Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

TRANSFORMACIÓN DE CÓDIGO FUENTE BASADA EN EL LENGUAJE AWK DE UNIX

Autor:

Fraile Cáceres, Álvaro

Tutor:

Podar Cristea, Smaranda
Mazaeda Echevarría, Rogelio

Departamento de Ingeniería de Sistemas y Automática

Valladolid, Noviembre de 2017.

Resumen y palabras clave

Este proyecto nació con la intención de modificar código C/C++ de modo que tuviera un estilo común de programación.

Pronto surgió la idea de que esta herramienta, por su estructura, podría destinarse de igual modo a la transformación de un código fuente en otro diferente. Sin embargo, antes de ello, nos planteamos diferentes cuestiones: ¿cuál es el tipo de lenguaje más adecuado para este propósito?, e incluso, ¿podríamos programarlo de modo que nos confirmara con total certeza si la transformación del código ha sido exitosa?, es decir, ¿el nuevo lenguaje realiza la misma función que el original?

Para responder a estas preguntas, realizaremos un recorrido a través de la ciencia de la computación. Entre otras cuestiones, veremos que las expresiones regulares son útiles para nuestro propósito. Para ello utilizaremos el AWK implementado en Unix, un lenguaje que trabaja bien con ellas.

Palabras clave: conversión de código fuente, ciencia de la computación, expresiones regulares, lenguaje AWK, Unix.

Contenido

1. Introducción

A día de hoy existen multitud de lenguajes de programación entre los cuales podemos encontrar grandes diferencias, no sólo en el lenguaje en sí, sino en su gestión de la memoria, tiempo de ejecución y otros parámetros. Sin embargo, el programador medio no conoce todos los lenguajes existentes, por lo que utilizar una función disponible en un lenguaje desconocido para él puede ser un problema. Lo mismo ocurre si se da un traspaso de conocimiento o de un proyecto entre empresas; pongamos la empresa A y la empresa B: la empresa A tiene una aplicación que le va a transmitir a la empresa B, creado en una herramienta comercial como *Matlab* y la empresa B utiliza *Scilab* como herramienta gratuita. Pero, ¿y si pudiéramos pasar de un programa en C a uno en Java, o de un archivo de *Matlab* a otro de *Scilab*? Esto es precisamente lo que persigue este proyecto, demostrar que es posible, empleando AWK, una herramienta de Unix.

Aunque no se han adquirido conocimientos del AWK en el grado, se ha optado por implementar la herramienta en este lenguaje porque sus características cuadran perfectamente con el objetivo que se persigue. AWK es un lenguaje basado en C (que si que hemos trabajado con él en el grado) y que por tanto tiene muchas de sus características, pero destaca especialmente por su facilidad y rapidez a la hora de identificar patrones de texto en ficheros; en otras palabras: aunque es Turing-completo, su facilidad a la hora de implementar expresiones regulares para buscar estructuras y sustituir estas estructuras por otras es ideal para nuestro propósito.

Pero antes de implementar la herramienta realizaremos un recorrido a través de la historia de la ciencia de la computación, con conceptos y teoremas que nos ayudarán a entender qué podemos resolver y qué no, qué limitaciones tenemos o qué diferencia hay entre un lenguaje Turing-completo o un lenguaje regular. También daremos pinceladas acerca de cómo maneja AWK las expresiones regulares y de la *shell* de Unix. Todo ello nos servirá para poner en práctica estos conceptos e implementar finalmente un ejemplo: en este caso no se trata de pasar de un lenguaje de programación a otro, sino de otra posible

aplicación, modificar un fichero en C/C++ de forma que realice la misma función pero con un estilo de programación diferente.

2. Antecedentes teóricos

2.1. El problema de la parada

"El problema de la parada", también conocido como *Entscheidungsproblem* (en Alemán), es un reto de la lógica matemática de encontrar un algoritmo general que decida si una fórmula de cálculo de primer orden es un teorema, es decir, si esa fórmula de cálculo de primer orden es una verdad demostrable [12]. Constituye uno de las mayores discusiones en teoría de la computación, y puesto que sus resultados son de interés para la herramienta propuesta, realizaremos un repaso de su historia.

Leibniz, importante matemático del siglo XVII, fue el que introdujo inicialmente el problema [12] tras crear una máquina mecánica de cálculo; quiso crear una máquina que, manipulando símbolos matemáticos, pudiera determinar si una lógica era un teorema. Como consecuencia, parte de la aportación de Leibniz a las matemáticas tiene que ver con la lógica simbólica, puesto que para construir esta máquina necesitaba de un lenguaje formal claro y preciso.

Sin embargo, fue David Hilbert quien formalizó la propuesta de Leibniz.

2.1.1. David Hilbert y la principal formulación del problema de la parada

Fue un importante matemático del siglo XIX con importantes aportes a esta ciencia; algunos de los más conocidos son "El Hotel infinito de Hilbert", que da una explicación al concepto de infinito, y los 23 problemas matemáticos que propuso, muchos de los cuales siguen sin resolverse actualmente.

Sin embargo vamos a centrarnos en su relación con "el problema de la parada". Hilbert propuso en 1920 lo que más tarde se conocería como "el programa de Hilbert" [7]; que perseguía el mismo objetivo que la máquina de Leibniz, pero Hilbert especificó algunas condiciones que debía cumplir:

- Al igual que Leibniz, se dio cuenta de que las afirmaciones matemáticas deberían de estar escritas en un lenguaje preciso y formal.
- Las afirmaciones matemáticas deben de ser íntegras, es decir, deben poder ser probadas.

- Las afirmaciones matemáticas deben de ser consistentes, en otras palabras, no pueden ser contradictorias.
- Conservación: las demostraciones de "objetos reales" a partir de "objetos ideales" no pueden ser probadas sin usar "objetos ideales".
- Finalmente, la condición que está más estrechamente relacionada es la decibilidad, que sugiere que debería haber un algoritmo que decidiera la veracidad o falsedad de cualquier verdad matemática.

Tan solo 11 años después, en 1931, Gödel demostró que era irrealizable esta aspiración del "programa de Hilbert".

2.1.2. Kurt Gödel y la primera respuesta negativa al programa de Hilbert

Gödel fue un matemático y lógico austriaco, que publicó dos teoremas conocidos como "Teoremas de incompletitud de Gödel" [6] centrados en la lógica matemática. El enunciado de estos teoremas es:

- Primer teorema: "cualquier teoría aritmética recursiva que sea consistente es incompleta".
- Segundo teorema: "en toda teoría aritmética recursiva consistente *T*, la fórmula consistente *T* no es un teorema".

Fue este último teorema el que afectó directamente al programa de Hilbert, pues venía a decir que "cualquier teoría lo suficientemente consistente como para cifrar la suma y multiplicación de enteros, no puede probar su propia consistencia". De esta forma, acabó con muchos de los puntos del programa de Hilbert:

- No es posible formalizar toda la matemática, ya que cualquier intento dentro del formalismo omitirá ciertas afirmaciones matemáticas verdaderas; rompiendo el primer punto.
- No hay una extensión completa consistente, por lo tanto la mayoría de las teorías matemáticas no están completas; lo que acabó con la idea del segundo punto.
- No existe algoritmo para decidir la veracidad (o probabilidad) de afirmaciones en ninguna extensión consistente del teorema de Peano

(teorema que garantiza la existencia de soluciones para un determinado problema de valores iniciales). Si somos estrictos, esta definición apareció unos años más tarde, pues aun no estaba bien definido lo que era un algoritmo. Este importante punto rompió con la idea de Leibniz y Hilbert.

2.1.3. Tesis de Church-Turing

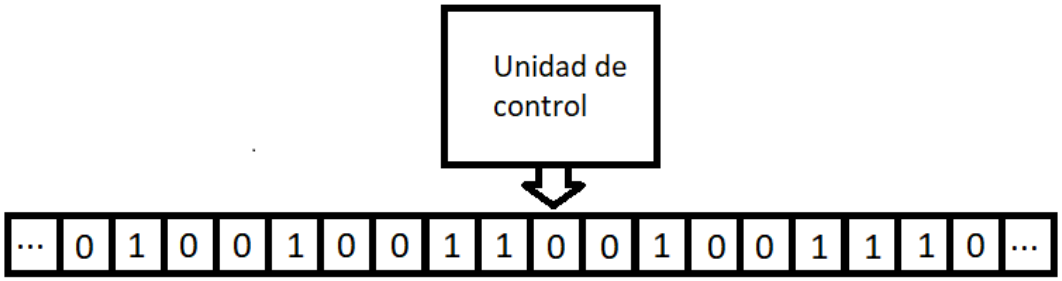
Ya hemos visto como Gödel acabó con la idea de un algoritmo que decidiera si una verdad matemática es cierta o no, rompiendo el problema de la parada. Sin embargo la aplicación de estas ideas al ámbito de la computación así como la definición de qué problemas pueden ser clasificados como computables, fue el fruto de la obra de Alonzo Church y Alan Turing quienes, paralelamente, postularon la conjetura, hoy conocida como la tesis de Church-Turing, que ha contribuido, como ninguna otra lo ha hecho, a la creación de las bases de la ciencia de la computación.

La aproximación de Church al problema se basa en su invención del "cálculo lambda", mientras que Turing lo hizo con su famosa "máquina de Turing" [5]. Aunque ambas formas son válidas, nosotros nos vamos a centrar en la que corresponde a Turing, por tener un planteamiento y unos resultados más "ingenieriles".

2.1.3.1. Alan Turing y la máquina de Turing

Turing es considerado uno de los padres de la ciencia de la computación, cuyo papel tuvo gran relevancia en el desarrollo de la Segunda Guerra Mundial: estaba al mando de una división de la inteligencia británica y con su máquina llamada "Bombe" descifraba los códigos de las máquinas nazis; una de las más conocidas es "Enigma". Turing desarrolló la idea de la máquina de Turing en la década de los 50, y consistía básicamente en una unidad de control de estados finitos, una cinta infinita dividida en casillas con diferentes símbolos y un cabezal que puede leer y escribir en la cinta [4]. La unidad de control a su vez cuenta con diferentes componentes: un registro de estado que guarda el estado actual de la máquina y una tabla de instrucciones que nos dice qué símbolo escribimos, hacia qué lado movemos el cabezal y el siguiente estado.

