palabra.

```
/**
 * @brief Método para saber si la palabra es palíndromo
 *
 * @param [in] pal Palabra a comprobar
 * @return Devuelve un boolean que indica si la palabra es palíndroma
 */
bool Palabra::palindromo(Palabra pal) {
    if (_palabra == pal.reves()._palabra) {
        return true;
    }
    return false;
}
```

Ilustración 12: Ejemplo de código bien hecho.

Como se puede apreciar, siempre se va a devolver un valor.

Por último mostraremos un ejemplo de la función buscar(Palabra pal) de la clase VerbosConjugados en la que se puede apreciar que también se respeta esta regla.

```
/**
 * @brief Función para buscar una palabra en los verbos conjugados
 *
 * @param [in] pal Objeto de tipo palabra a buscar
 * @return Devuelve un boolean que indica si la palabra ha sido encontrada
 */
bool VerbosConjugados::buscar(Palabra pal) {
    Palabra result; //palabra con el resultado encontrado
    if (_vconjugados.buscaIt(pal, result)) {
        return true;
    }
    return false;
}
```

Ilustración 13: Ejemplo de código bien hecho.

CTR53-CPP. Utilice rangos de iteradores válidos

Enlace:

[https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR53-CPP.+Use+valid+iterato
r+ranges](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR53-CPP.+Use+valid+iterato+r+ranges)

Explicación:

El objetivo de esta regla, es hacernos entender que al utilizar iteradores sobre diferentes contenedores, debemos asegurarnos se utilizar un rango válido, para cumplir esto debemos asegurarnos que en este rango, los iteradores hacen referencia al mismo contenedor y en el iterador la cabecera precede a la cola. Importante cumplir estas condiciones ya que si no se cumplen se puede dar un comportamiento indefinido de la función y provocar un desbordamiento del buffer, lo que puede provocar que un atacante ejecute código arbitrariamente.

Aplicación en el proyecto:

Mostraremos unos ejemplos en los que utilizamos iteradores para navegar y manejar los contenedores.

```

/**
 * @brief Función para borrar una posición iterada
 *
 * @param [in] i Iterador a la posición a borrar
 */
template<typename T>
void ListaEnlazada<T>::borrar(Iterador<T>& i) {
    Nodo<T>* p = i.nodo;
    if (!i.fin()) {
        if (i.nodo == _cola) {
            borraFinal();
        } else if (i.nodo == _cabecera) {
            borraInicio();
        } else {
            Nodo<T> *anterior = 0;
            if (_cabecera != _cola) {
                anterior = _cabecera;
                while (anterior->sig != p) {
                    anterior = anterior->sig;
                }
            }
            anterior->sig = p->sig;
            _tam--;
        }
    }
}

```

Ilustración 14: Ejemplo código bien hecho.

Como podemos ver en esta función de la clase ListaEnlazada, recorreremos la lista para borrar un nodo dado un iterador, para ello antes deberemos de modificar los punteros de anterior del nodo siguiente. Para cumplir la regla sabemos que el iterador *i* y la cola hacen referencia al mismo contenedor. Además, a la hora de realizar comprobaciones lo que hacemos es ver si la cola hace referencia a la misma posición de memoria que la cabecera.

Otro ejemplo sería el de la función inserta de la clase ListaEnlazada

```

/**
 * @brief Función para insertar en medio de la Lista Enlazada
 *
 * @param [in] i Iterador a la posición en la que insertar
 * @param [in] dato Dato de tipo T a insertar
 */
template<typename T>
void ListaEnlazada<T>::inserta(Iterador<T>& i, T& dato) {
    if (!i.fin()) {
        if (i.nodo == _cabecera) {
            insertaInicio(dato);
        } else if (i.nodo == _cola) {
            insertaFinal(dato);
        } else {
            Nodo<T> *anterior = 0;
            anterior = _cabecera;
            while (anterior->sig != i.nodo) {
                anterior = anterior->sig;
            }
            Nodo<T> *nuevo = new Nodo<T>(dato, i.nodo);
            anterior->sig = nuevo;
            tama++;
        }
    }
}

```

Ilustración 15: Ejemplo código bien hecho.

Como podemos ver sucede igual que en el anterior ejemplo, utilizamos un rango de iteradores válidos, ya que el iterador que se pasa como parámetro deberá de hacer referencia al contenedor de la lista enlazada, en caso contrario no se iterará sobre el contenedor.

STR53-CPP. Compruebe el rango del elemento de acceso

Enlace:

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access>

Explicación:

El objetivo de esta regla es evitar errores al llamar a una posición del string que esté fuera de rango. Es decir, la posición de la cadena de caracteres a la que se intenta acceder es mayor que el número de caracteres. Esto se consigue evitar con un try/catch.

Aplicación en el proyecto:

```
/**
 * @brief Método para convertir en minúscula la primera letra de la palabra
 */
void Palabra::convertirPrimeraLetraMinuscula() {
    char car[_palabra.length()]; //array de char de la misma longitud que la palabra
    strcpy(car, _palabra.c_str()); //conversión de palabra a array de char

    car[0] = tolower(car[0]); //convertimos a minúscula

    string s = ""; //string auxiliar onde se irá guardando la palabra
    for (unsigned int i = 0; i < _palabra.length(); i++) {
        s = s + car[i];
    }
    this->_palabra = s;
}
```

Ilustración 16: Ejemplo código que no cumple el estándar.

Como podemos ver en la función `convertirPrimeraLetraMinuscula()` de la clase `Palabra` en la llamada de la posición cero del string al no realizar una comprobación de rango en caso de que la longitud de la cadena de caracteres sea cero provocaría un error, ya que estamos realizando una llamada a la primera posición.

A continuación encontramos la solución al problema por medio del uso de este estándar:

```

/**
 * @brief Método para convertir en minúscula la primera letra de la palabra
 */
void Palabra::convertirPrimeraLetraMinuscula() {
    char car[_palabra.length()]; //array de char de la misma longitud que la palabra
    strcpy(car, _palabra.c_str()); //conversión de palabra a array de char

    try{
        car[0] = tolower(car[0]); //convertimos a minúscula
    }catch (const std::out_of_range& oor) {
        std::cerr << "Out of Range error: " << oor.what() << '\n';
    }

    string s = ""; //string auxiliar onde se irá guardando la palabra
    for (unsigned int i = 0; i < _palabra.length(); i++) {
        s = s + car[i];
    }
    this->_palabra = s;
}

```

Ilustración 17: Ejemplo código bien hecho.

Se ha incorporado un try catch capturando una excepción de fuera de rango.

Por otro lado, en la función primeraLetraEsMayus() de la clase Palabra encontramos el mismo error.

```

/**
 * @brief Método para comprobar si la primera letra es mayúscula
 *
 * @return Boolean que indica si la primera letra es mayúscula
 */
bool Palabra::primeraLetraEsMayus() {
    char car[_palabra.length()];
    strcpy(car, _palabra.c_str());
    bool m = false;

    if (isupper(car[0])) {
        m = true;
    }

    return m;
}

```

Ilustración 18: Ejemplo código que no cumple la regla.

En esta foto podemos ver como se ha solucionado para el cumplimiento del estándar con el mismo try/catch que en el ejemplo anterior:

```
/**
 * @brief Método para comprobar si la primera letra es mayúscula
 *
 * @return Boolean que indica si la primera letra es mayúscula
 */
bool Palabra::primeraLetraEsMayus() {
    char car[_palabra.length()];
    strcpy(car, _palabra.c_str());
    bool m = false;

    try{
        if (isupper(car[0])) {
            m = true;
        }
    } catch (const std::out_of_range& oor) {
        std::cerr << "Out of Range error: " << oor.what() << '\n';
    }

    return m;
}
```

Ilustración 19: Ejemplo código bien hecho.

STR51-CPP. No intente crear un string desde un puntero nulo

Enlace:

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std%3A%3Astring+from+a+null+pointer>

Explicación:

Esta regla nos aconseja la no creación de una cadena de caracteres vacía o nula como parámetro ya que puede inducir a error. Por lo que es necesaria una comprobación de la cadena invocada antes de trabajar sobre ella.

Aplicación en el proyecto:

En esta imagen de la función `revés(String palabra)` de la clase `Palabra` podemos ver que esta comprobación no se hace:

```
/**
 * @brief Método para poner al revés una palabra pasada como parametro y asignarla
 *
 * @param pal Palabra que queremos poner como al revés
 * @return Devuelve el resultado de la palabra
 */
void Palabra::revés(std::string pal) {
    std::string aux = pal;
    int i = pal.length() - 1;
    while (i >= 0) {
        aux += pal[i];
        i--;
    }
    this->setPalabra(pal);
}
```

Ilustración 20: Ejemplo de código que no cumple la regla.

Tras modificar el código introduciendo la comprobación pedida por el estándar queda así:

```
/**
 * @brief Método para poner al revés una palabra pasada como parametro y asignarla
 *
 * @param pal Palabra que queremos poner como al revés
 * @return Devuelve el resultado de la palabra
 */
void Palabra::revés(std::string pal) {
    if(pal!=""){
        std::string aux = pal;
        int i = pal.length() - 1;
        while (i >= 0) {
            aux += pal[i];
            i--;
        }
        this->setPalabra(pal);
    }
}
```

Ilustración 21: Ejemplo de código que cumple la regla.

En la función `palíndromo()` de la clase `palabra` encontramos el mismo error. No se comprueba el caso de que la cadena recibida esté vacía:

```
/**
 * @brief Método para saber si la palabra es palíndromo
 *
 * @param [in] pal Palabra a comprobar
 * @return Devuelve un boolean que indica si la palabra es palíndroma
 */
bool Palabra::palindromo(Palabra pal) {
    if (_palabra == pal.reves()._palabra) {
        return true;
    }
    return false;
}
```

Ilustración 22: Ejemplo de código que no cumple la regla.

Como podemos ver se ha solucionado incluyendo, de la misma forma, la comprobación pedida por el estándar:

```
/**
 * @brief Método para saber si la palabra es palíndromo
 *
 * @param [in] pal Palabra a comprobar
 * @return Devuelve un boolean que indica si la palabra es palíndroma
 */
bool Palabra::palindromo(Palabra pal) {
    if (pal.getPalabra() != "") {
        if (_palabra == pal.reves()._palabra) {
            return true;
        }
    }
    return false;
}
```

Ilustración 23: Ejemplo de código que cumple la regla.

OOP58-CPP. Las operaciones de copia no deben mutar el objeto origen

Enlace:

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP58-CPP.+Copy+operations+must+not+mutate+the+source+object>

Explicación:

Este estándar nos habla de la mutación del objeto origen en un constructor copia. El objeto origen ha de permanecer inmutable ya que si realizamos una modificación sobre este las sucesivas copias siguientes serán diferentes del original.