Un ejemplo del esquema de una máquina de Turing, cuyos símbolos se corresponden con 1 y 0 es:



1. Ejemplo máquina de Turing

Algo tan simple, es esencialmente equivalente a cualquier computadora actual, por lo que nos permite probar problemas de la ciencia de la computación que serían más complicados demostrarlos en las computadoras convencionales, concretamente, nos ayuda a distinguir lo que es computable y lo que no. Aunque pueda parecer tremendamente lenta comparada con una computadora actual, lo cierto es que la diferencia de tiempo frente a las actuales es polinomial [10]. En las máquinas de Turing, se le conoce a δ como función de transición, que nos indica el siguiente estado de la máquina, lo que escribirá en la casilla en la que está situado el cabezal y hacia qué lado se moverá (derecha o izquierda), en función del estado actual y del símbolo de la casilla; su notación [12] es la siguiente:

$$\delta(q,X)=(p,Y,D)$$

Siendo:

- q, el estado actual.
- X, el símbolo de la casilla actual.
- p, el estado siguiente.
- Y, el símbolo que escribiremos en la casilla.
- D, desplazamiento hacia la derecha o hacia la izquierda en la cinta.

2.1.3.1.1. Ejemplo de funcionamiento

En este ejemplo, se pretende hacer el complemento a 1 de un número en binario, de forma que el que el cabezal empieza en el primero de los bits (el que está más a la izquierda), y las casillas por delante de él (y por detrás del último bit) están en blanco.

Vamos a tener dos estados, uno q_0 en el que realizaremos la conversión, y uno q_1 en el que finalizaremos la ejecución. Definimos la función de transición de este ejemplo:

$$\delta(q_0, 0)=(q_0, 1, R)$$

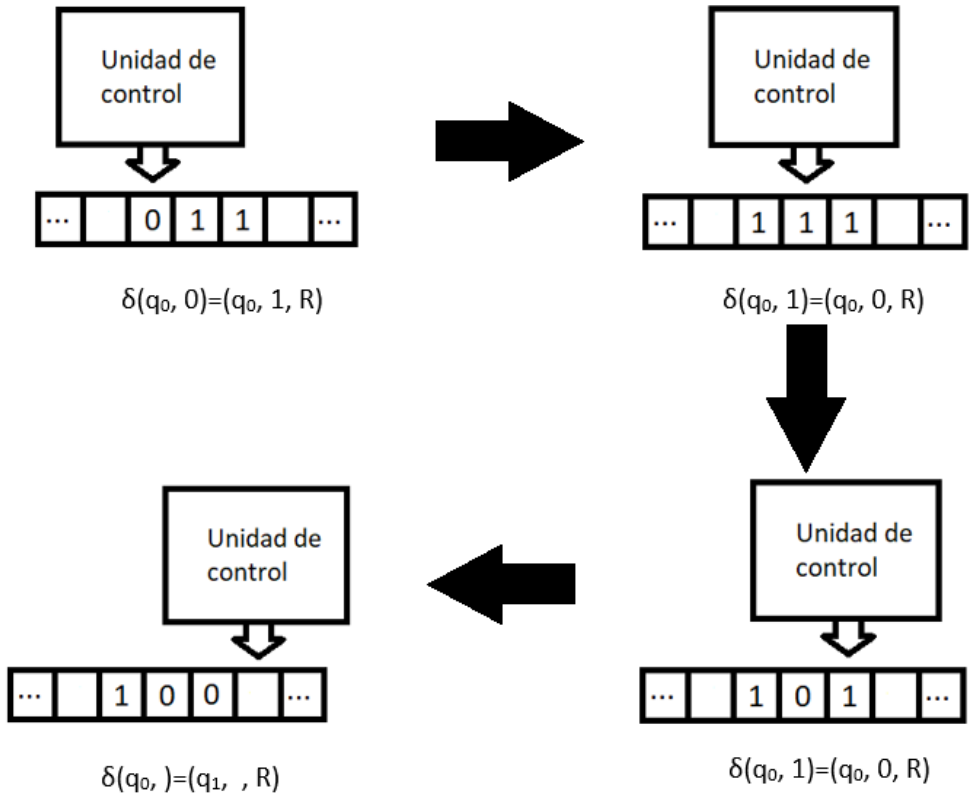
Modificamos el 0 por el 1 y nos movemos hacia la derecha, manteniendo q_0 como el estado actual.

$$\delta(q_0, 1)=(q_0, 0, R)$$

Modificamos el 1 por el 0 y nos movemos hacia la derecha, manteniendo q_0 como el estado actual.

$$\delta(q_0,)=(q_1, , R)$$

Al encontrar un símbolo blanco (ausencia de símbolo al fin y al cabo), los cambiamos por otro símbolo en blanco y nos movemos a la derecha, cambiando al estado q_1, que como no está definido finalizará el programa. Si tenemos 3 bits:



2. Imagen correspondiente al complemento a 1 de un número mediante la máquina de Turing

Con estas mismas instrucciones podríamos tratar números de un número cualquiera de bits, pues como hemos comentado, la cinta es infinita.

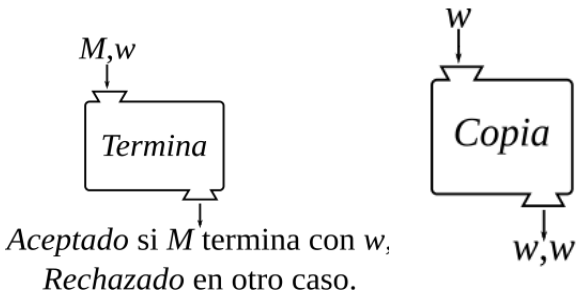
2.1.3.1.2. La máquina de Turing y el problema de la parada

Turing ideó esta máquina para estudiar si la idea de Hilbert acerca de la decibilidad de las matemáticas era posible o no, además, demostró que había problemas que no se podían resolver, a partir de ahí se empezó a distinguir entre problemas P y NP.

Aplicado a la máquina de Turing, el problema de la parada cambia un poco de forma, pero el significado se mantiene: ¿puede una máquina de Turing saber si otra máquina de Turing se detiene? A esta pregunta Turing respondió negativamente. Vamos realizar una pequeña comprobación, obtenida de la referencia [8] de la bibliografía:

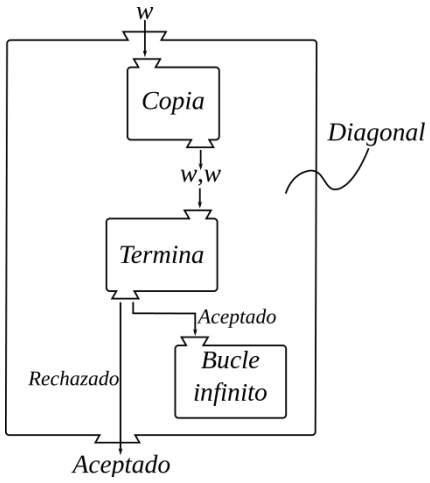
Tenemos una máquina de Turing que llamaremos *M* (que contiene una 7-upla que la describe completamente, pero no hemos tratado en este proyecto) a la que alimentamos con una cinta marcada con determinados símbolos (w).

Vamos a suponer que existe una máquina de Turing a la que dándole como entrada la configuración de otra máquina de Turing (*M*) y la entrada de esta última (*w*), termina en estado de aceptación si nuestra máquina ha parado ante su entrada; llamaremos a esta máquina "termina". También utilizaremos otra máquina "copia", que teniendo como entrada en su cinta una cadena *w* termina en su cinta con *w, w*.



3. Máquinas "termina" y "copia" mencionadas

Podemos crear a partir de estas máquinas otra máquina de Turing con la siguiente estructura:



4. Esquema de la máquina de Turing resultante

Diremos que esta máquina de Turing es una máquina "*D, w*", es decir, tiene *D* como descripción y *w* como entrada. Pues bien, el funcionamiento de esta

máquina es el siguiente: en primer lugar duplicamos la entrada *w* con "copia", cuya cinta nos sirve de entrada para "termina"; estamos alimentando a "termina" con "*w, w*"; es decir, w es una codificación de una máquina de Turing, cuyos símbolos tenemos también como entrada. Si "termina" acaba en un estado de aceptación, entramos en un bucle infinito, en caso contrario, "Diagonal" termina en estado de aceptación. Por lo tanto:

$$\text{"}D\text{iagonal" para ante la entrada } w \Leftrightarrow \text{La máquina codificada en } w \text{ no para ante la entrada } w$$

Si ahora tomamos la codificación de "Diagonal" como su propia entrada, es decir ahora es "*D, D*" en vez de "*D, w*" podemos concluir:

$$\text{"}D\text{iagonal" para ante sí misma como entrada} \Leftrightarrow \text{"Diagonal" no para ante sí misma como entrada}$$

Esta paradójica conclusión pone en entredicho el funcionamiento de nuestra máquina, y dado que todos los componentes son realizables salvo la máquina "termina" que hemos supuesto, podemos concluir por reducción al absurdo que no existe una máquina de Turing capaz de decidir si todas las máquinas de Turing terminan, por lo que el problema de la parada no es aplicable a máquinas de Turing.

2.2. Tipos de lenguajes formales

La máquina de Turing y con más razón, los ordenades actuales y los lenguajes de programación de propósito general, pueden resolver todos los problemas que se consideran computables. Existen, sin embargo muchos problemas prácticos que no requieren la potencia completa que representa este formalismo, y que con herramientas menos potentes se puede conseguir el mismo resultado; estas herramientas menos potentes son, además, mas robustas, por lo que nos interesa escoger la herramienta adecuada para cada tipo de lenguaje. En este apartado, hablaremos de los diferentes tipos de los 4 tipos de gramáticas formales (que generan lenguajes formales) que diferenció el lingüista Noam Chomsky en la década de los 50, que actualmente

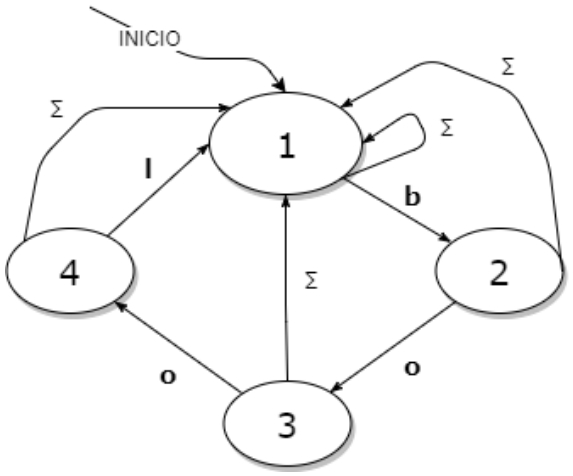
conocemos como “La Jerarquía de Chomsky” [10] y que nos ayudarán a distinguir qué podemos hacer con un lenguaje y qué no:

0. Lenguaje recursivamente enumerable: es el nivel más alto y engloba a todos los lenguajes capaces de ser reconocidos por una máquina de Turing, es decir, puede simular la lógica de cualquier algoritmo de computador. Cuando un lenguaje pertenece a este tipo, decimos que es Turing-Completo o “completo en el sentido de Turing”.
1. Lenguaje dependiente del contexto: se refiere a los lenguajes que pueden ser interpretados mediante un autómata linealmente acotado, esto es: una máquina de Turing que no cuenta con una cinta de infinitas posiciones, sino $K \cdot n$ posiciones (siendo n el tamaño de la entrada y K una constante de la máquina). Es el tipo menos presente, pero se usan por ejemplo para definir la sintaxis de los lenguajes de programación (*yacc* de Unix).
2. Lenguaje independiente del contexto: aquellos lenguajes que pueden ser simulados mediante un autómata con una pila.
3. Lenguaje regular: el lenguaje más básico de todos, se simula con un autómata.

5. Tipos de lenguajes formales

Conociendo los diferentes tipos de lenguajes existentes, podemos centrarnos en lo que mencionábamos anteriormente, ¿qué podemos resolver con cada lenguaje? La respuesta puede ser obvia si nos fijamos en la imagen superior, pues los lenguajes se van incluyendo entre ellos como una *matrioshka*, pero, ¿con qué lenguaje es mas óptimo resolver un problema?.

Vamos a centrarnos en los lenguajes regulares, pues como ya hemos mencionado, es uno de los grandes atractivos del lenguaje AWK. Vamos a suponer que queremos buscar la palabra reservada *bool* del lenguaje C, y sustituirla por *boolean* del lenguaje Java. El autómata que *matchearía* esta expresión regular sería:



6. Ejemplo autómata

Donde "$\sum$" indica que encaja con cualquier otro carácter.

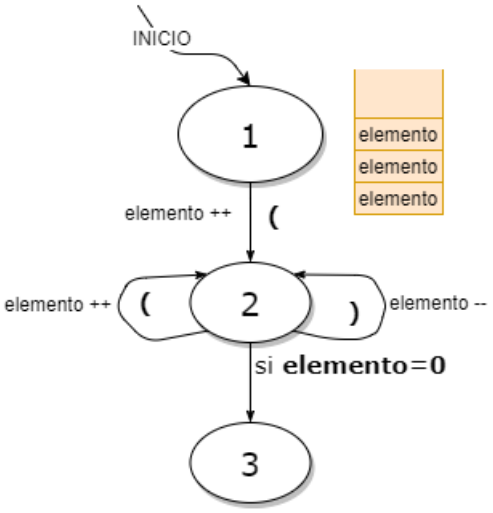
Pero qué ocurre si en vez de esto, queremos *matchear* una línea que contenga el mismo número de paréntesis de cierre que de apertura. Para comprobar si un lenguaje es regular, podemos recurrir al "lema del bombeo" [10], mas conocido como "*pumping lemma*". El lenguaje que queremos identificar si es regular o no, es el siguiente:

$$(^n)^n, \qquad n \geq 1$$

Si este lenguaje fuese regular, sería el lenguaje de algún autómata, que suponemos de *K* estados. Después de leer los primeros *K+1* paréntesis de apertura de la entrada, el autómata estará en un estado determinado, que por ser mayor que *K* podemos deducir por el el principio del "juego de las sillas" [10] que después de leer dos prefijos diferentes "(^{i} y (^{j}", el autómata se encuentra en el mismo estado *q*. Sin embargo, después de recibir *i* paréntesis de apertura, comienzan a llegar paréntesis de cierre; de forma que debe aceptar si previamente ha recibido *i* paréntesis de apertura, pero no *j*. Como al

llegar *i* o *j* paréntesis de apertura, se encontraba en el mismo estado, no puede decidir si aceptar o no la entrada como válida.

Este caso descrito obedece a un autómata con pila, de forma que los estados no solo dependen de la entrada, sino también del estado de la pila: cuando encontramos un paréntesis de apertura, introducimos un elemento en la pila, y cuando encontramos un paréntesis de cierre, eliminamos un elemento de la pila. El autómata se inicia cuando encuentra un paréntesis de apertura y finaliza cuando la pila está vacía:



7. Ejemplo de autómata a pila

Este es solo un ejemplo de las muchas ocasiones a las que nos enfrentaremos, que querremos resolver con expresiones regulares, pero nos será imposible; el lema del bombeo nos ayudará a evitar estrujarnos el cerebro intentando resolver estos casos mediante expresiones regulares, de este modo podremos descartarlos. Muchos de ellos, aunque pertenecen a lenguajes de menor nivel (como los lenguajes independientes del contexto) no conocemos las herramientas que trabajan con estos lenguajes (*Yacc*), por lo que usaremos un lenguaje superior que si conozcamos: el AWK, pues como ya hemos dicho, es Turing-Completo y basado en C.

3. Herramienta de transformación de código fuente

Una vez conocemos los antecedentes a nuestro propósito, que nos han servido para conocer qué podemos resolver y que no, podemos centrarnos finalmente en el desarrollo de la herramienta.

En primer lugar realizaremos una introducción a los lenguajes que hemos empleado, para conocer mejor sus posibilidades y comportamiento; en segundo lugar hablaremos sobre la herramienta (sus características, funcionamiento, etc) para finalmente ver algunas aplicaciones.

3.1. Lenguajes empleados en la implementación

Para crear la herramienta utilizaremos dos lenguajes: AWK y el *Shell* de Unix, puesto que en nuestro caso utilizaremos el AWK implementado en Unix. Antes de centrarnos en el AWK, hablemos un poco de Unix.

3.1.1. El sistema operativo Unix

Unix ha sido el sistema operativo mas influyente de la historia; la mayoría de los SO actuales incorporan ideas o conceptos que nacieron con Unix [13]. Fue desarrollado por miembros de laboratorios Bell de AT&T y reescrito en multitud de ocasiones para adaptarlo a necesidades concretas; quizá la mas conocida sea la de Linus Torvalds, que lo reescribió desde cero y lo llamó “Linux”, recibiendo mas tarde el apoyo del proyecto GNU de Richard Stallman (que pretende que el software sea libre), conociéndose ahora como “GNU/Linux” [17].

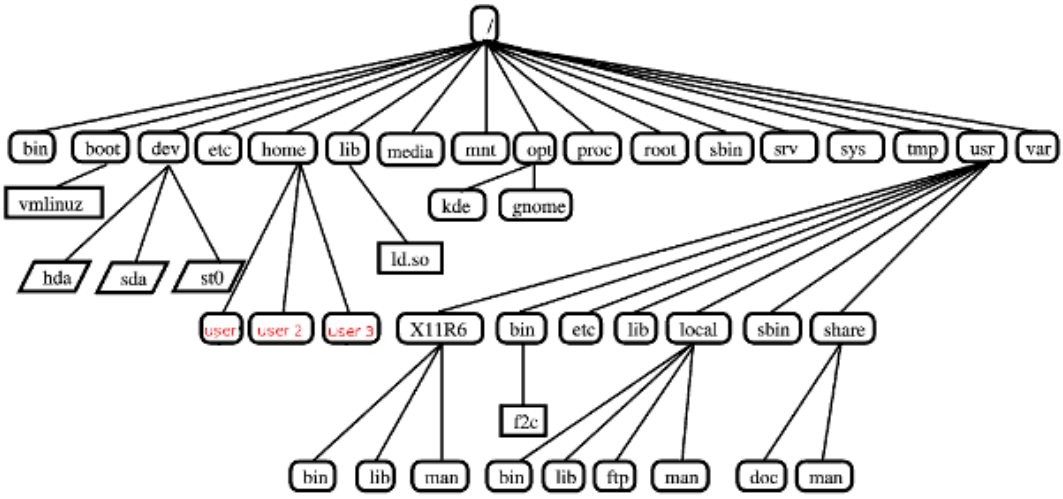
Unix es un sistema operativo multitarea y multiusuario [17], esta última es una de las características más destacables de Unix: un sistema operativo multiusuario es aquel que pemite proveer servicio procesamiento a múltiples usuarios simlutáneamente, de modo que comparten recursos como: procesador, memoria, programas, periféricos… De este modo, podremos guardar nuestro transformador de código fuente en un equipo, y que multitud de usuarios se conecten y lo lancen simultáneamente.

Una de sus características mas importantes es la de crear programas rápidos, simples y con poca de intervención por parte del usuario; las *pipes* o “filtros”,

son un buen ejemplo de ello: programas que leen sus datos de la entrada estándar (típicamente asociadas al teclado pero que se pueden redirigir a que sean la salida de otro programa) y que escriben en la salida estándar (asociados a la pantalla pero que también puede ser redirigido), no sin antes modificar en algún sentido los datos (filtrarlos de alguna manera). Este hecho de poder enlazar *pipes* simplifica y garantiza la correcta ejecución de multitud de órdenes.

Unix trae consigo una "filosofía" [2], es decir, una forma de trabajo definida y bastante relacionada con las *pipes* que hemos mencionado anteriormente. Según esta filosofía, un programa debe de realizar únicamente la tarea para la que fue creado y hacerla bien, realizando el mayor número de tareas posibles por unidad de tiempo; y no solo eso, sino que nuestro programa debe de tener en cuenta que en algún momento su salida puede ser entrada de otro programa, por lo que debemos de configurarlo como tal.

Otra de las partes mas importantes de Unix es su manejo de archivos [14]. Un archivo en Unix puede ser: archivos de texto, directorios y descriptores de fichero. Los archivos de texto en Unix se almacenan en directorios, de forma que dentro de estos puede haber mas directorios, llamados subdirectorios; sin embargo, el directorio mas importante es el directorio raíz (representado por el símbolo "/"), que como su propio nombre indica, es el directorio principal y el único que no tiene un directorio por encima de él. Fruto de esta organización de directorios se obtiene lo que se conoce como estructura de "árbol invertido". Los archivos de texto, traen consigo una serie de permisos (diferenciando entre usuario, grupo y otros), que nos indican las acciones que podemos realizar con ese archivo: lectura, escritura y ejecución en el caso de archivos, y lectura, escritura y acceso en el caso de directorios.



8. Estructura de árbol invertido en directorios de Unix

Por otro lado tenemos el *Shell*, el intérprete de comandos de Unix, que contituye la interfaz entre el usuario y el *kernel* (o núcleo) de Unix. Gracias a él podemos realizar multitud de funciones: desplazarnos entre directorios, comprobar los permisos de los archivos, lanzar los scripts de Unix (incluido el AWK), etc... Puesto que los comandos que vamos a utilizar son demasiado grandes, utilizaremos lo que se conoce como *script*, que no es mas que un archivo que contiene comandos de el *Shell* que se ejecutan secuencialmente.