Aplicación en el proyecto:

En nuestro proyecto todos los constructores copia cumplen este estándar por los que a continuación se adjuntan imágenes de algunos de estos:

En el caso del constructor copia del vector Dinámico en ningún momento se muta el vector origen:

```
/**
 * @brief Constructor copia de la clase VDinamico
 *
 * @param [in] origen VDinamico a copiar
 */
template<typename T>
VDinamico<T>::VDinamico(const VDinamico<T>& origen) :
    _tamal(origen._tamal), _tamaf(origen._tamaf)
{
    _v = new T[_tamaf];
    for (unsigned long int i = 0; i < origen._tamal; i++) {
        _v[i] = origen._v[i];
    }
}
```

Ilustración 24: Ejemplo de código que cumple el estándar.

Lo mismo ocurre con el constructor copia de la Lista Enlazada, tampoco modificamos la Lista Enlazada origen:

```

/**
 * @brief Constructor copia de la clase Lista Enlazada
 *
 * @param origen [in] Lista Enlazada que queremos copiar
 */
template<typename T>
ListaEnlazada<T>::ListaEnlazada(const ListaEnlazada<T>& origen) {
    Nodo<T> *i = origen._cabecera;
    _tam = origen._tam;
    _cabecera = _cola = 0;
    while (i) {
        Nodo<T> *nuevo = new Nodo<T>(i->dato, 0);
        if (_cola != 0) {
            _cola->sig = nuevo;
        }
        if (_cabecera == 0) {
            _cabecera = nuevo;
        }
        _cola = nuevo;
        i = i->sig;
    }
}

```

Ilustración 25: Ejemplo de código que cumple la regla.

INT33-C. Asegúrese de que las operaciones de división no resulten en errores de divisiones por cero.

Enlace:

<https://wiki.sei.cmu.edu/confluence/display/c/INT33-C.+Ensure+that+division+and+remainder+operations+do+not+result+in+divide-by-zero+errors>

Explicación:

Este estándar, si bien es cierto proviene de C, también lo encontramos aplicable en C++ según el CERT. El desbordamiento ocurre cuando el divisor de una operación aritmética de división es cero. Además el desbordamiento puede ocurrir durante la división de enteros con signo en complemento a dos cuando el dividendo es igual al valor mínimo (más negativo) para el tipo de entero con signo y el divisor es igual a -1.

Aplicación en el proyecto:

```

/**
 * @brief Método para reducir el tamaño físico el vector dinámico
 */
template<typename T>
void VDinamico<T>::disminuye() {
    _tamaf = _tamaf / 2;
    T *vaux = new T[_tamaf];
    for (int i = 0; i < _tamaf; i++) {
        vaux[i] = _v[i];
    }
    delete[] _v;
    _v = vaux;
}

```

Ilustración 26: Ejemplo de código que cumple la regla.

Como nuestro divisor de la función disminuye() de la clase Vector Dinámico permanece estático a 2 no ocurrirá nunca este desbordamiento. En caso de que el divisor fuera variable habría que incluir una comprobación de la forma:

```

signed long result;
if ((s_b == 0) || ((s_a == LONG_MIN) && (s_b == -1))) {
    /* Handle error */
} else {
    result = s_a / s_b;
}

```

Ilustración 27: Ejemplo de código a implementar en un caso con divisor variable.

CTR50-CPP. Garantizar que los índices de los contenedores e iteradores están en un rango válido

Enlace:

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR50-CPP.+Guarantee+that+container+indices+and+iterators+are+within+the+valid+range>

Explicación:

Esta regla nos induce que el programador es el responsable de garantizar que los índices se encuentren dentro de los límites del vector. Es decir, no se intenten acceder a posiciones inferiores al índice mínimo o superiores al índice máximo.

Aplicación en el proyecto:

Como podemos ver en el operador [] de la clase Vector Dinámico no se cumple el estándar por lo que si intentamos acceder a una posición mayor que el tamaño máximo se producirá un error que no hemos capturado:

```
/**
 * @brief Operador de acceso a posiciones de la clase VDinamico
 *
 * @param [in] pos Entero con la posición a acceder
 * @return Devuelve un objeto de tipo T con el dato accedido
 */
template<typename T>
T& VDinamico<T>::operator[](unsigned long int pos) {
    return _v[pos];
}
```

Ilustración 28: Ejemplo de código que no cumple el estándar.

A continuación podemos ver la solución, aplicando una simple comprobación con el tamaño lógico del vector:

```
/**
 * @brief Operador de acceso a posiciones de la clase VDinamico
 *
 * @param [in] pos Entero con la posición a acceder
 * @return Devuelve un objeto de tipo T con el dato accedido
 */
template<typename T>
T& VDinamico<T>::operator[](unsigned long int pos) {
    if(pos >= tamal){
        throw ("Index out of bounds");
    }
    return _v[pos];
}
```

Ilustración 29: Ejemplo de código que cumple el estándar.

En la función borrado vemos que se cumple el estándar con la comprobación del índice con el tamaño lógico justo nada más entrar a la función por lo que no ha sido necesario modificar código:

```

* @param [in] pos Entero con la posición a borrar
* @return Devuelve un Objeto de tipo T con el elemento borrado
*/
template<typename T>
T VDinamico<T>::borrar(unsigned long int pos) {
    T borrado;
    if (_tamal == 0 || pos >= _tamal) { //en caso de posición fuera de rango o vector no tenga elementos
        throw string("No se puede borrar");
    }
    if (pos == UINT_MAX) { //borrado última posición
        borrado = _v[_tamal - 1];
    } else {
        borrado = _v[pos]; //borrado entre medias
        for (int i = pos; i < _tamal - 1; i++) {
            _v[i] = _v[i + 1];
        }
    }
    if (_tamal * 3 < _tamaf) { //disminución de tamaño fijo-sico
        disminuye();
    }
    _tamal--;
    return borrado;
}

```

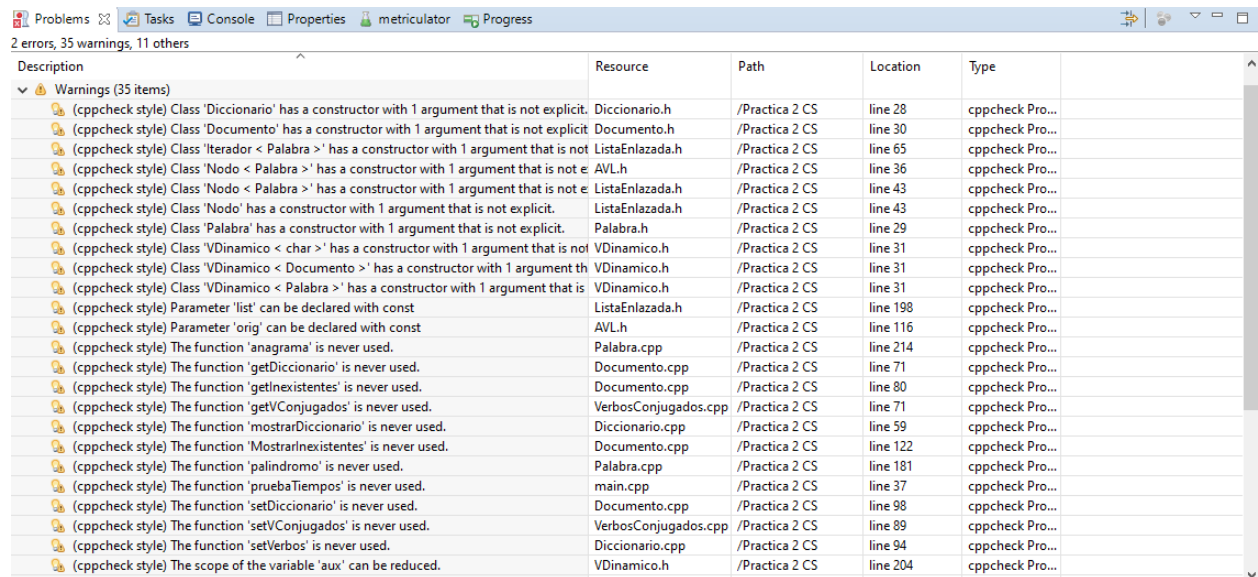
Ilustración 30: Ejemplo de código que cumple el estándar.

Errores que aparecen en la práctica 3

Problems Tasks Console Properties metriculator Progress					
2 errors, 35 warnings, 11 others					
Description	Resource	Path	Location	Type	
Errors (2 items)					
(cppcheck error) The one definition rule is violated, different classes/structs have the same name	AVL.h	/Practica 2 CS	line 24	cppcheck Pro...	
(cppcheck warning) Possible null pointer dereference: anterior	ListaEnlazada.h	/Practica 2 CS	line 344	cppcheck Pro...	
Warnings (35 items)					
Infos (11 items)					

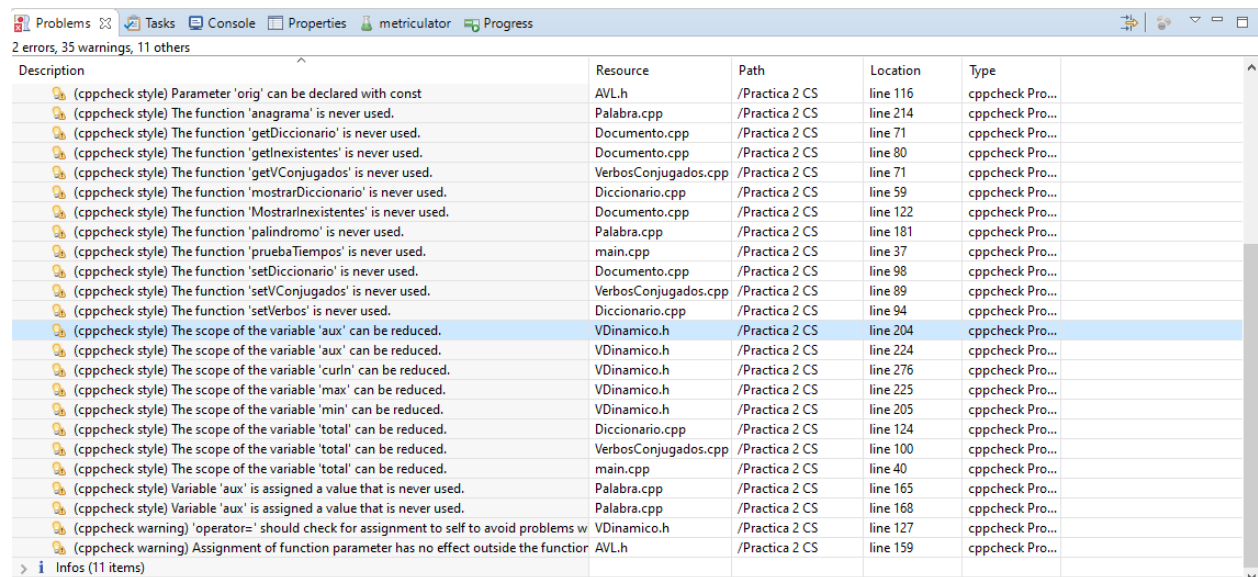
Ilustración 31: Errores al ejecutar cppcheck

Práctica 1 | Calidad del Software



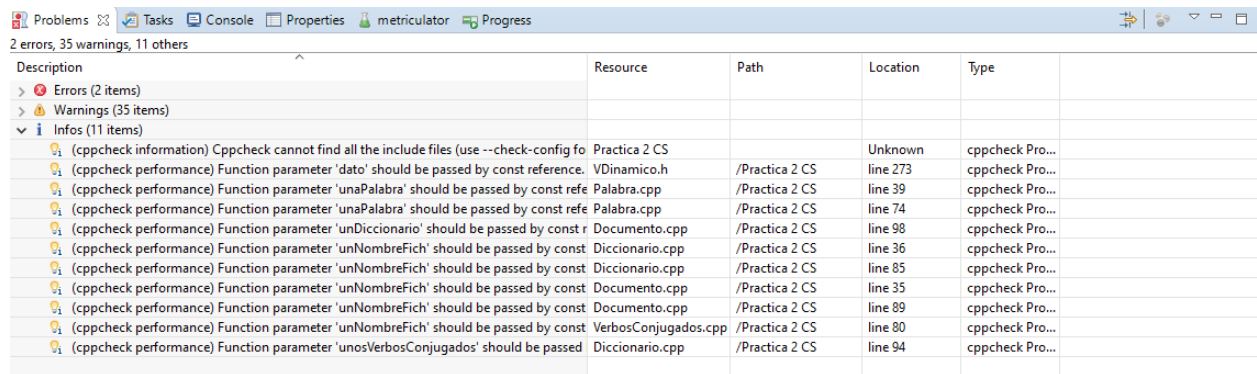
Description	Resource	Path	Location	Type
Warnings (35 items)				
(cppcheck style) Class 'Diccionario' has a constructor with 1 argument that is not explicit.	Diccionario.h	/Practica 2 CS	line 28	cppcheck Pro...
(cppcheck style) Class 'Documento' has a constructor with 1 argument that is not explicit.	Documento.h	/Practica 2 CS	line 30	cppcheck Pro...
(cppcheck style) Class 'Iterador < Palabra >' has a constructor with 1 argument that is not explicit.	ListaEnlazada.h	/Practica 2 CS	line 65	cppcheck Pro...
(cppcheck style) Class 'Nodo < Palabra >' has a constructor with 1 argument that is not explicit.	AVL.h	/Practica 2 CS	line 36	cppcheck Pro...
(cppcheck style) Class 'Nodo < Palabra >' has a constructor with 1 argument that is not explicit.	ListaEnlazada.h	/Practica 2 CS	line 43	cppcheck Pro...
(cppcheck style) Class 'Nodo' has a constructor with 1 argument that is not explicit.	ListaEnlazada.h	/Practica 2 CS	line 43	cppcheck Pro...
(cppcheck style) Class 'Palabra' has a constructor with 1 argument that is not explicit.	Palabra.h	/Practica 2 CS	line 29	cppcheck Pro...
(cppcheck style) Class 'VDinamico < char >' has a constructor with 1 argument that is not explicit.	VDinamico.h	/Practica 2 CS	line 31	cppcheck Pro...
(cppcheck style) Class 'VDinamico < Documento >' has a constructor with 1 argument that is not explicit.	VDinamico.h	/Practica 2 CS	line 31	cppcheck Pro...
(cppcheck style) Class 'VDinamico < Palabra >' has a constructor with 1 argument that is not explicit.	VDinamico.h	/Practica 2 CS	line 31	cppcheck Pro...
(cppcheck style) Parameter 'list' can be declared with const.	ListaEnlazada.h	/Practica 2 CS	line 198	cppcheck Pro...
(cppcheck style) Parameter 'orig' can be declared with const.	AVL.h	/Practica 2 CS	line 116	cppcheck Pro...
(cppcheck style) The function 'anagrama' is never used.	Palabra.cpp	/Practica 2 CS	line 214	cppcheck Pro...
(cppcheck style) The function 'getDiccionario' is never used.	Documento.cpp	/Practica 2 CS	line 71	cppcheck Pro...
(cppcheck style) The function 'getInexistentes' is never used.	Documento.cpp	/Practica 2 CS	line 80	cppcheck Pro...
(cppcheck style) The function 'getVConjugados' is never used.	VerbosConjugados.cpp	/Practica 2 CS	line 71	cppcheck Pro...
(cppcheck style) The function 'mostrarDiccionario' is never used.	Diccionario.cpp	/Practica 2 CS	line 59	cppcheck Pro...
(cppcheck style) The function 'MostrarInexistentes' is never used.	Documento.cpp	/Practica 2 CS	line 122	cppcheck Pro...
(cppcheck style) The function 'palindromo' is never used.	Palabra.cpp	/Practica 2 CS	line 181	cppcheck Pro...
(cppcheck style) The function 'pruebaTiempos' is never used.	main.cpp	/Practica 2 CS	line 37	cppcheck Pro...
(cppcheck style) The function 'setDiccionario' is never used.	Documento.cpp	/Practica 2 CS	line 98	cppcheck Pro...
(cppcheck style) The function 'setVConjugados' is never used.	VerbosConjugados.cpp	/Practica 2 CS	line 89	cppcheck Pro...
(cppcheck style) The function 'setVerbos' is never used.	Diccionario.cpp	/Practica 2 CS	line 94	cppcheck Pro...
(cppcheck style) The scope of the variable 'aux' can be reduced.	VDinamico.h	/Practica 2 CS	line 204	cppcheck Pro...

Ilustración 32: Warnings al ejecutar cppcheck



Description	Resource	Path	Location	Type
(cppcheck style) Parameter 'orig' can be declared with const.	AVL.h	/Practica 2 CS	line 116	cppcheck Pro...
(cppcheck style) The function 'anagrama' is never used.	Palabra.cpp	/Practica 2 CS	line 214	cppcheck Pro...
(cppcheck style) The function 'getDiccionario' is never used.	Documento.cpp	/Practica 2 CS	line 71	cppcheck Pro...
(cppcheck style) The function 'getInexistentes' is never used.	Documento.cpp	/Practica 2 CS	line 80	cppcheck Pro...
(cppcheck style) The function 'getVConjugados' is never used.	VerbosConjugados.cpp	/Practica 2 CS	line 71	cppcheck Pro...
(cppcheck style) The function 'mostrarDiccionario' is never used.	Diccionario.cpp	/Practica 2 CS	line 59	cppcheck Pro...
(cppcheck style) The function 'MostrarInexistentes' is never used.	Documento.cpp	/Practica 2 CS	line 122	cppcheck Pro...
(cppcheck style) The function 'palindromo' is never used.	Palabra.cpp	/Practica 2 CS	line 181	cppcheck Pro...
(cppcheck style) The function 'pruebaTiempos' is never used.	main.cpp	/Practica 2 CS	line 37	cppcheck Pro...
(cppcheck style) The function 'setDiccionario' is never used.	Documento.cpp	/Practica 2 CS	line 98	cppcheck Pro...
(cppcheck style) The function 'setVConjugados' is never used.	VerbosConjugados.cpp	/Practica 2 CS	line 89	cppcheck Pro...
(cppcheck style) The function 'setVerbos' is never used.	Diccionario.cpp	/Practica 2 CS	line 94	cppcheck Pro...
(cppcheck style) The scope of the variable 'aux' can be reduced.	VDinamico.h	/Practica 2 CS	line 204	cppcheck Pro...
(cppcheck style) The scope of the variable 'aux' can be reduced.	VDinamico.h	/Practica 2 CS	line 224	cppcheck Pro...
(cppcheck style) The scope of the variable 'curln' can be reduced.	VDinamico.h	/Practica 2 CS	line 276	cppcheck Pro...
(cppcheck style) The scope of the variable 'max' can be reduced.	VDinamico.h	/Practica 2 CS	line 225	cppcheck Pro...
(cppcheck style) The scope of the variable 'min' can be reduced.	VDinamico.h	/Practica 2 CS	line 205	cppcheck Pro...
(cppcheck style) The scope of the variable 'total' can be reduced.	Diccionario.cpp	/Practica 2 CS	line 124	cppcheck Pro...
(cppcheck style) The scope of the variable 'total' can be reduced.	VerbosConjugados.cpp	/Practica 2 CS	line 100	cppcheck Pro...
(cppcheck style) The scope of the variable 'total' can be reduced.	main.cpp	/Practica 2 CS	line 40	cppcheck Pro...
(cppcheck style) Variable 'aux' is assigned a value that is never used.	Palabra.cpp	/Practica 2 CS	line 165	cppcheck Pro...
(cppcheck style) Variable 'aux' is assigned a value that is never used.	Palabra.cpp	/Practica 2 CS	line 168	cppcheck Pro...
(cppcheck warning) 'operator=' should check for assignment to self to avoid problems w	VDinamico.h	/Practica 2 CS	line 127	cppcheck Pro...
(cppcheck warning) Assignment of function parameter has no effect outside the function	AVL.h	/Practica 2 CS	line 159	cppcheck Pro...

Ilustración 33: Warnings al ejecutar cppcheck



Description	Resource	Path	Location	Type
> 2 errors, 35 warnings, 11 others				
> Errors (2 items)				
> Warnings (35 items)				
▼ Infos (11 items)				
❗ (cppcheck information) Cppcheck cannot find all the include files (use --check-config fo	Practica 2 CS		Unknown	cppcheck Pro...
❗ (cppcheck performance) Function parameter 'dato' should be passed by const reference.	VDinamico.h	/Practica 2 CS	line 273	cppcheck Pro...
❗ (cppcheck performance) Function parameter 'unaPalabra' should be passed by const refe	Palabra.cpp	/Practica 2 CS	line 39	cppcheck Pro...
❗ (cppcheck performance) Function parameter 'unaPalabra' should be passed by const refe	Palabra.cpp	/Practica 2 CS	line 74	cppcheck Pro...
❗ (cppcheck performance) Function parameter 'unDiccionario' should be passed by const r	Documento.cpp	/Practica 2 CS	line 98	cppcheck Pro...
❗ (cppcheck performance) Function parameter 'unNombreFich' should be passed by const	Diccionario.cpp	/Practica 2 CS	line 36	cppcheck Pro...
❗ (cppcheck performance) Function parameter 'unNombreFich' should be passed by const	Diccionario.cpp	/Practica 2 CS	line 85	cppcheck Pro...
❗ (cppcheck performance) Function parameter 'unNombreFich' should be passed by const	Documento.cpp	/Practica 2 CS	line 35	cppcheck Pro...
❗ (cppcheck performance) Function parameter 'unNombreFich' should be passed by const	Documento.cpp	/Practica 2 CS	line 89	cppcheck Pro...
❗ (cppcheck performance) Function parameter 'unNombreFich' should be passed by const	VerbosConjugados.cpp	/Practica 2 CS	line 80	cppcheck Pro...
❗ (cppcheck performance) Function parameter 'unosVerbosConjugados' should be passed	Diccionario.cpp	/Practica 2 CS	line 94	cppcheck Pro...