3.1.2. El lenguaje AWK

AWK es un lenguaje de programación basado en C, creado, nuevamente, por miembros de los laboratorios Bell de AT&T y rediseñada por GNU en una versión que conocemos como “gawk” [1]. Esta especialmente diseñado para trabajar con archivos estructurados y patrones de texto con precisión gracias a su manejo de expresiones regulares. Las expresiones regulares permiten implementar multitud de opciones de gran utilidad, como búsqueda de palabras, sustitución de expresiones, contaje de nombres, etc. Ya hemos hablado de las expresiones regulares (lenguajes regulares) cuando hablamos de los lenguajes formales de Chmosky.

La función básica de AWK es buscar líneas en ficheros de texto que contienen ciertos patrones. Uno de los factores mas importantes a tener en cuenta es que AWK siempre buscar el patrón mas largo y situado mas a la izquierda que encuentra en la línea [9]. Cuando AWK encuentra el patrón en una línea, realiza

las acciones especificadas y continua leyendo las líneas de entrada hasta el final del último fichero de entrada, pues podemos leer varios ficheros sin necesidad de pasarle uno a uno al AWK. Si no aplicamos ningún programa a la entrada del AWK, este leerá por defecto la entrada estándar.

Otro aspecto importante es la estructura de un programa AWK. En el caso mas general, un programa AWK consta de 3 secciones diferenciadas [9]: "BEGIN {…}", "{…}" y "END {…}"; en "BEGIN {…}" se ejecutarán las instrucciones antes de que comencemos a procesar la entrada, en esta sección podemos inicializar variables, indicar por pantalla la inicialización del programa y otras muchas tareas. Dentro de las llaves centrales ("{…}") se encuentra todo el código que se ejecutará de forma cíclica, una vez por cada línea de entrada. Una vez que hemos tratado todas las línea de entrada, y por tanto ha finalizado el apartado "{…}", pasamos a la sección "END {…}", dónde las instrucciones se ejecutarán una vez y finalizará el programa.

Si bien hemos dividido la estructura en 3 secciones, bien podríamos haberlo hecho en 4, puesto que las declaraciones de las funciones AWK, se realizan en las líneas superiores a la sentencia "BEGIN {…}". Finalmente, todo ello va precedido de la sentencia que inicializa el programa: awk –OPCIONES 'BEGIN {…}{…} END {…}'.

Puede obtener más información acerca del funcionamiento del AWK en el manual de GNU [9].

3.2. Implementación de la herramienta

Una vez que hemos justificado el uso de AWK y sabemos que método aplicar para saber si podemos *matchear* un patrón con expresiones regulares o no, es el momento de hablar de la herramienta desarrollada. Esta herramienta tiene como filosofía principal su flexibilidad en cuanto al manejo de diferentes tipos de lenguajes de entrada y salida, de modo que para modificarlo haya que realizar los mínimos cambios posibles. Para ello usaremos *scripts* en *bash* (una *Shell* de GNU) en los que incluiremos GAWK, una extensión de AWK, nuevamente, de GNU. Tanto *bash* como la extensión de GNU de AWK están incluidos en la mayoría de consolas (en caso contrario, se pueden instalar) por

lo que no deberíamos de tener problema a la hora de lanzar la herramienta en nuestro sistema.

Cabe destacar también, que aunque no se tenga un SO Unix, es posible utilizar la herramienta con otros programas que simulan una consola Unix, como Cygwin, una aplicación gratuita.

3.2.1. Estructura

Como ya hemos comentado, la herramienta está formada por *scripts* que realizan diferentes funciones:

- *lanzador_TC.sh*: se ocupa de administrar todo lo necesario para la ejecución de la herramienta, así como ejecutar otros *scripts* hijos: *lib_TC.sh*, *detector_TC.sh* y *corrector_TC.sh*.
- *lib_TC.sh*: contiene las funciones de *bash* que emplean los *scripts*.
- *detector_TC.sh*: contiene el AWK que detecta las estructuras que queramos modificar.
- *corrector_TC.sh*: corrige las estructuras detectadas por *detector_TC.sh*.

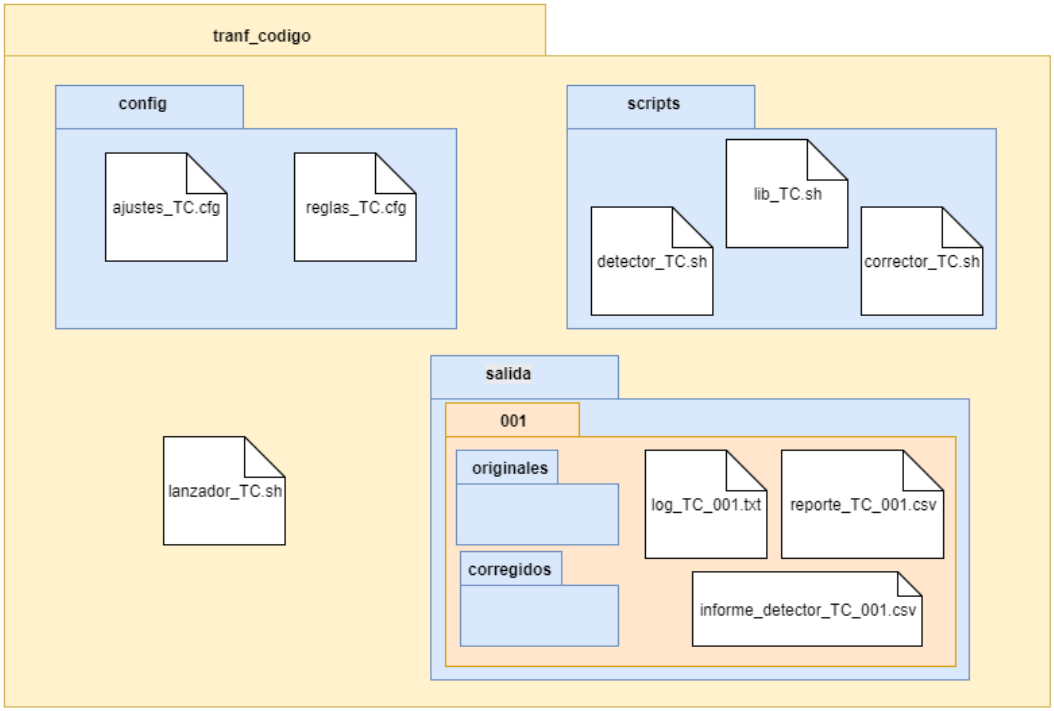
Excepto *lanzador_TC.sh*, todos los *scripts* están incluidos en el directorio "scripts". A su vez, tenemos dos ficheros de configuración, incluidos en el directorio "config":

- *ajustes_TC.cfg*: en este fichero se especifican algunas de las variables que sirven de entrada a nuestra herramienta; las comentaremos más adelante.
- *reglas_TC.cfg*: aquí especificamos las reglas, las cuales hemos definido previamente, que queremos que sean ejecutadas.

Finalmente contamos con dos directorios: "entrada" y "salida"; en ellos, por defecto, se encuentran los ficheros que queremos modificar, y los ficheros modificados (respectivamente). Veremos más adelante cómo es posible cambiar estos directorios de entrada-salida. En el directorio "salida" se crea un directorio cuyo nombre es un número que llamaremos "secuencia" o "secuencia de ejecución", que contendrá:

- Un fichero de *log* con la información de la ejecución, llamado *log_TC_*nºsecuencia*.txt*.
- Un fichero llamado *informe_detector_TC_*nºsecuencia*.csv*, que es la salida de *detector_TC.sh* y la entrada de *corrector_TC.sh*.
- En el caso de que hayamos definido variables que generan algún reporte (veremos cómo), se crea el fichero *reporte_TC_*nºsecuencia*.csv*.
- Si almacenamos los ficheros modificados en "salida", se crean dos subdirectorios: "originales", que contiene los ficheros originales duplicados, y "corregidos", que contiene los ficheros modificados.

De esta forma tenemos una estructura semejante a la imagen inferior:



9. Estructura de la herramienta para secuencia de ejecución 1

3.2.2. Funcionamiento

En este apartado vamos a comentar qué funciones desempeña la herramienta y, en particular, cada uno de los archivos que la componen. Como comentábamos en la introducción, la herramienta está diseñada con la flexibilidad como objetivo prioritario; esto implica que para aplicarla, por ejemplo, a una modificación de Java a C++ debemos realizar ciertas

modificaciones en el código. Para mayor comodidad, se han etiquetado estas modificaciones como "#MOD_REQ#"; insistimos en que esta etiqueta indica que modificar esa línea de código es vital para el correcto funcionamiento de su aplicación.

Otro aspecto importante a destacar es que la herramienta es unidireccional, es decir, podemos transformar un lenguaje A en un lenguaje B, pero no un lenguaje B en un lenguaje A (aunque teniendo el primer caso podemos obtener fácilmente el segundo caso configurando la herramienta de modo que lo que corregimos en el primer caso, es lo que buscamos en el segundo; lo que buscamos en el primero, es lo que corregimos en el segundo).

$$Lenguaje\ A \rightarrow Lenguaje\ B \not\equiv Lenguaje\ B \rightarrow Lenguaje\ A$$

La flexibilidad de la aplicación hace que en la parte *bash* de los *scripts* haya que realizar pocas modificaciones (comentadas anteriormente), pero eso no implica que no pueda retocarlo tanto como le convenga. Sin embargo es en el código AWK donde se deben codificar las reglas de la aplicación, por lo que comprender el funcionamiento de *detector_TC.sh* y de *corrector_TC.sh* es necesario para codificarlas correctamente.

3.2.2.1. lib_TC.sh

Se ha decidido comenzar la explicación por este *script* porque no realiza ninguna función especial en la herramienta, simplemente contiene funciones y variables en *bash* que utilizan el resto de *scripts* y que han sido agrupadas en este fichero para dar mayor claridad al código.

3.2.2.2. lanzador_TC.sh

Este *script* se ocupa de iniciar la ejecución de la herramienta; es, por tanto, el que ejecutamos en la consola. Una vez ejecutado lo primero que hace es comprobar que existe el directorio "scripts" y el *script lib_TC.sh*, de no ser así generaría un error y acabaría la ejecución del programa, hablaremos en profundidad de los errores próximamente. Comprueba además que *lib_TC.sh* tiene permisos de ejecución, y en caso contrario se los otorga y lo ejecuta, de esta forma ahora tenemos acceso a sus funciones y variables.

A continuación comprueba que existen los directorios “config”, “entrada” y “salida” (si no existen estos dos últimos, simplemente los crea) y que existen los archivos *ajustes_TC.cfg* y *reglas_TC.cfg*. Lo mismo ocurre con los *scripts detector_TC.sh* y *corrector_TC.sh*, comprueba que existen y además, que tengan permisos de ejecución.

Es ahora cuando se carga el fichero de configuración *ajustes_TC.cfg*, a la vez que se comprueba que los valores de las variables que contiene son válidas:

- RUTA_FICHEROS: contiene el PATH del directorio donde se encuentran los ficheros que queremos modificar. Se comprueba que se ha asignado un valor a la variable y que el directorio especificado existe; se buscan los ficheros con una extensión determinada (este es un ejemplo de código marcado con “#MOD_REQ#”). Es importante destacar también que RUTA_FICHEROS no puede ser el directorio de la herramienta.
- SOBRESCRIBIR: mediante esta variable se pueden eliminar los archivos originales y crear los modificados en el mismo PATH, de forma que no aparecen en el directorio “salida”. Debe de estar a “SI” o “NO”. Por motivos de seguridad (eliminar código de forma equívoca), se ha añadido una confirmación por pantalla para asegurarnos de que realmente se desean sobrescribir los ficheros en el caso de que la asignemos el valor “SI”.
- SECUENCIA: contiene el número de la secuencia de ejecución, que se incrementa en 1 en cada nueva ejecución de la herramienta; es de gran utilidad para poder lanzar la herramienta varias veces sin sobrescribir los ficheros generados, así como para la mejora iterativa de la aplicación a la que se esté destinando la herramienta. Comprobamos que está inicializada a un valor numérico, en cuyo caso se incrementa en 1 el valor y se continúa con la ejecución del *script*.

Los valores de estas variables de configuración los almacenará para utilizarlos cuando sea necesario.

Lo siguiente que hace el *script* es pasar los ficheros encontrados en RUTA_FICHEROS a formato Unix y comprobar que los parámetros de ejecución son correctos, se admite:

- -D: modo solo detección. Su función principal es la de probar la herramienta en la mejora iterativa de la aplicación.
- -DC: modo detección y corrección. Es el modo de ejecución más habitual: se detectan y corrigen las estructuras.
- -R: modo solo reportes. Tan solo se ejecutan las reglas que generan un reporte (en caso de que se hayan creado como tal). Las reglas que generar reporte deben de ser siempre las últimas en la lista de reglas (es decir, si tenemos 5 reglas y vamos a configurar 2 reglas de reporte, deberán de ser las reglas 6 y 7). Para modificar el número de reglas de reporte, podemos hacer en la variable NUM_REG_REP (marcada con la etiqueta).
- –help: muestra la ayuda de las diferentes opciones de modos de ejecución.

A continuación se carga el otro fichero de configuración, *reglas_TC.cfg*, y comprueba que las reglas están a "SI" o "NO" (en caso contrario salta un error y finaliza el programa). Almacena en un vector un 1 si la regla está activa y un 0 en caso contrario. Finalmente, se crea el fichero con la secuencia de la ejecución actual en "salida", de modo que si esa carpeta ya existe, se sobrescribe.

Una vez finalizadas todas las comprobaciones, se ejecuta la siguiente instrucción en función del parámetro de ejecución:

- -D: se llama a *detector_TC.sh* y después, finaliza la ejecución de la herramienta
- -DC: se llama a *detector_TC.sh* y *corrector_TC.sh* secuencialmente y después, finaliza la ejecución de la herramienta.
- -R: se ponen a 0 todas las reglas que no sean de reporte, se llama a *detector_TC.sh* y después, finaliza la ejecución de la herramienta.

3.2.2.3. detector_TC.sh

Es llamado por *lanzador_TC.sh* y contiene el AWK con la lógica de detección de las reglas. Tiene tres ficheros de salida: el archivo de *log* (*log_TC.txt*), el archivo de los posibles reportes generados (*reporte_TC.csv*) y finalmente, el fichero en el que aparecen las líneas en las que hemos detectado alguna de las estructuras que buscamos (*informe_detector_TC.csv*); por otra parte tiene como entrada los archivos a modificar.

Estos dos últimos ficheros, al ser de extensión *.csv,* consiste en columnas separadas por el carácter punto y coma; además, tienen una determinada estructura. *informe_detector_TC.csv* tiene cuatro columnas, que por orden son: el nombre del fichero dónde hemos detectado la estructura, la línea de ese fichero en la que lo hemos detectado, la regla que ha detectado esa estructura y, finalmente, la subregla, que constituye una forma de distinguir entre diferentes casos bajo la misma regla. *reporte_TC.csv* contiene cinco columnas: las dos primeras son "fichero" y línea" (con el mismo significado que en el fichero anterior), la tercera es "regla" en la viene incluida el número de regla que ha generado el reporte, justo después tenemos "regla reporte", que no es más que otra forma de llamar a la regla, finalmente en "incidencia" encontramos un texto que nos explica el origen de la advertencia. En *reporte_TC.csv* podemos incluir, por ejemplo, información para que el usuario pueda modificar estructuras que serían complicadas de implementar en la herramienta. Para clarificar la diferencia entre "regla" y "regla reporte", vamos a poner un ejemplo: tenemos 8 reglas en total, 3 de las cuales son reglas que generan reporte; como estas reglas deben de ser las últimas, las tres reglas que generan reporte serán las reglas 6, 7 y 8; por tanto en la columna "regla", para referirnos a la regla reporte 1, escribiremos un 6 (y en "regla reporte" escribiremos un 1).

3.2.2.4. corrector_TC.sh

Es también llamado por *lanzador_TC.sh* y contiene el AWK con la lógica de corrección de los ficheros a modificar. Antes de ejecutar el AWK, comprueba que el fichero *informe_detector_TC.csv*, contenga al menos una modificación a realizar, en caso contrario se cierra indicando que no hay modificaciones. Tiene

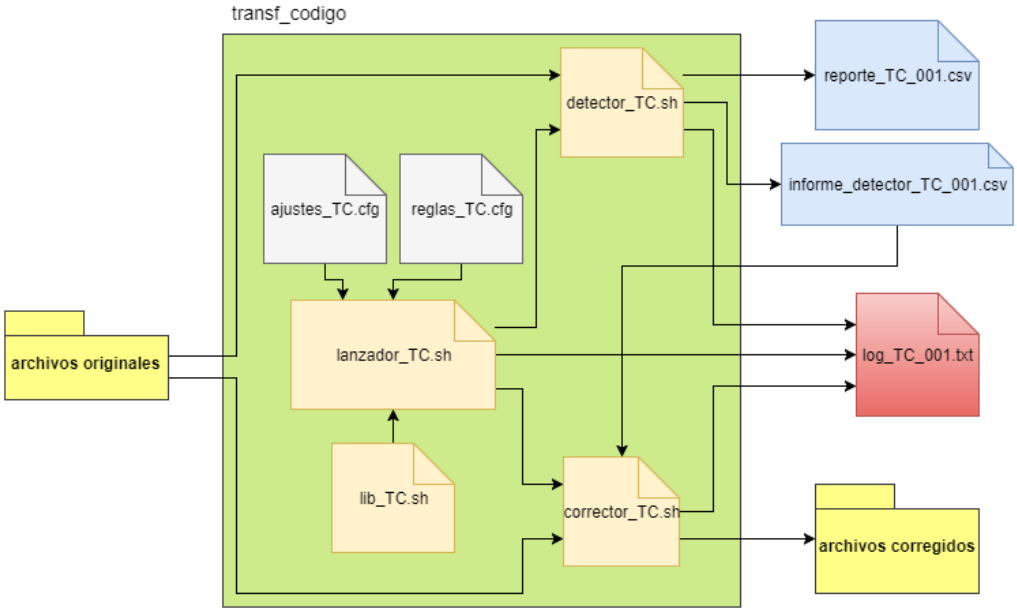
como entrada el fichero *informe_detector_TC.csv* y los archivos a modificar y como salida los archivos modificados y el fichero de *log* (*log_TC.txt*). Este fichero contiene las expresiones regulares que se aplican en cada subregla para corregir la estructura localizada en un determinado fichero y en una determinada línea. Las subreglas son útiles, por ejemplo, a la hora de localizar una estructura de un código que ocupa varias líneas, ya que como ya hemos comentado, el AWK lee las líneas una a una.

3.2.2.5. Resumen de funcionamiento

El *script lanzador_TC.sh* se ocupa de "preparar el terreno" antes de tratar los archivos originales, escribiendo la información de la ejecución en *log_TC.txt*, y en función de los parámetros de ejecución realiza diferentes secuencias:

- -D: llama a *detector_TC.sh*, que lee los archivos originales en busca de las estructuras definidas con las expresiones regulares, que una vez detectadas las documenta en el archivo *informe_detector_TC.csv* o *reporte_TC.csv* en función de si son incidencias que queremos modificar tan solo informar de que se han encontrado, escribiendo también la información relevante en *log_TC.txt*.
- -DC: se llama a *detector_TC.sh*, que realiza las mismas funciones antes mencionadas, y después se llama a *corrector_TC.sh*, que lee *informe_detector_TC.csv* y modifica las líneas de los ficheros que aparecen reflejadas, según la subregla. Escribe en *log_TC.txt* lo que sea necesario.
- -R: se desactivan las reglas que no sean de reporte y se llama solo a *detector_TC.sh*, de modo que tan solo se generará el fichero *reporte_TC.csv*.
- --help: muestra por pantalla la ayuda de la ejecución, que contiene las opciones antes descritas.

A continuación se muestra un diagrama que muestra gráficamente el funcionamiento de entradas-salidas:



10. Diagrama entradas/salidas de la herramienta

3.2.3. Ejecución herramienta

En primer lugar, copie el directorio "transf_codigo" en el lugar en el que desee ejecutar la herramienta, asegúrese de que dentro de este directorio se encuentran los directorios "config" y "scripts" y el fichero *lanzador_TC.sh*; a su vez compruebe que en el directorio "config" se encuentran los archivos *ajustes_TC.cfg* y *reglas _TC.cfg* y en "scripts" están los archivos *corrector_TC.sh*, *detector_TC.sh* y *lib_TC.sh*. Una vez que ha comprobado que todos los archivos están en su directorio correspondiente, ejecute el comando: "*which bash*", debe haber aparecido una ruta en la pantalla, en ese caso compruebe que la ruta coincide con la que aparece en *lanzador_TC.sh* precedido por un "#!" (y marcada con la etiqueta "#MOD_REQ#"). Cambie la ruta por la que ha mostrado su terminal en caso de que no coincidan. Si por el contrario ha aparecido en el terminal un mensaje del estilo de: "*which: no bash in…*" debe instalar *bash* en su máquina.

A continuación sitúese en el directorio "transf_codigo" y ejecute el siguiente comando para dar permisos de ejecución a *lanzador_TC.sh*: "*chmod +x lanzador_TC.sh*".