Ilustración 34: Infos al ejecutar cppcheck

Corrección de los problemas

(cppcheck error) The one definition rule is violated, different classes/structs have the same name 'Nodo < Palabra >'

La causa de este error es que tenemos 2 clase declaradas con el mismo nombre, este nombre es `Nodo`, y las clases están declaradas como clases auxiliares dentro de la clase `AVL` y `ListaEnlazada`.

```
template<typename U>
class Nodo {
public:
    Nodo<U> *izq;    ///< Nodo a la izquierda del nodo a
    Nodo<U> *der;    ///< Nodo a la derecha del nodo a
    U dato;          ///< Contenido que contendrá el no
    char bal;        ///< Factor de equilibrio del nodo

    /**
     * @brief Constructor parametrizado
     *
     * @param [in] ele Dato a introducir en el nodo
     */
    Nodo(U &ele) :
        izq(0), der(0), dato(ele), bal(0) {
```

Ilustración 1: Clase `Nodo` con error producido por el `cppcheck`

Para solucionar este error, lo que haremos será cambiar el nombre de la clase `Nodo` del fichero `AVL.h`

```
template<typename U>
class Nodo2 {
public:
    Nodo2<U> *izq;    ///< Nodo a la izquierda del nodo actual
    Nodo2<U> *der;    ///< Nodo a la derecha del nodo actual
    U dato;           ///< Contenido que contendrá el nodo
    char bal;         ///< Factor de equilibrio del nodo

    /**
     * @brief Constructor parametrizado
     *
     * @param [in] ele Dato a introducir en el nodo
     */
    Nodo2(U &ele) :
        izq(0), der(0), dato(ele), bal(0) {
    }

    /**
     * @brief Constructor copia
     */
}
```

Ilustración 2: Corrección del error

(cppcheck warning) Possible null pointer dereference: anterior

```

template<typename T>
void ListaEnlazada<T>::borrar(Iterador<T>& i) {
    Nodo<T>* p = i.nodo;
    if (!i.fin()) {
        if (i.nodo == _cola) {
            borraFinal();
        } else if (i.nodo == _cabecera) {
            borraInicio();
        } else {
            Nodo<T> *anterior = 0;
            if (_cabecera != _cola) {
                anterior = _cabecera;
                while (anterior->sig != p) {
                    anterior = anterior->sig;
                }
            }
            anterior->sig = p->sig;
            _tam--;
        }
    }
}

```

Ilustración 3: Error en la función borrar de la clase ListaEnlazada

Para solucionarlo solo tenemos que comprobar antes de asignarlo que `p->sig` es distinto de `nullptr`

```

template<typename T>
void ListaEnlazada<T>::borrar(Iterador<T>& i) {
    Nodo<T>* p = i.nodo;
    if (!i.fin()) {
        if (i.nodo == _cola) {
            borraFinal();
        } else if (i.nodo == _cabecera) {
            borraInicio();
        } else {
            Nodo<T> *anterior = 0;
            if (_cabecera != _cola) {
                anterior = _cabecera;
                while (anterior->sig != p) {
                    anterior = anterior->sig;
                }
            }
            if(p->sig != nullptr){
                anterior->sig = p->sig;
                _tam--;
            }
        }
    }
}

```

Ilustración 4: Corrección del error.

Corrección de las advertencias

(cppcheck style) Parameter 'list' can be declared with const

```

template<typename T>
ListaEnlazada<T> &ListaEnlazada<T>::operator =(ListaEnlazada<T>& list)

```

Ilustración 5: Código del error

Como Podemos ver cppcheck nos aconseja que podemos pasar el parámetro list como constante ya que no se modifica en la función.

Para solucionarlo solo tenemos que declarar el argumento que se pasa como una constante.

```

ListaEnlazada<T> &operator=(const ListaEnlazada<T>& list);

```

Ilustración 6: Cabecera del operador, solucionando el error

```
template<typename T>
ListaEnlazada<T> &ListaEnlazada<T>::operator =(const ListaEnlazada<T>& list) {
    if (_cabecera) { //borramos todos los nodos (mismo código que el destructor)
        Nodo<T> *elimina = _cabecera;
        while (elimina) {
            _cabecera = _cabecera->sig;
            delete elimina;
        }
    }
}
```

Ilustración 7: Implementación del operador con la solución.

(cppcheck style) Parameter 'orig' can be declared with const

Este consejo es igual que el anterior, ya que tenemos una función en la que le pasamos un parámetro que no se modifica en la función.

```
template<typename T>
AVL<T>& AVL<T>::operator=(AVL<T>& orig) {
    copiar(raiz, orig.raiz);
    return *this;
}
```

Ilustración 8: Función con warning

Para solucionarlo el error declaramos la variable que pasamos como constante.

```
template<typename T>
AVL<T>& AVL<T>::operator=(const AVL<T>& orig) {
    copiar(raiz, orig.raiz);
    return *this;
}
```

Ilustración 9: Corrección del warning

(cppcheck style) The scope of the variable 'aux' can be reduced.

(cppcheck style) The scope of the variable 'min' can be reduced.

Esta advertencia lo que nos dice es que reduzcamos el alcance de la variable local que hemos declarado.


```

template<typename T>
void Vdinamico<T>::ordenar() { //algoritmo de ordenaci3n por selecci3n
    T aux;
    int min;
    for (unsigned long int i = 0; i < _tamal; i++) {
        min = i;
        for (unsigned long int j = i + 1; j < _tamal; j++) {
            if (_v[j] < _v[min]) {
                min = j;
            }
        }
        aux = _v[i];
        _v[i] = _v[min];
        _v[min] = aux;
    }
}

```

Como podemos ver en la funci3n ordenar utilizamos 2 variables locales (aux y min) que las declaramos antes que en la secci3n donde las utilizamos, para resolver estos problemas declararemos las variables dentro del bucle for.

```

/**
 * @brief M3todo para ordenar el vector dinámico
 */
template<typename T>
void Vdinamico<T>::ordenar() { //algoritmo de ordenaci3n por selecci3n

    for (unsigned long int i = 0; i < _tamal; i++) {
        int min = i;
        for (unsigned long int j = i + 1; j < _tamal; j++) {
            if (_v[j] < _v[min]) {
                min = j;
            }
        }
        T aux = _v[i];
        _v[i] = _v[min];
        _v[min] = aux;
    }
}

```

Ilustraci3n 10: Correcci3n de las advertencias en la funci3n ordenar de la clase Vdinamico

(cppcheck style) The scope of the variable 'aux' can be reduced.

(cppcheck style) The scope of the variable 'max' can be reduced.

Nos Vuelve a aparecer la advertencia de que hay variables locales que se declaran fuera de su alcance en la función.

```
/**
 * @brief Método para ordenar el vector dinámico pero en sentido contrario
 */
template<typename T>
void VDinamico<T>::ordenarRev() { //algoritmo de ordenación por selección
    T aux;
    int max;
    for (int i = 0; i < _tamal; i++) {
        max = i;
        for (int j = i + 1; j < _tamal; j++) {
            if (_v[j] > _v[max]) {
                max = j;
            }
        }
        aux = _v[i];
        _v[i] = _v[max];
        _v[max] = aux;
    }
}
```

Ilustración 11: Función que produce warnings

Como podemos ver las variables locales aux y max están declaradas fuera de su alcance, el cual es el dentro del bucle for, ya que fuera del bucle for no se utilizan. Para solucionar este error lo que hacemos es declararlas dentro del bucle for.

```

/**
 * @brief Método para ordenar el vector dinámico pero en sentido contrario
 */
template<typename T>
void VDinamico<T>::ordenarRev() { //algoritmo de ordenación por selección
    for (int i = 0; i < _tamal; i++) {
        int max = i;
        for (int j = i + 1; j < _tamal; j++) {
            if (_v[j] > _v[max]) {
                max = j;
            }
        }
        T aux = _v[i];
        _v[i] = _v[max];
        _v[max] = aux;
    }
}

```

Ilustración 12: Corrección del error

(cppcheck style) Variable 'aux' is assigned a value that is never used.

Esta advertencia nos dice que tenemos una variable local declarada que no utilizamos en ningún momento.

```

/**
 * @brief Método para poner al revés una palabra pasada como parametro y asignarla
 *
 * @param pal Palabra que queremos poner como al revés
 * @return Devuelve el resultado de la palabra
 */
void Palabra::reves(std::string pal) {
    if(pal!=""){
        std::string aux = pal;
        int i = pal.length() - 1;
        while (i >= 0) {
            aux += pal[i];
            i--;
        }
        this->setPalabra(pal);
    }
}

```

Ilustración 13: Función con una variable local que no utilizamos

Para solucionar este problema, lo que tenemos que hacer es pasar a la función setPalabra, la variable local que declaramos.

```

/**
 * @brief Método para poner al revés una palabra pasada como parametro y asignarla
 *
 * @param pal Palabra que queremos poner como al revés
 * @return Devuelve el resultado de la palabra
 */
void Palabra::reves(std::string pal) {
    if(pal!=""){
        std::string aux = "";
        int i = pal.length() - 1;
        while (i >= 0) {
            aux += pal[i];
            i--;
        }
        this->setPalabra(aux);
    }
}

```

Ilustración 14: Corrección del funcionamiento de la función

(cppcheck style) The scope of the variable 'curIn' can be reduced.

Esta advertencia nos Vuelve a decir que tenemos declarada una función fuera de su alcance.

```

template<typename T>
unsigned int VDinamico<T>::busquedaBin(T dato) {
    int inf = 0;
    int sup = tamal - 1;
    int curIn;
    while (inf <= sup) {
        curIn = (inf + sup) / 2;
        if (_v[curIn] == dato) {
            return curIn;
        } else {
            if (_v[curIn] < dato) {
                inf = curIn + 1;
            } else {
                sup = curIn - 1;
            }
        }
    }
    return -1;
}

```

Ilustración 15: Función con warning

Como podemos ver, la variable local curIn debería de haber sido declara dentro del bucle while.

```
template<typename T>
unsigned int VDinamico<T>::busquedaBin(T dato) {
    int inf = 0;
    int sup = _tamal - 1;
    while (inf <= sup) {
        int curIn = (inf + sup) / 2;
        if (_v[curIn] == dato) {
            return curIn;
        } else {
            if (_v[curIn] < dato) {
                inf = curIn + 1;
            } else {
                sup = curIn - 1;
            }
        }
    }
    return -1;
}
```

Ilustración 16: Corrección del warning

(cppcheck style) The scope of the variable 'total' can be reduced.

Este error nos aparece en 3 funciones ya que hemos declarado con el mismo nombre 3 variables locales en distintas funciones y en todas nos aconseja cambiarlo.

Vuelve a ser el error de que declaramos una variable fuera de su alcance.

```

void pruebaTiempos() { //funcion para llenar las estructuras de datos de verbos y usarlas en la prueba de ti
    std::ifstream fe;
    string linea;
    int total = 0;
    std::string palabra;

    fe.open("verbos_conjugados_sin_tildes_desordenados.txt"); //abrimos fichero
    if (fe.good()) {
        while (!fe.eof()) { //bucle hasta llegue a la ultima linea
            getline(fe, linea); //leemos la línea
            std::stringstream ss;

            ++total;
            ss << linea;
            getline(ss, palabra); //extraemos la palabra en forma de string de la línea leída
            Palabra pal(palabra); //creamos una Palabra con el string leído
            vd.insertar(pal);
            le.insertaFinal(pal);
            avl.insertar(pal);
        }
        fe.close();
    }
}

```

Ilustración 17: Función `pruebaTiempos()` en la que cometemos el error

En este caso directamente podemos borrar la variable `total` ya que no la utilizamos para nada.

```

void Diccionario::leerDiccionario() {
    std::ifstream fe;
    string linea;
    int total = 0;
    std::string palabra;

    fe.open(getNombreFich()); //abrimos fichero
    if (fe.good()) {
        while (!fe.eof()) { //bucle hasta llegue a la ultima linea
            getline(fe, linea); //leemos la línea
            std::stringstream ss;

            ++total;
            ss << linea;
            getline(ss, palabra); //extraemos la palabra en forma de string de la línea leída
            Palabra pal(palabra); //creamos una Palabra con el string leído
            _terminos.insertar(pal);
        }
        std::cout << "Palabras leídas diccionario: " << total << std::endl;
        std::cout << "Tamaño diccionario: " << _terminos.tam() << std::endl;
        fe.close();
    }
}

```

Ilustración 18: Función `leerDiccionario()` en la que cometemos el error de alcance

Como vemos deberíamos declarar la variable `total` dentro de la sección del `if`.

```

void Diccionario::leerDiccionario() {
    std::ifstream fe;
    string linea;
    std::string palabra;

    fe.open(getNombreFich()); //abrimos fichero
    if (fe.good()) {
        int total = 0;
        while (!fe.eof()) { //bucle hasta llegue a la ultima linea
            getline(fe, linea); //leemos la línea
            std::stringstream ss;

            ++total;
            ss << linea;
            getline(ss, palabra); //extraemos la palabra en forma de string de la línea leída
            Palabra pal(palabra); //creamos una Palabra con el string leído
            _terminos.insertar(pal);
        }
        std::cout << "Palabras leídas diccionario: " << total << std::endl;
        std::cout << "Tamaño diccionario: " << _terminos.tam() << std::endl;
        fe.close();
    }
}

```

Ilustración 19: Corrección del warning

```

/**
 * @brief Método para leer el fichero de verbos e insertarlo en el AVL
 */
void VerbosConjugados::leerVerbos() {
    std::ifstream fe;
    string linea;
    int total = 0;
    std::string palabra;

    fe.open(getNombreFich()); //abrimos fichero
    if (fe.good()) {
        while (!fe.eof()) { //bucle hasta llegue a la ultima linea
            getline(fe, linea); //leemos la línea
            std::stringstream ss;

            ++total;
            ss << linea;
            getline(ss, palabra); //extraemos la palabra en forma de string de la línea leída
            Palabra pal(palabra); //creamos una Palabra con el string leído
            _vconjugados.insertar(pal);
        }

        fe.close();
    }
}

```

Ilustración 20: Función leerVerbos() en la que cometemos el error

En este caso también podemos borrar la variable local total, ya que en ningún momento le damos utilidad.

(cppcheck warning) 'operator=' should check for assignment to self to avoid problems with dynamic memory.

Este warning nos advierte de que deberíamos de comprobar si el parámetro que pasamos somos nosotros mismos, para evitar problemas con la memoria dinámica.

```
template<typename T>
VDinamico<T>& VDinamico<T>::operator =(const VDinamico<T>& origen) {
    if (_v) {
        delete[] _v;
        _tamal = origen._tamal;
        _v = new T[_tamal = origen._tamal];
        for (unsigned long int i = 0; i < _tamal; i++) {
            _v[i] = origen._v[i];
        }
    }
    return *this;
}
```

Ilustración 21: Función con warning

(cppcheck warning) Assignment of function parameter has no effect outside the function. Did you forget dereferencing it?

Esta advertencia nos indica que tal vez hayamos olvidado referenciar una variable

```
template<typename T>
void AVL<T>::destruir(Nodo<T>* nodo) {
    if (nodo) {
        destruir(nodo->izq);
        destruir(nodo->der);
        delete nodo;
        nodo = 0;
    }
}
```

Ilustración 22: Función con warning

Como podemos ver, el parámetro nodo al final de la función lo igualamos a 0, la mejor manera de hacerlo sería referenciarlo a nullptr antes de destruirlo.


```
template<typename T>
void AVL<T>::destruir(Nodo<T>* nodo) {
    if (nodo) {
        destruir(nodo->izq);
        destruir(nodo->der);
        nodo = nullptr;
        delete nodo;
    }
}
```

Ilustración 23: Corrección del warning

(cppcheck style) Class 'Diccionario' has a constructor with 1 argument that is not explicit.

```
Diccionario()
Diccionario(const std::string &unNombreFich);
```

Ilustración 24: Función con warning

Cppcheck nos advierte de que las promociones implícitas en constructores de un solo parámetro pueden provocar un comportamiento inesperado por lo que deben evitarse. Esto se consigue añadiendo “explicit” delante del constructor.

```
explicit Diccionario(const std::string &unNombreFich);
```

Ilustración 25: Corrección del warning

(cppcheck style) Class 'Documento' has a constructor with 1 argument that is not explicit.

```
Documento(const std::string &unNombreFich);
```

Ilustración 26: Función con warning

Al igual que en el warning anterior, cppcheck nos advierte de que las promociones implícitas en constructores de un solo parámetro pueden provocar un comportamiento

inesperado por lo que deben evitarse. Esto se consigue añadiendo “explicit” delante del constructor.

```
explicit Documento(const std::string &unNombreFich);
```

Ilustración 27: Corrección del warning

(cppcheck style) Class 'Iterador<Palabra>' has a constructor with 1 argument that is not explicit.

```
Iterador(Nodo<V> *aNodo) :  
    nodo(aNodo) {  
}
```

Ilustración 28: Función con warning

Al igual que en el warning anterior, cppcheck nos advierte de que las promociones implícitas en constructores de un solo parámetro pueden provocar un comportamiento inesperado por lo que deben evitarse. Esto se consigue añadiendo “explicit” delante del constructor.

```
explicit Iterador(Nodo<V> *aNodo) :  
    nodo(aNodo) {  
}
```

Ilustración 29: Corrección del warning

(cppcheck style) Class 'Nodo<Palabra>' has a constructor with 1 argument that is not explicit.

```
Nodo(const U& aDato, Nodo* aSig = 0) :  
    dato(aDato), sig(aSig) {  
}
```

Ilustración 30: Función con warning

Al igual que en el warning anterior, cppcheck nos advierte de que las promociones implícitas en constructores de un solo parámetro pueden provocar un comportamiento inesperado por lo que deben evitarse. Esto se consigue añadiendo “explicit” delante del constructor.

```
explicit Nodo(const U& aDato, Nodo* aSig = 0) :
    dato(aDato), sig(aSig) {
}
```

Ilustración 31: Corrección del warning

(cppcheck style) Class 'Nodo2<Palabra>' has a constructor with 1 argument that is not explicit.

```
Nodo2(U &ele) :
    izq(0), der(0), dato(ele), bal(0) {
}
```

Ilustración 32: Función con warning

Al igual que en el warning anterior, cppcheck nos advierte de que las promociones implícitas en constructores de un solo parámetro pueden provocar un comportamiento inesperado por lo que deben evitarse. Esto se consigue añadiendo “explicit” delante del constructor.

```
explicit Nodo2(U &ele) :
    izq(0), der(0), dato(ele), bal(0) {
}
```

Ilustración 33: Corrección del warning

(cppcheck style) Class 'Palabra' has a constructor with 1 argument that is not explicit.

```
Palabra(const std::string &unaPalabra);
```

Ilustración 34: Función con warning

Al igual que en el warning anterior, cppcheck nos advierte de que las promociones implícitas en constructores de un solo parámetro pueden provocar un comportamiento inesperado por lo que deben evitarse. Esto se consigue añadiendo “explicit” delante del constructor.

```
explicit Palabra(const std::string &unaPalabra);
```

Ilustración 35: Corrección del warning

(cppcheck style) Class 'VDinamico<char>' has a constructor with 1 argument that is not explicit.

```
VDinamico(unsigned long int tam);
```

Ilustración 36: Función con warning

Al igual que en el warning anterior, cppcheck nos advierte de que las promociones implícitas en constructores de un solo parámetro pueden provocar un comportamiento inesperado por lo que deben evitarse. Esto se consigue añadiendo “explicit” delante del constructor.

```
explicit VDinamico(unsigned long int tam);
```

Ilustración 37: Corrección del warning

(cppcheck style) Class 'VDinamico<Documento>' has a constructor with 1 argument that is not explicit.

```
VDinamico(unsigned long int tam);
```

Ilustración 38: Función con warning

Al corregir el warning anterior, desaparece automáticamente.

```
explicit VDinamico(unsigned long int tam);
```

Ilustración 39: Corrección del warning

(cppcheck style) Class 'VDinamico<Palabra>' has a constructor with 1 argument that is not explicit.

```
VDinamico(),
VDinamico(unsigned long int tam);
```

Ilustración 40: Función con warning

Al corregir el warning anterior, desaparece automáticamente.

```
explicit VDinamico(unsigned long int tam);
```

Ilustración 41: Corrección del warning

(cppcheck style) The function 'anagrama' is never used.

```
bool Palabra::anagrama(Palabra pal) { //comparamos dos vectores de letras ord
    VDinamico<Palabra> aux1 = this->getCombinaciones(); //se obtienen los vec
    VDinamico<Palabra> aux2 = pal.getCombinaciones();
    bool anagrama = true;
    if (aux1.tam() == aux2.tam()) { //se comparan las dos palabras comparando
        for (unsigned int i = 0; i < this->getCombinaciones().tam(); i++) {
            if (aux1[i]._palabra != aux2[i]._palabra) {
                return false; //si todas las palabras de cada posición son i
            }
        }
    } else {
        return false;
    }
    return anagrama;
}
```