Finalmente para ejecutar la herramienta escribiremos con el terminal en el directorio "transf_codigo":

"./lanzador_TC.sh [OPCIÓN]"

Donde [OPCIÓN] es el parámetro de ejecución comentado en el apartado anterior.

Finalmente, se recomienda tener instalado el comando *"dos2unix"* para que la herramienta transforme su código automáticamente a formato Unix; si no lo tiene instalado, le aparecerá una notificación cuando ejecute la herramienta, pero la ejecución continuará.

3.2.4. Adaptación de la herramienta a una aplicación concreta

En este apartado comentaremos cómo convertir el esqueleto de la herramienta en un conversor de código en concreto (que puede encontrar en el "Anexo I. Código esqueleto de la herramienta". Para ello, obviamente, necesitamos conocimientos de AWK; por otro lado la parte en *bash*, que constituye la mayoría del esqueleto, es funcional por lo que solo serían necesarios los retoques que el programador convenga oportunos (y los cambios etiquetados con "#MOD_REQ#").

Antes de nada, se recomienda leer y entender el código esqueleto, de esta forma podremos identificar mejor las dificultades que pueden suponer las reglas cuando comencemos a plantearlas. Una vez hecho esto, y sabiendo los dos lenguajes que queremos transformar, debemos pensar las estructuras que queremos identificar, y las estructuras que queremos sustituir. Cuando tengamos una larga lista (no necesariamente completa) de cambios pensados, es recomendable plantearse qué cambios son parecidos (cuáles podrían llevar expresiones regulares comunes que los identifiquen), de esta forma se aprovecharán expresiones regulares que pueden llegar a ser largas; se recuerda que existen las subreglas, por lo que no hay problema en tener diferentes arreglos para estructuras parecidas. Cuando tengamos una idea clara de qué queremos modificar, con las reglas bien definidas, podemos empezar a programar los *scripts*, realizando las modificaciones necesarias tanto en la parte *bash* como en los archivos de configuración *ajustes_TC.cfg* y

reglas _TC.cfg. Cada AWK tiene una lógica diferente por lo que vamos a comentarla por separado:

- *detector_TC.sh*: tenemos, en la parte de procesamiento que se ejecuta en bucle, diferentes bucles *if* que comprueban si las diferentes reglas están activas. Para que sirva como guía estructural, se han creado dos bucles *if* vacíos de una supuesta regla 1 y una supuesta regla 2. En caso de que una regla esté activa, se ejecutan sus sentencias; es ahí donde entra la lógica de las expresiones regulares, que en caso de *matchear* las diferentes alternativas usaremos la función de AWK *log_awk*, creada para enviar las incidencias a los archivos de extensión *.csv*, así como la información necesaria a *log_TC.txt*.
- *corrector_TC.sh*: la parte más importante del AWK de este *script* reside en una función de AWK llamada *genera_array_arreglos*. Esta función contiene *arrays*, que son inicializados a un valor; este valor es en un primer caso una expresión regular, que por ser una cadena de caracteres (está situada entre comillas en vez de entre barras inclinadas) recibe el nombre de *string constant*, en el otro caso es una sustitución de los *matcheos* con la expresión regular. Vamos a centrarnos en estudiar estos *arrays*.

  Lo primero que debemos saber es que tienen tres dimensiones, y siguen la siguiente estructura:

  arreglos[subregla][índice_subregla][tipo_cadena]=""

  Donde:

  - subregla es, obviamente, la subregla a la que se aplica la expresión.
  - Índice_subregla es un número que utilizamos para casos en los que una subregla requiera más de un arreglo.
  - Tipo_cadena puede ser o bien “regexp”, que indica que el *array* almacena una expresión regular, o “replac”, que indica que el *array* almacena la cadena por la que queremos sustituir el *matcheo* de la expresión regular.

Estos *arrays*, como se puede ver en el código en la función *aplica_reglas*, son pasados como argumentos a la función *gensub*. Por ejemplo, si queremos que una subregla 0101 nos pase de un tipo *bool* de lenguaje C a un tipo *boolean* de lenguaje Java, y además elimine todos los tabuladores y los cambie por dos espacios, escribiríamos:

```
arreglos["0101"][0]["regexp"]="bool"
arreglos["0101"][0]["replac"]="boolean"
arreglos["0101"][1]["regexp"]="\\t"
arreglos["0101"][1]["replac"]="  "
```

Es importante destacar que al ser expresiones regulares de tipo *string constant*, cuando escapamos un carácter con una barra, debemos poner dos barras en vez de una. Nótese también que los arreglos con *índice_subregla* menor se realizan antes que los de mayor valor. Como guía para unas primeras reglas, se han creado *arrays* para unas supuestas subreglas 0101, 0102 y 0201.

Una vez configuradas las reglas (individualmente primero y más adelante se va incrementando el número de reglas activas) se recomienda ejecutar la herramienta en modo detección para comprobar que las expresiones regulares encajan con las estructuras con las que debería *matchear*. Una vez que la detección va razonablemente bien, podemos probar a lanzar ejecuciones con el modo detección-corrección los *matcheos* del corrector. Es conveniente que la herramienta siga una mejora iterativa, es decir, ir añadiendo reglas poco a poco e ir comprobando que estas funcionan correctamente; pero como dice el refrán: “cada maestrillo tiene su librillo”.

3.2.5. Código de error y su significado

Como podemos ver en la función *log_awk*, tenemos diferentes niveles del texto a imprimir en función de la salida que siga:

- I: texto informativo, se imprime el texto en el fichero *log_TC.txt*.
- TI: texto informativo, que además de imprimirse en el fichero *log_TC.txt*, se imprime también en el terminal Unix.

- E: texto de error, se imprime en el fichero *log_TC.txt* y en el terminal Unix, después se abandona el *script* correspondiente.

Además en el caso de *detector_TC.sh* se incluyen dos niveles adicionales:

- D: se imprime la cadena de texto en el fichero *informe_detector_TC.csv*.
- R: se imprime la cadena de texto en el fichero *reporte_TC.csv*.

El nivel "E" nos indica, como hemos comentado anteriormente, que ha ocurrido un error, por lo que abandonamos la ejecución del *script* en el que se haya dado el error y nos aparecerá un mensaje de error. Este mensaje de error contiene un código con el que podemos acudir a este apartado y comprobar qué significa e incluso, en algunos casos, se nos sugieren posibles formas de solucionar el problema. Veamos estos códigos de error, que vienen acompañados de un pequeño mensaje:

ERROR 000 - Parámetro de nivel incorrecto

Error en la función *log* de *bash*, el nivel que se pasa como argumento a la función es desconocido. Compruebe que las funciones envían un nivel conocido; compruebe también la función en el caso de que haya olvidado programar la lógica de ese nivel de texto.

ERROR 001 - Directorio scripts no encontrado

El directorio donde deben estar los archivos *corrector_TC.sh, detector_TC.sh y lib_TC.sh* no se encuentra en la ruta correcta, busque el directorio o los archivos y sitúe el directorio en la misma ruta que *lanzador_TC.sh*. En el caso de que encuentre los archivos pero no el directorio "scripts", cree uno en la ruta mencionada con ese nombre; si por el contrario no encuentra los archivos, pruebe a descargar de nuevo la herramienta.

ERROR 002 – lib_TC.sh no existe

El archivo mencionado no se encuentra en la ruta adecuada, es decir, dentro del directorio "scripts". Coloque el archivo en la ruta mencionada y si no lo encuentra pruebe a descargar de nuevo la herramienta.

ERROR 003 - Directorio "config" no encontrado

El directorio "config" no está en la ruta adecuada. Compruebe que la ruta coincide con la de *lanzador_TC.sh*; si no es así, cree el directorio o descargue de nuevo la herramienta.

ERROR 004 – ajustes_TC.cfg no existe

El archivo *ajustes_TC.cfg* no se encuentra en el directorio "config". Coloque el archivo en la ruta mencionada y si no lo encuentra pruebe a descargar de nuevo la herramienta.

ERROR 005 – reglas_TC.cfg no existe

El archivo reglas_TC.cfg no se encuentra en el directorio "config". Coloque el archivo en la ruta mencionada y si no lo encuentra pruebe a descargar de nuevo la herramienta.

ERROR 006 - Ruta de ficheros a arreglar no especificada

La variable *RUTA_FICHEROS*, que puede editar en el archivo, *ajustes_TC.cfg*, tiene un valor nulo, fije un directorio válido.

ERROR 007 - Ruta de ficheros a arreglar no existe

La ruta de ficheros que ha especificado en el archivo *ajustes_TC.cfg* no se encuentra, compruebe que ha escrito bien la ruta.

ERROR 008 - La ruta de los ficheros a corregir no es válida

Ha fijado la ruta de los ficheros a arreglar a la misma ruta en la que se encuentra *lanzador_TC.sh*. Esta ruta no se permite, pues se modificarían todos los ficheros del directorio "salida". Pruebe con otra ruta válida.

ERROR 009 - No se encontraron ficheros que arreglar

No se han encontrado ficheros con la extensión que ha especificado en la función *check_conf* en la ruta especificada.

ERROR 010 - Variable SOBREESCRIBIR debe ser SI o NO

La variable *SOBREESCRIBIR* del archivo *ajustes_TC.cfg* tan solo es capaz de interpretar los valores SI o NO. Estos valores deben ir sin comillas, en mayúsculas y sin espacios.

ERROR 011 - Variable SECUENCIA debe de ser un número

La variable *SECUENCIA* del archivo *ajustes_TC.cfg* debe de ser un número, con el número de cifras que se desee, pero sin espacios.

ERROR 012 - detector_TC.sh no existe

El archivo mencionado no se encuentra en la ruta adecuada, es decir, dentro de la carpeta "scripts". Coloque el archivo en la ruta mencionada y si no lo encuentra pruebe a descargar de nuevo la herramienta.

ERROR 013 - corrector_HMCC.sh no existe

El archivo mencionado no se encuentra en la ruta adecuada, es decir, dentro de la carpeta "scripts". Coloque el archivo en la ruta mencionada y si no lo encuentra pruebe a descargar de nuevo la herramienta.

ERROR 014 - Parámetros de ejecución incorrectos, consulte la ayuda con –help

Los parámetros que se le han proporcionado a *lanzador_TC.sh* son erróneos. Separe el parámetro de la llamada con un espacio y consulte la ayuda para comprobar los parámetros disponibles.

ERROR 015 - Error en reglas_TC.cfg: las reglas deben estar a SI o NO

Las variables *R"X"* del archivo *reglas_TC.cfg* deben de ser SI o NO. Se distingue entre mayúsculas y minúsculas, además, el valor no puede contener espacios.

ERROR 016 - DETECTOR NO FINALIZÓ CORRECTAMENTE

El *script detector_TC.sh* ha devuelto un valor distinto de cero, por lo que ha generado un error. El error puede ser uno definido por el programador u otro generado por el *script*, que se podrá ver en el terminal.