Ilustración 42: Función con warning

Como vimos en clase la solución a este problema es seleccionar "ignore problem in this whole file" en la bombilla de advertencia ya que nos interesa mantener la función para un futuro uso antes que borrarla.

```

bool Palabra::anagrama(Palabra pal) { //comparamos dos vectores de letras or
    VDinamico<Palabra> aux1 = this->getCombinaciones(); //se obtienen los ve
    VDinamico<Palabra> aux2 = pal.getCombinaciones();
    bool anagrama = true;
    if (aux1.tam() == aux2.tam()) { //se comparan las dos palabras comparand
        for (unsigned int i = 0; i < this->getCombinaciones().tam(); i++) {
            if (aux1[i]._palabra != aux2[i]._palabra) {
                return false; //si todas las palabras de cada posiciÃ³n son
            }
        }
    } else {
        return false;
    }
    return anagrama;
}

```

Ilustración 43: Corrección del warning

(cppcheck style) The function 'getDiccionario' is never used.

```

Diccionario Documento::getDiccionario() {
    return this->_dicc;
}

```

Ilustración 44: Función con warning

Como vimos en clase la solución a este problema es seleccionar “ignore problem in this whole file” en la bombilla de advertencia ya que nos interesa mantener la función para un futuro uso antes que borrarla.

```

Diccionario Documento::getDiccionario() {
    return this->_dicc;
}

```

Ilustración 45: Corrección del warning

(cppcheck style) The function 'getInexistentes' is never used.

```

ListaEnlazada<Palabra> Documento::getInexistentes() {
    return this->_inexistentes;
}

```

Ilustración 46: Función con warning

Como vimos en clase la solución a este problema es seleccionar “ignore problem in this whole file” en la bombilla de advertencia ya que nos interesa mantener la función para un futuro uso antes que borrarla.

```
1 //  
2 ListaEnlazada<Palabra> Documento::getInexistentes() {  
3     return this->_inexistentes;  
4 }  
5
```

Ilustración 47: Corrección del warning

(cppcheck style) The function 'getVConjugados' is never used.

```
1 AVL<Palabra> VerbosConjugados::getVConjugados() {  
2     return _vconjugados;  
3 }  
4
```

Ilustración 48: Función con warning

Como vimos en clase la solución a este problema es seleccionar “ignore problem in this whole file” en la bombilla de advertencia ya que nos interesa mantener la función para un futuro uso antes que borrarla.

```
1 AVL<Palabra> VerbosConjugados::getVConjugados() {  
2     return _vconjugados;  
3 }  
4
```

Ilustración 49: Corrección del warning

(cppcheck style) The function 'mostrarDiccionario' is never used.

```
1 void Diccionario::mostrarDiccionario() {  
2     for (unsigned int i = 0; i < _terminos.tam(); i++) {  
3         std::cout << _terminos[i].getPalabra() << std::endl;  
4     }  
5 }
```

Ilustración 50: Función con warning

Como vimos en clase la solución a este problema es seleccionar “ignore problem in this whole file” en la bombilla de advertencia ya que nos interesa mantener la función para un futuro uso antes que borrarla.

```
void Diccionario::mostrarDiccionario() {
    for (unsigned int i = 0; i < _terminos.tam(); i++) {
        std::cout << _terminos[i].getPalabra() << std::endl;
    }
}
```

Ilustración 51: Corrección del warning

(cppcheck style) The function ‘MostrarInexistentes’ is never used.

```
void Documento::MostrarInexistentes() { //Se recorre con in iterador la lista y la
    ListaEnlazada<Palabra>::Iterador<Palabra> it = _inexistentes.iterador();
    int contador = 0;
    while (it.haySiguiente()) {
        std::cout << contador << ": " << it.nodo->dato.getPalabra()
            << std::endl;
        it.siguiente();
        contador++;
    }
}
```

Ilustración 52: Función con warning

Como vimos en clase la solución a este problema es seleccionar “ignore problem in this whole file” en la bombilla de advertencia ya que nos interesa mantener la función para un futuro uso antes que borrarla.

```
void Documento::MostrarInexistentes() { //Se recorre con in iterador la lista :
    ListaEnlazada<Palabra>::Iterador<Palabra> it = _inexistentes.iterador();
    int contador = 0;
    while (it.haySiguiente()) {
        std::cout << contador << ": " << it.nodo->dato.getPalabra()
            << std::endl;
        it.siguiente();
        contador++;
    }
}
```


Ilustración 53: Corrección del warning

(cppcheck style) The function 'palindromo' is never used.

```
bool Palabra::palindromo(Palabra pal) {  
    if(pal.getPalabra()!=""){  
        if (_palabra == pal.reves()._palabra) {  
            return true;  
        }  
    }  
    return false;  
}
```

Ilustración 54: Función con warning

Como vimos en clase la solución a este problema es seleccionar “ignore problem in this whole file” en la bombilla de advertencia ya que nos interesa mantener la función para un futuro uso antes que borrarla.

```
bool Palabra::palindromo(Palabra pal) {  
    if(pal.getPalabra()!=""){  
        if (_palabra == pal.reves()._palabra) {  
            return true;  
        }  
    }  
    return false;  
}
```

Ilustración 55: Corrección del warning

(cppcheck style) The function 'pruebaTiempos' is never used.

```

//
void pruebaTiempos() { //funcion para llenar las estructuras de datos de verbos
    std::ifstream fe;
    string linea;
    std::string palabra;

    fe.open("verbos_conjugados_sin_tildes_desordenados.txt"); //abrimos fichero
    if (fe.good()) {
        while (!fe.eof()) { //bucle hasta llegue a la ultima linea
            getline(fe, linea); //leemos la linea
            std::stringstream ss;

            ss << linea;
            getline(ss, palabra); //extraemos la palabra en forma de string de
            Palabra pal(palabra); //creamos una Palabra con el string leído
            vd.insertar(pal);
            le.insertaFinal(pal);
            avl.insertar(pal);
        }

        fe.close();
    }
}

```

Ilustración 56: Función con warning

Como vimos en clase la solución a este problema es seleccionar “ignore problem in this whole file” en la bombilla de advertencia ya que nos interesa mantener la función para un futuro uso antes que borrarla.

```

void pruebaTiempos() { //funcion para llenar las estructuras de datos de ver
    std::ifstream fe;
    string linea;
    std::string palabra;

    fe.open("verbos_conjugados_sin_tildes_desordenados.txt"); //abrimos fich
    if (fe.good()) {
        while (!fe.eof()) { //bucle hasta llegue a la ultima linea
            getline(fe, linea); //leemos la línea
            std::stringstream ss;

            ss << linea;
            getline(ss, palabra); //extraemos la palabra en forma de string
            Palabra pal(palabra); //creamos una Palabra con el string leído
            vd.insertar(pal);
            le.insertaFinal(pal);
            avl.insertar(pal);
        }

        fe.close();
    }
}

```

Ilustración 57: Corrección del warning

(cppcheck style) The function 'setDiccionario' is never used.

```

void Documento::setDiccionario(const Diccionario &unDiccionario) {
    this->_dicc = unDiccionario;
}

```

Ilustración 58: Función con warning

Como vimos en clase la solución a este problema es seleccionar "ignore problem in this whole file" en la bombilla de advertencia ya que nos interesa mantener la función para un futuro uso antes que borrarla.

```

void Documento::setDiccionario(const Diccionario &unDiccionario) {
    this->_dicc = unDiccionario;
}

```

(cppcheck style) The function 'setVConjugado' is never used.

```

void VerbosConjugados::setVConjugados(AVL<Palabra> unVConjugados) {
    this->_vconjugados = unVConjugados;
}

```

Ilustración 59: Función con warning

Como vimos en clase la solución a este problema es seleccionar “ignore problem in this whole file” en la bombilla de advertencia ya que nos interesa mantener la función para un futuro uso antes que borrarla.

```

void VerbosConjugados::setVConjugados(AVL<Palabra> unVConjugados) {
    this->_vconjugados = unVConjugados;
}

```

Ilustración 60: Corrección del warning

(cppcheck style) The function ‘setVerbos’ is never used.

```

void Diccionario::setVerbos(const VerbosConjugados &unosVerbosConjugados) {
    this->_verbos = unosVerbosConjugados;
}

```

Ilustración 61: Función con warning

Como vimos en clase la solución a este problema es seleccionar “ignore problem in this whole file” en la bombilla de advertencia ya que nos interesa mantener la función para un futuro uso antes que borrarla.

```

void Diccionario::setVerbos(const VerbosConjugados &unosVerbosConjugados) {
    this->_verbos = unosVerbosConjugados;
}

```

Ilustración 62: Corrección del warning

Corrección de los infos

En estos avisos que aparecen a continuación, todos son producidos por el mismo fallo, por lo que explicaremos una vez que nos quiere decir cppcheck dándonos esta información.

Esta información nos comenta que deberíamos de pasar el parámetro por una referencia constante.

Para solucionar las siguientes informaciones, lo que haremos será hacer caso a cppcheck y pasar todos los parámetros como referencias constantes

(cppcheck performance) Function parameter 'dato' should be passed by const reference.

```
template<typename T>
unsigned int VDinamico<T>::busquedaBin(T dato) {
    int inf = 0;
    int sup = _tamal - 1;
    while (inf <= sup) {
        int curIn = (inf + sup) / 2;
        if (_v[curIn] == dato) {
            return curIn;
        } else {
            if (_v[curIn] < dato) {
                inf = curIn + 1;
            } else {
                sup = curIn - 1;
            }
        }
    }
}
```

Ilustración 63: Función que produce un info por cppcheck

Para solucionarlo basta con hacer caso a cppcheck y pasar el dato por una referencia constante.

```
template<typename T>
unsigned int VDinamico<T>::busquedaBin(T const &dato) {
    int inf = 0;
    int sup = _tamal - 1;
    while (inf <= sup) {
        int curIn = (inf + sup) / 2;
        if (_v[curIn] == dato) {
            return curIn;
        } else {
            if (_v[curIn] < dato) {
                inf = curIn + 1;
            } else {
                sup = curIn - 1;
            }
        }
    }
    return -1;
}
```

Ilustración 64: Corrección de la info

En las siguientes informaciones solamente pondremos la solución realizada mostrando la cabecera de las funciones afectadas.

(cppcheck performance) Function parameter 'unaPalabra' should be passed by const reference.

Constructor de la clase Palabra

```
Palabra(const std::string &unaPalabra);
```

Ilustración 65: Corrección de la info

(cppcheck performance) Function parameter 'unaPalabra' should be passed by const reference.

Función setPalabra de la clase Palabra

```
void setPalabra(const std::string &unaPalabra);
```

Ilustración 66: Corrección de la info

(cppcheck performance) Function parameter 'unDiccionario' should be passed by const reference.

```
void setDiccionario(const Diccionario &unDiccionario);
```

Ilustración 67: Corrección de la info

(cppcheck performance) Function parameter 'unNombreFich' should be passed by const reference.

Función setNombreFich de la clase VerbosConjugados

```
void setNombreFich(const std::string &unNombreFich);
```

Ilustración 68: Corrección de la info

(cppcheck performance) Function parameter ' unNombreFich ' should be passed by const reference.

La función setNombreFich de la clase Diccionario

```
void setNombreFich(const std::string &unNombreFich);
```

Ilustración 69: Corrección de la info

(cppcheck performance) Function parameter ' unNombreFich ' should be passed by const reference.

La función setNombreFich de la clase Documento

```
void setNombreFich(const std::string &unNombreFich);
```

Ilustración 70: Corrección de la info

(cppcheck performance) Function parameter ' unNombreFich ' should be passed by const reference.

Constructor de la clase Documento

```
Documento(const std::string &unNombreFich);
```

Ilustración 71: Corrección de la info

(cppcheck performance) Function parameter ' unNombreFich ' should be passed by const reference.

Constructor de la clase Diccionario

```
Diccionario(const std::string &unNombreFich);
```

Ilustración 72: Corrección de la info

(cppcheck performance) Function parameter 'unosVerbosConjugados' should be passed by const reference.

Función setVerbos de la clase Diccionario

```
void setVerbos(const VerbosConjugados &unosVerbosConjugados);
```

Ilustración 73: Corrección de la info

Apartado 4: Completar tipos de errores.

Como podemos ver nuestro proyecto tiene 9 tipos de errores. Por lo que se ha seleccionado una comprobación extra de las disponibles en cppcheck. En concreto (cppcheck style) Condition "x" is always false.

Ilustramos su cumplimiento:

```
bool Palabra::palindromo(Palabra pal) {  
    if(pal.getPalabra()!=""){  
        if (_palabra == pal.reves()._palabra) {  
            return true;  
        }  
    }  
    return false;  
}
```


Ilustración 74: Cumplimiento de la comprobación.

Observamos que la condición `pal.getPalabra()!=" "` no será siempre falsa ya que cuando la palabra esté vacía la condición será verdadera.

```
void Palabra::reves(std::string pal) {  
    if(pal!=""){  
        std::string aux = "";  
        int i = pal.length() - 1;  
        while (i >= 0) {  
            aux += pal[i];  
            i--;  
        }  
        this->setPalabra(aux);  
    }  
}
```

Ilustración 75: Cumplimiento de la comprobación.

En esta imagen podemos observar que también se cumple.

Práctica 4

Configurar el plugin metriculator para que marque aquellos módulos con una complejidad ciclomática superior a 10

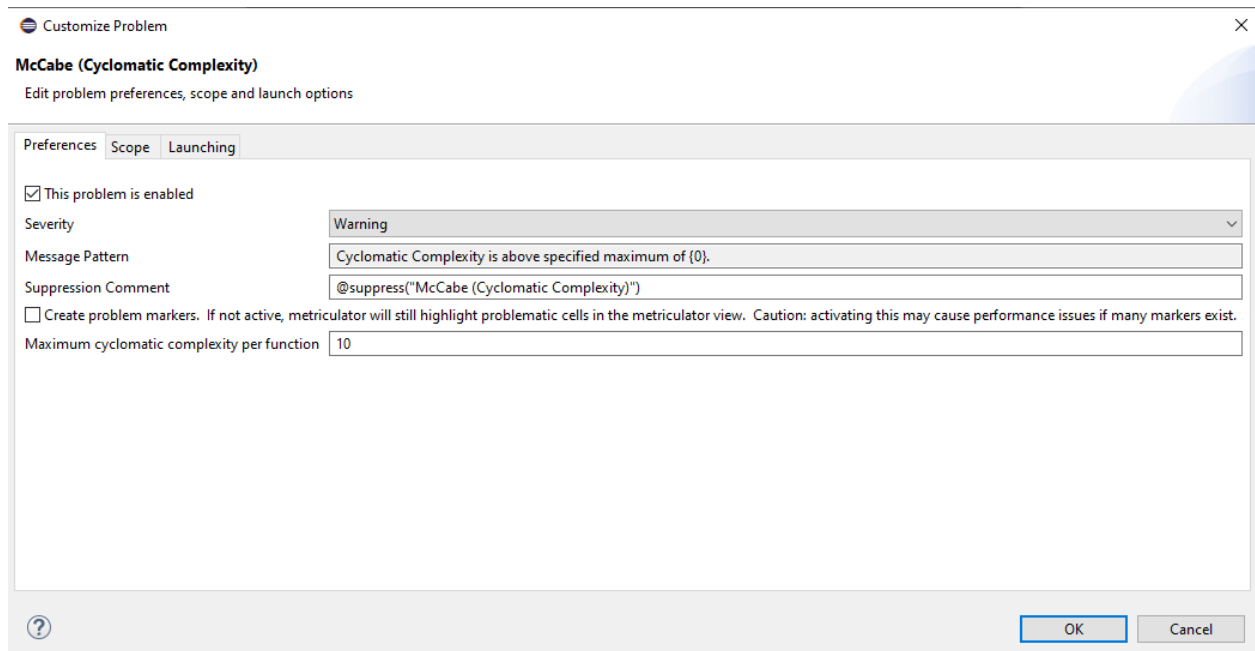


Ilustración 1: Cambio de la configuración del matriculador para calcular la complejidad ciclomática

Utilizar metriculador para calcular la complejidad ciclomática $V(G)$ de los módulos de nuestro proyecto

Scope	EfferentCoupli...	NbMembers	McCabe	NbParams	LSLOC
▼ Practica 2 CS	34	152	137	152	887
> ListaEnlazada.h	6	32	41	19	183
> AVL.h	4	27	30	46	152
> VDinamico.h	2	19	29	24	131
> Palabra.cpp	0	0	18	12	105
> Documento.cpp	0	0	9	6	63
> Diccionario.cpp	0	0	5	5	44
> VerbosConjugados.	0	0	4	5	39
> main.cpp	0	0	3	2	37
> Diccionario.h	5	14	2	5	22
> Documento.h	5	17	2	6	26
> GestorTextos.h	5	9	2	3	16
> Palabra.h	3	21	2	12	27
> VerbosConjugados.	4	13	2	5	20
> GestorTextos.cpp	0	0	1	2	22

Ilustración 2: Complejidad ciclomática de los ficheros del proyecto

Obtener una captura de pantalla de la pestaña asociada a la primera aplicación de metriculador ordenada por la métrica de McCabe en orden descendente

Scope (239 items)	EfferentCoupli...	NbMembers	McCabe	NbParams	LSLOC
● AVL<T>::inserta(Nodo2...	0	0	12	2	26
● ListaEnlazada<T>::inser...	0	0	9	1	22
● ListaEnlazada<T>::borr...	0	0	7	1	15
● &ListaEnlazada<T>::op...	0	0	6	1	20
● VDinamico<T>::insertar...	0	0	6	2	13
● VDinamico<T>::borrar(...	0	0	6	1	13
● Documento::borrarNo...	0	0	5	0	12
● ListaEnlazada<T>::inser...	0	0	5	2	13
● Palabra::limpiar()	0	0	5	0	14
● *AVL<T>::buscaClave(c...	0	0	4	2	8
● AVL<T>::buscalt(const ...	0	0	4	2	10

Ilustración 3: Complejidad ciclomática de los distintos módulos del proyecto

Como podemos ver en total tenemos 239 funciones en el proyecto, de las cuales solo 1 supera los límites prefijados de la métrica de McCabe.

Modificar aquellos módulos que presenten una $V(G) > 10$ hasta conseguir una $V(G) \leq 10$. Si ningún módulo alcanza dicho valor, modificar aquellos con $LSLOC > 25$ de manera que todos los módulos presenten $LSLOC \leq 25$. En este último caso, añadir a la captura del punto anterior otra captura ordenada por LSLOC en orden descendente

Función: AVL<T>::inserta(Nodo2<T>*& c, T& dato)

```

272 template<typename T>
273 int AVL<T>::inserta(Nodo2<T>*& c, T& dato) {
274     Nodo2<T> *p = c;
275     int deltaH = 0;
276     if (!p) {
277         p = new Nodo2<T>(dato);
278         c = p;
279         deltaH = 1;
280     } else if (dato > p->dato) {
281         if (inserta(p->der, dato)) {
282             p->bal--;
283             if (p->bal == -1) {
284                 deltaH = 1;
285             } else if (p->bal == -2) {
286                 if (p->der->bal == 1) {
287                     rotDecha(p->der);
288                     rotIzqda(c);
289                 }
290             }
291         }
292     } else if (dato < p->dato) {
293         if (inserta(p->izq, dato)) {
294             p->bal++;
295             if (p->bal == 1) {
296                 deltaH = 1;
297             } else if (p->bal == 2) {
298                 if (p->izq->bal == -1) {
299                     rotIzqda(p->izq);
300                     rotDecha(c);
301                 }
302             }
303         }
304     }
305     return deltaH;
306 }

```

Ilustración 4: Función inserta de la clase AVL con un McCabe de 12

Para reducir la complejidad ciclomática lo que haremos es crear 2 nuevas funciones las que se encargarán de insertar el dato en la parte izquierda o en la parte derecha del árbol AVL y realizar las correspondientes rotaciones de las ramas.

```

/**
 * @brief Método para insertar un nuevo nodo en el Árbol
 *
 * @param [in] c Nodo nuevo a insertar
 * @param [in] dato Dato que contiene el nuevo Nodo
 * @return Devuelve la variación de la altura del árbol producida por la insercción
 */
template<typename T>
int AVL<T>::inserta(Nodo2<T>*& c, T& dato) {
    Nodo2<T> *p = c;
    int deltaH = 0;
    if (!p) {
        p = new Nodo2<T>(dato);
        c = p;
        deltaH = 1;
    } else if (dato > p->dato) {
        this->insertaDerecha(c,dato,p,deltaH);
    } else if (dato < p->dato) {
        this->insertaIzquierda(c,dato,p,deltaH);
    }
    return deltaH;
}

```

Ilustración 5: Modificación de la función inserta de la clase AVL

Las funciones auxiliares creadas son:

```

/**
 * @brief Función para insertar un dato en un nodo de la parte derecha el árbol
 *
 * @param [in out] c Nodo del árbol que sería rotado hacia la izquierda
 * @param [in] dato Dato a introducir en el árbol
 * @param [in out] p Nodo del árbol en el que se inserta el dato
 * @param [in out] deltaH Valor del delta de h del árbol
 */
template<typename T>
void AVL<T>::insertaDerecha(Nodo2<T>*& c, T& dato,Nodo2<T>* &p,int &deltaH) {
    if (inserta(p->der, dato)) {
        p->bal--;
        if (p->bal == -1) {
            deltaH = 1;
        } else if (p->bal == -2) {
            if (p->der->bal == 1) {
                rotDecha(p->der);
                rotIzqda(c);
            }
        }
    }
}