ERROR 017 - Error de retorno de la función check_informe_det

En el *script corrector_TC.sh*, se llama a esta función que se encuentra en *lib_TC.sh*, que comprueba si los ficheros tienen errores. En caso afirmativo esta función devuelve un 1 y en caso negativo un 0, pero en el caso de que no sea ninguno de estos dos valores el devuelto por la función, se genera este error. Revise la función para comprobar que no devuelve otro valor que no sea 0 o 1.

ERROR 018 - CORRECTOR NO FINALIZÓ CORRECTAMENTE

El *script corrector_HMCC.sh* ha devuelto un valor distinto de cero, por lo que ha generado un error. El error puede ser uno definido por el programador u otro generado por el *script* que se podrá ver en el terminal.

3.3. Ejemplo de aplicación: conversor de estilo de código C/C++

Para comprender mejor el funcionamiento del programa se ha realizado un ejemplo usando el esqueleto de la herramienta. Hasta ahora siempre habíamos hablado de convertir un lenguaje de programación en otro diferente, sin embargo, otra de las posibles conversiones es modificar el código de forma que no cambie ni su funcionamiento ni su lenguaje, sino simplemente su estilo. Esta aplicación es de utilidad en varios ámbitos, como la enseñanza (pues permite unificar el estilo del código de los alumnos, facilitando la corrección y comparación) o el empresarial: muchos de los proyectos informáticos tienen un generador de código, aunque no todos los programadores respetan sus normas de estilo, por lo que ahora podemos unificar el estilo de todos ellos.

Como hemos comentado, para crear la herramienta se ha hecho uso de los conocimientos de C/C++ adquiridos en el grado, además de algunas páginas web para consultar los formalismos de dicho código [11] [12]. A su vez, para concretar el estilo que queremos obtener como resultado, se han consultado otras fuentes en las que podemos encontrar diferentes opiniones y justificaciones acerca de diferentes aspectos [13] [3].

3.3.1. Funcionamiento

Antes de ejecutar la herramienta con el código que queremos modificar, debemos de tener en cuenta algunas cosas importantes:

- La herramienta se debe ejecutar siempre y cuando el código haya sido precompilado con éxito, de otro modo no se garantiza el correcto funcionamiento.
- Es recomendable ejecutar la herramienta con “SOBRESCRIBIR”=NO, al menos durante las primeras ejecuciones, puesto que fruto de la conclusión a la que llegó Turing (y llegamos nosotros en el apartado 2): al igual que es imposible que un programa *B* conozca si un programa *A*

termina correctamente alimentando a *B* con *A*, si llamamos "modificar el código correctamente" a "el programa acaba correctamente", podemos llegar a la misma conclusión.

- Como consecuencia del anterior punto, debemos compilar el código una vez ejecutado con la herramienta, de esta forma sabremos que si ahora no compila y antes sí lo hacía, hemos modificado incorrectamente el código. En la misma línea, es recomendable realizar una revisión visual del código (hay multitud de herramientas de comparación de archivos que nos hacen mas sencilla esta tarea), poniendo especial atención a ciertas reglas que pueden generar mas conflictos; veremos qué reglas son mas conflictivas a continuación.

3.3.2. Reglas implementadas

Se han implementado un total de 11 reglas, 3 de las cuales generan un reporte. Estas reglas pueden servir de guía o ayuda para la creación de otras similares, que aunque no coincida su expresión regular, se pueda mantener la estrucutra. Las reglas se han creado con la filosofía de modificar todo lo posible sin hacer peligrar el funcionamiento del código, es decir, si modificar un caso implica dejar mal el código o modificar algo que no queremos modificar, descartaremos el cambio. Puede ver en detalle las expresiones regulares empleadas en el código incluido en el "Anexo II. Conversor de estilo de código C/C++". Se hará incapie en las funciones que desempeñan estas reglas, así como los problemas que pueden ocasionar, poniendo ejemplos de las modificaciones mas relevantes:

Regla 1 - Regla de Identación

Si esta regla está activa, la herramienta edita por defecto la identación de todas las líneas del fichero; tan solo existe una excepción, y es cuando una línea tiene una regla 0103. Es una regla muy compleja, que tiene en cuenta varias posibilidades que son frecuentes en el lenguaje C, sin embargo no es una regla que requiera de un gran detenimiento a la hora de comprobar que la herramienta ha funcionado correctamente y el resultado ha sido el esperado, pues no hay cambios funcionales, tan solo de estilo. Esta regla trae consigo una modificación en la función de *bash check_informe_det*, que provoca que si los

ficheros no contienen errores, si está activa esta regla se devuelva un 1. La identación se realiza con 4 espacios.

Modificaciones que realiza:

- La identación de los *switch-case*. Presenta problemas cuando tenemos un *switch-case* dentro de otro; por el modo en el que se ha realizado la herramienta tiene un arreglo complicado, y al considerarlo como un caso que se escapa de lo habitual, se ha optado por no retocarlo.
- Cuando tenemos un *cout* o un *cin* que ocupa más de una línea, la primera línea tendrá la identación que le corresponda, mientras que las siguientes hasta que finalice tendrán 8 espacios más.
- Si tenemos una operación algebraica multilínea, la primera tendrá nuevamente la identación que le corresponde y el resto 8 más que ella.
- Si hay una inicialización de un array multilínea de números. Puede dar problemas en el caso de que el array sea multidimensional.
- Cuando encontramos una llave de apertura ({), aumentamos la identación en 4; si encontramos una llave de cierre (}), la disminuimos en 4. No cambiamos la identación si tenemos una llave de cierre y una de apertura en la misma línea. Si se da el caso de que tenemos dos llaves de apertura y una de cierre en la misma línea, no cambiaría el valor de la identación; lo mismo ocurriría si tenemos dos llaves de cierre y una de apertura.
- Si detectamos que una línea tiene un número de paréntesis de apertura y de cierre diferente, y no acaba en punto y coma, identamos las siguientes líneas con ocho espacios más; la primera línea conserva su identación.
- Identa las clases, lo cual incluye añadir 4 espacios a los miembros *public*, *private* y *protected*.
- Cuando tenemos varias llaves de apertura en una línea (y sólo llaves de apertura), las separa en varias líneas.
- Cuando tenemos varias llaves de cierre en una línea (y sólo llaves de cierre), las separa en varias líneas.
- Puede provocar un fallo en el caso de que tengamos una cadena de caracteres en varias líneas.

Es una regla que se considera básica, ya que no activarla implica, por la lógica del herramienta, que la mayoría de las líneas pierdan su identación; por lo que se recomienda activarla siempre.

Regla *2* - *Regla* de Tabulación

Es una regla muy simple y sencilla, que no provoca fallos en el funcionamiento del código. Sí que puede provocar, sin embargo, algunos fallos de legibilidad, puesto que un tabulador es sólo una referencia de que queremos situar el cursor en una casilla múltiplo de 4, y nosotros lo estamos sustituyendo por 4 espacios; puede provocar también fallos si se edita el tabulador de una cadena de caracteres. Lo único que realiza esta regla es sustituir cada tabulador de la línea por 4 espacios.

Regla *3* - *Regla* de Bucles sin Llaves

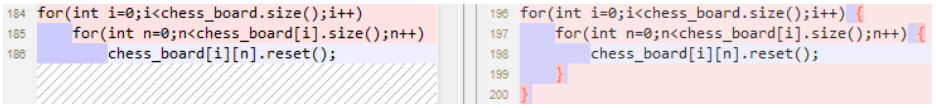
Es, probablemente, la regla más compleja de todas; no solo por su lógica, sino porque los cambios que realiza sí que pueden conllevar fallos en el funcionamiento. Su función es la de colocar las llaves en los bucles que tan solo tienen una sentencia, en los que el programador ha optado por prescindir de éstas.

Modificaciones que realiza:

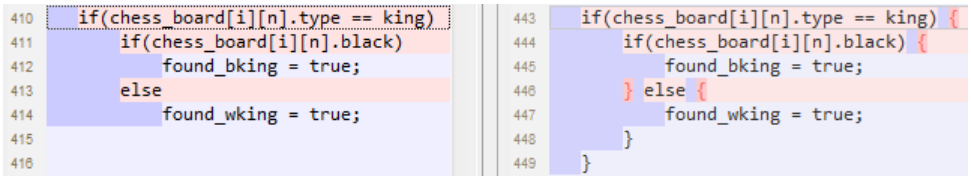
- Coloca las llaves que faltan en los *loops* que tienen una sola sentencia. Estos *loops* incluyen: *"if (), for (;;), while (), else, else if (), do* y *while ();"*.
- Si tenemos que colocar una llave de cierre, buscamos a ver si le corresponde a un *else, else if ()* o un *while ();* y en caso afirmativo se coloca con un espacio detrás; en caso negativo se crea una línea en blanco justo después de la sentencia y se inserta en ella la llave de cierre.
- Si la condición del bucle ocupa más de una línea, cuenta los paréntesis y sitúa la llave en el paréntesis de cierre correspondiente.
- Si la sentencia ocupa más de una línea se busca el punto y coma que indica su final y se coloca la llave de cierre después de ella.
- Si la sentencia se encuentra en la misma línea que el *loop*, pasamos la sentencia a la siguiente línea y situamos los corchetes de apertura en su lugar correspondiente.
- Si dentro del *loop* tenemos otro *loop*, buscamos el final del *loop* interno y situamos ahí la llave de cierre. Nótese que el interno puede ser cualquiera, incluso un *if-elseif-else*; también puede que el *loop* interno tenga más *loops* dentro de él.