```

Ilustración 6: Función nueva para insertar un dato en la parte derecha del árbol

```

/**
 * @brief Función para insertar un dato en un nodo de la parte izquierda el arbol
 *
 * @param [in out] c Nodo del arbol que sería rotado hacia la derecha
 * @param [in] dato Dato a introducir en el arbol
 * @param [in out] p Nodo del arbol en el que se inserta el dato
 * @param [in out] deltaH Valor del delta de h del árbol
 */
template<typename T>
void AVL<T>::insertaIzquierda(Nodo2<T>*& c, T& dato, Nodo2<T>* &p, int &deltaH) {
    if (inserta(p->izq, dato)) {
        p->bal++;
        if (p->bal == 1) {
            deltaH = 1;
        } else if (p->bal == 2) {
            if (p->izq->bal == -1) {
                rotIzqda(p->izq);
                rotDecha(c);
            }
        }
    }
}
}

```

Ilustración 7: Función para insertar un dato en la parte izquierda del árbol

Aplicar cppcheck al proyecto actualizado y corregir los errores y avisos que se pudieran detectar sobre el código modificado. Realizar las capturas de pantalla necesarias para justificar el antes y el después de su aplicación













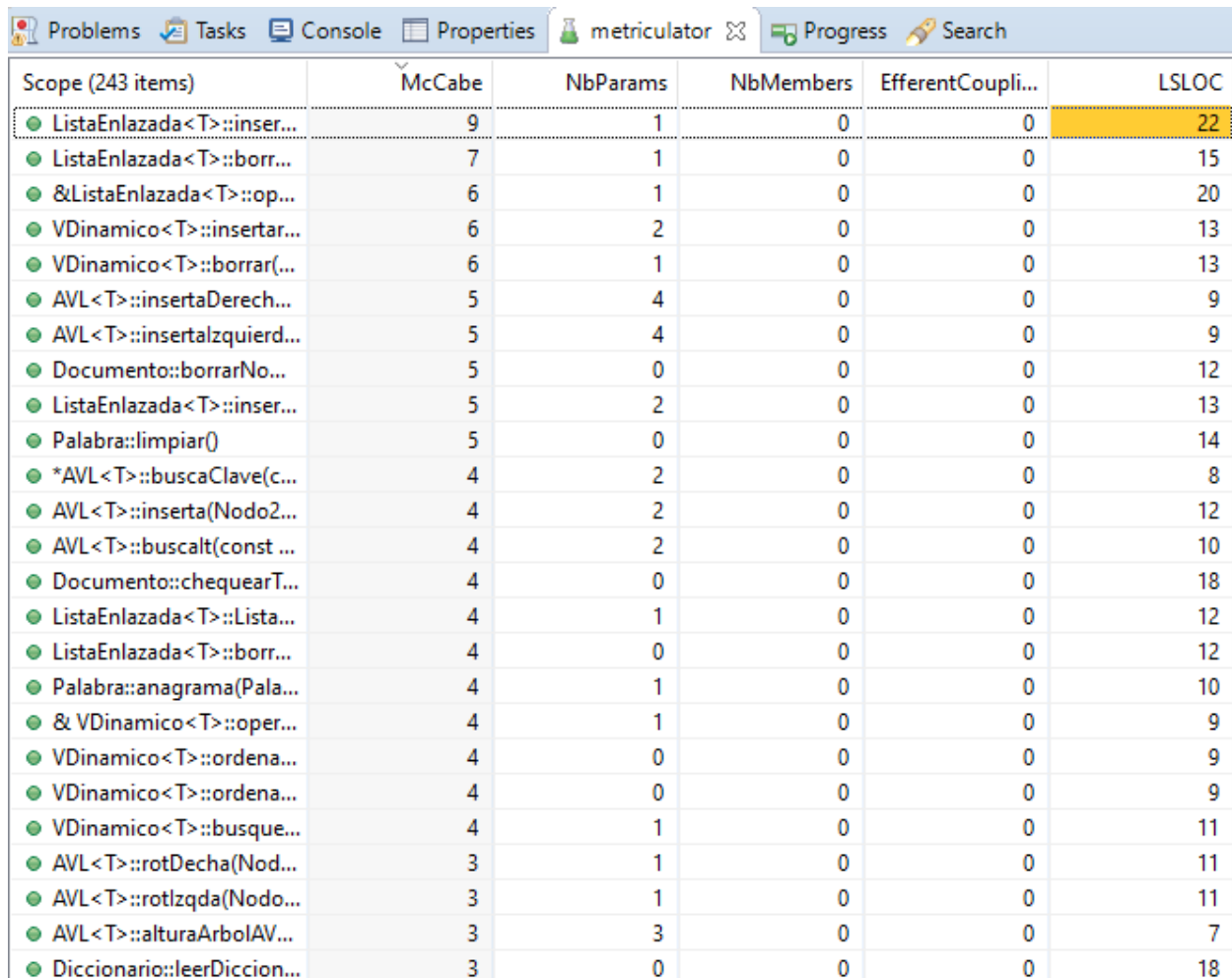
Description	Resource	Path	Location	Type
Warnings (11 items)				
 (cppcheck style) The function 'anagrama' is never used.	Palabra.cpp	/Practica 2 CS	line 214	cppcheck Problem
 (cppcheck style) The function 'getDiccionario' is never used.	Documento.cpp	/Practica 2 CS	line 71	cppcheck Problem
 (cppcheck style) The function 'getInexistentes' is never used.	Documento.cpp	/Practica 2 CS	line 80	cppcheck Problem
 (cppcheck style) The function 'getVConjugados' is never used.	VerbosConjugados.cpp	/Practica 2 CS	line 71	cppcheck Problem
 (cppcheck style) The function 'mostrarDiccionario' is never used.	Diccionario.cpp	/Practica 2 CS	line 59	cppcheck Problem
 (cppcheck style) The function 'MostrarInexistentes' is never used.	Documento.cpp	/Practica 2 CS	line 122	cppcheck Problem
 (cppcheck style) The function 'palindromo' is never used.	Palabra.cpp	/Practica 2 CS	line 181	cppcheck Problem
 (cppcheck style) The function 'pruebaTiempos' is never used.	main.cpp	/Practica 2 CS	line 37	cppcheck Problem
 (cppcheck style) The function 'setDiccionario' is never used.	Documento.cpp	/Practica 2 CS	line 98	cppcheck Problem
 (cppcheck style) The function 'setVConjugados' is never used.	VerbosConjugados.cpp	/Practica 2 CS	line 89	cppcheck Problem
 (cppcheck style) The function 'setVerbos' is never used.	Diccionario.cpp	/Practica 2 CS	line 94	cppcheck Problem
Infos (1 item)				
 (cppcheck information) Cppcheck cannot find all the include files (use --check-config fo	Practica 2 CS		Unknown	cppcheck Problem

Ilustración 8: Ejecución del cppcheck

Como podemos ver no tenemos ningún problema y ningún warning, ya que esos warning hace referencia a funciones que no se utilizan, pero aun así queremos mantener, como se explico en la práctica 3.

Obtener una captura de pantalla de la pestaña asociada a la última aplicación de metriculator ordenada por la métrica de McCabe en orden descendente. Añadir también captura de pantalla ordenada por LSLOC si se ha utilizado esta métrica



Scope (243 items)	McCabe	NbParams	NbMembers	EfferentCoupli...	LSLOC
● ListaEnlazada<T>::inser...	9	1	0	0	22
● ListaEnlazada<T>::borr...	7	1	0	0	15
● &ListaEnlazada<T>::op...	6	1	0	0	20
● VDinamico<T>::insertar...	6	2	0	0	13
● VDinamico<T>::borrar(...	6	1	0	0	13
● AVL<T>::insertaDerech...	5	4	0	0	9
● AVL<T>::insertalzquierd...	5	4	0	0	9
● Documento::borrarNo...	5	0	0	0	12
● ListaEnlazada<T>::inser...	5	2	0	0	13
● Palabra::limpiar()	5	0	0	0	14
● *AVL<T>::buscaClave(c...	4	2	0	0	8
● AVL<T>::inserta(Nodo2...	4	2	0	0	12
● AVL<T>::buscalt(const ...	4	2	0	0	10
● Documento::chequearT...	4	0	0	0	18
● ListaEnlazada<T>::Lista...	4	1	0	0	12
● ListaEnlazada<T>::borr...	4	0	0	0	12
● Palabra::anagrama(Pala...	4	1	0	0	10
● & VDinamico<T>::oper...	4	1	0	0	9
● VDinamico<T>::ordena...	4	0	0	0	9
● VDinamico<T>::ordena...	4	0	0	0	9
● VDinamico<T>::busque...	4	1	0	0	11
● AVL<T>::rotDecha(Nod...	3	1	0	0	11
● AVL<T>::rotlzqda(Nodo...	3	1	0	0	11
● AVL<T>::alturaArbolAV...	3	3	0	0	7
● Diccionario::leerDiccion...	3	0	0	0	18

Ilustración 9: Ejecución del metriculator

Como podemos ver ya no hay ninguna función en nuestro proyecto que supere un 10 en complejidad ciclomática.

Scope	McCabe	NbParams	NbMembers	EfferentCoupli...	LSLOC
▼ Practica 2 CS	137	168	154	34	893
> ListaEnlazada.h	41	19	32	6	183
▼ AVL.h	30	62	29	4	158
● AVL<T>::insertaDerecha(Nodo2<T>	5	4	0	0	9
● AVL<T>::insertaIzquierda(Nodo2<T>	5	4	0	0	9
● *AVL<T>::buscaClave(const T& dato	4	2	0	0	8
● AVL<T>::inserta(Nodo2<T>* &c, T&	4	2	0	0	12
● AVL<T>::buscaIzq(const T& dato, T&	4	2	0	0	10
● AVL<T>::rotDcha(Nodo2<T>* &p)	3	1	0	0	11
● AVL<T>::rotIzqda(Nodo2<T>* &p)	3	1	0	0	11
● AVL<T>::alturaArbolAVL(Nodo2<T>	3	3	0	0	7
● AVL<T>::copiar(Nodo2<T>* &nuev	2	2	0	0	6
● AVL<T>::destruir(Nodo2<T>* nodo)	2	1	0	0	6
● AVL<T>::buscaR(const T& dato, T&	2	2	0	0	6
● AVL<T>::inorden(Nodo2<T>* p, int	2	2	0	0	5
● AVL<T>::numElementosAVL(Nodo2	2	1	0	0	4

Ilustración 10: Funciones de la clase AVL

Como podemos ver la función inserta ha pasado de tener un McCabe de 12 a uno de 4, disminuyendo su $V(G)$ lo que proporciona un mayor mantenimiento de la función.