Algunos ejemplos del funcionamiento:

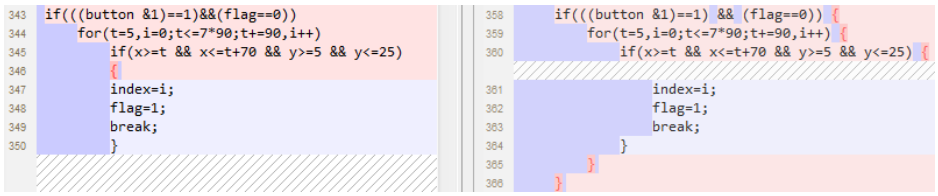
- Cuando tenemos un bucle dentro de otro, este cuenta como una sola sentencia, varios casos diferentes:



11. Caso 1: lo más sencillo

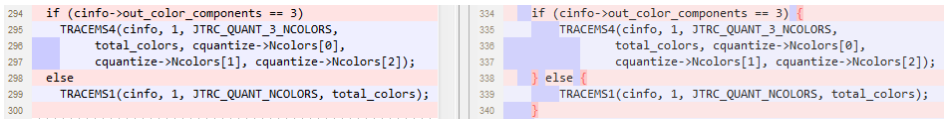


12. Caso 2: tenemos varios bucles pero son if-else if-else



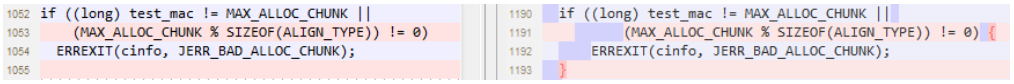
13. Caso 3: se aplica también si tenemos varias sentencias dentro de otras

- En el caso de que la sentencia no ocupe una sola línea:



14. La sentencia dentro del bucle ocupa más de una línea

- La condición del bucle ocupa más de una línea:



15. La condición del bucle ocupa varas líneas

Regla 4 - Regla de Bucles

Esta regla, aunque más sencilla que la anterior, sí que tiene esa complejidad que puede causar un error de funcionamiento. Hace referencia a aquellos *loops* que tienen llave de apertura y de cierre, y coloca la llave de apertura de una forma determinada: precedida por un espacio en blanco, justo después del paréntesis de la condición del *loop* (si la tiene) o del *loop*; por ejemplo:

- if ()_{
- else_{

Este convenio es el mismo que sigue la regla 3 a la hora de colocar las llaves. Los *loops* que incluye la Regla 4 son: "*if (), for (;;), while (), catch (), switch (),try, case :, default:, else, else if (), do* y *while ();*".

Modificaciones que realiza:

- Aunque tengamos la llave de apertura en la misma línea, comprueba que contenga el espacio mencionado anteriormente; si no lo tiene, lo coloca.
- En el caso de que la condición del *loop* ocupe varias líneas, buscará el paréntesis que corresponda con el final de la condición y lo colocará con él.
- Se asegura de que la llave que mueve, corresponda al bucle. Por ejemplo, si tenemos un *if-else* cuyo *else* no tiene llaves por ser de una sola sentencia, y el *if* acaba en un *if* con corchete: podría ocurrir que la herramienta moviera la llave de cierre del *if* interno al else. La herramienta tiene estos casos en cuenta y no mueve las llaves.

Regla 5 - Regla de Funciones

Esta regla retoca todas las funciones (que devuelven un tipo de dato conocido), *class*, *namespace, struct* y *enum*, de forma que coloca la llave en la línea siguiente a estos. Los tipos de datos conocido que pueden devolver las funciones son: "*unsigned, signed, char, auto, bool, short, long, int, float, double, void*" (incluso combinaciones de ellos, como *unsigned long*) y pueden ir precedidos o no por *const* o *inline*. Hay que tener cuidado con esta regla, pues siempre que hay movimientos de llaves hay peligro de modificar el funcionamiento del programa. Por ejemplo:

Sitúa un retorno de carro antes de la llave, de forma que pasa a estar en la línea siguiente.

16. Movimiento de la llave en una función

Regla 6 - Regla de Variables

Se encarga de separar la declaración de diferentes variables que se han realizado en la misma sentencia según el mismo tipo de dato, en varias líneas, de modo que a cada variable le corresponde una línea. Los tipos de datos que se han incluido son: "unsigned, signed, char, auto, bool, short, long, int, float,

double" (incluso combinaciones de ellos, como *unsigned long*) y pueden ir precedidos o no por *const*.

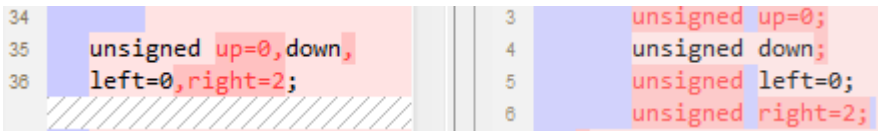
Modificaciones que realiza:

- Trata tanto declaraciones de varias variables en una línea como varias líneas, es decir:
  int a,b,c,
  d,e;
- Si alguna de las variables ha sido inicializada, se traslada la inicialización junto con la variable a su nueva línea.
- Hay determinados casos que se han aislado y no se tratan debido a los problemas que dan a la hora de corregirlos: cuando las variables contienen paréntesis (porque se han inicializado con el valor que devuelve una función por ejemplo) y cuando tenemos en esa línea un comentario "monolínea" que tiene una coma en él.
- Puede dar problemas también en el caso de tener una inicialización de un *array* con llaves.
- Hay que tener precaución con las funciones, por ejemplo:
  void window::SetFont(const int iSize,
  const unsigned short usStyle, const fontfamily ffFamily,
  const char *cFontName) {

  Puede detector la segunda línea como una declaración en varias líneas; se recomienda evitar este tipo de construcciones.

El mayor problema que puede dar esta regla es la incongruencia de los datos, por lo que se recomienda que se tenga cuidado con los tipos de datos en los que son declaradas las variables una vez separadas.

A continuación se puede ver un ejemplo de funcionamiento:



17. Separación de diferentes variables de tipo unsigned

Regla 7 - Regla de Comentarios

Su función es la de poner los comentarios de acuerdo a un formato determinado. Lo primero que tenemos que hacer es diferenciar entre los

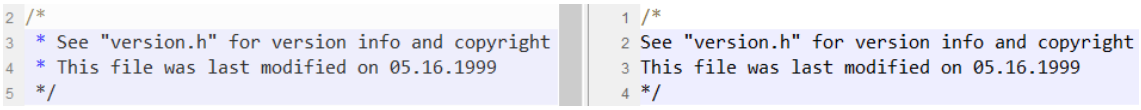
comentarios monolínea (empiezan por doble barra inclinada "//") y los multilínea (empiezan por "/*" y acaban en "*/"). El formato consiste en:

- Monolínea: tiene que haber al menos un espacio en blanco después de las barras inclinadas y antes del siguiente carácter.
- Multilínea: "/*" se sitúa solo en una línea, así como "*/". El resto del comentario estará separado por líneas de modo que cada línea va precedida por un asterisco.

Algunos detalles:

- Si un comentario monolínea tiene más de dos barras inclinadas, estas se conservan.
- Si un comentario multilínea de apertura ("/*") o de cierre ("*/") tiene más de un asterisco, se eliminan todos salvo uno.
- Si un comentario multilínea tiene más de un espacio después de las barras, se conservan.
- La identación que tenían anteriormente las líneas intermedias de los comentarios multilínea se eliminan.
- En los comentarios multilínea se añade una línea adicional por encima y por debajo de él, esto es un pequeño fallo, pero dado que no es considerado crítico y que su arreglo complica el código, se ha optado por mantener.

Un ejemplo del funcionamiento de esta regla, para el caso de multilínea:



18. Ejemplo de modificación de comentarios multilínea

Regla 8 - Regla de Operadores Lógicos

Es una regla muy sencilla, que además no cambia el funcionamiento del programa. Simplemente detecta un OR lógico o un AND lógico que no contengan espacios a su alrededor y los añade.

Regla 9 - Regla Reporte de No Inicialización

Nos genera un reporte cuando una línea contiene una variable que ha sido creada pero no ha sido inicializada en su creación.

Algunas cosas a tener en cuenta:

- Es importante destacar que el reporte aparece siempre y cuando la inicialización no sea en la misma línea que la creación, por lo que si está inicializada en otra línea, el reporte se generará igualmente.
- Si tenemos paréntesis en la línea, el reporte no aparecerá.
- Si tenemos comentarios monolínea con una coma, el reporte no aparecerá.
- Hay que tener precaución con las funciones, por ejemplo:
  void window::SetFont(const int iSize,

  const unsigned short usStyle, const fontfamily ffFamily,

  const char *cFontName) {

  La segunda línea puede aparecer como reporte.

Regla 10 - Regla Reporte de Longitud de Línea

Genera un reporte cuando una línea excede el tamaño definido en la variable de configuración "LONG_MAX", que ha sido añadida en el archivo *ajustes_TC.cfg*.

Regla 11 - Regla Reporte de Explicación Funcional

Cuando encontramos un función que devuelve un tipo de dato conocido, es decir: "*unsigned, signed, char, auto, bool, short, long, int, float, double, void*" (incluso combinaciones de ellos, como *unsigned long*), pueden ir precedidos o no por *const* o *inline*; que no tiene un comentario por encima de ella, se sobreentiende que no existe explicación funcional de esta función, por lo que se genera un reporte.

Notas:

- Se ha tenido en cuenta que por encima de la función puedan existir espacios ante de tener el comentario.
- Lo que en realidad se busca es, o un comentario monolínea o el fin de uno multilínea ("*/"), por lo que ambos tipos de comentarios se admiten.
- Como en todos los reportes, queda a juicio del programador si esa función necesita de un comentario explicativo.

3.3.3. Códigos de error adicionales

Continuando con la numeración de los códigos de la herramienta original, se han añadido algunos errores adicionales, como el *checkeo* de la nueva variable de configuración, e incluso algunos tratamientos sobre el AWK.

ERROR 019 - Variable LONG_MAX debe de ser un número

La variable *LONG_MAX* del archivo *ajustes_TC.cfg* debe de ser un número, con el número de cifras que se desee, pero sin espacios.

ERROR 020 - Error en detección de la regla... para el fichero...

Error generado por el AWK del *detector_TC.sh* cuando se embucla con una regla. Se nos indica la regla que ha provocado el fallo y el fichero en el que ha ocurrido. Se recomienda eliminar el fichero de la ruta a corregir y volver a pasar la herramienta, una vez transcurra con éxito la ejecución, se puede pasar la herramienta al archivo problemático desactivando la regla que ha causado los errores.

Nuevamente volvemos a mencionar el problema de la parada: no podemos saber si un programa se detiene o no, por lo que si observamos que llevamos demasiadas iteraciones en un bucle, salta este error y finaliza la ejecución.

ERROR 021 - Error con la identación del fichero... en el corrector

Error generado por el AWK del *corrector_TC.sh* cuando ha habido un error con la identación del fichero mencionado. Se recomienda revisar el fichero, pues es posible que contenga algún fallo, en especial los *loops*; en caso de considerar que el fichero está correcto, elimine el fichero de la ruta de fichero a corregir.

4. Conclusiones

Hemos creado el esqueleto de una herramienta genérica que permite al usuario, mediante la programación de dos scripts en AWK, transformar el código de un lenguaje de programación en otro. Para ello se han repasado algunas demostraciones y conceptos de la historia de la computación, que de una forma o de otra pueden ayudarnos a hacer más fácil esta tarea; así, por ejemplo, sabemos que podemos *matchear* y qué no con las expresiones regulares (ayudándonos del *pumping lemma*) sin perder el tiempo buscando una forma de hacerlo.

La parte correspondiente a la programación de estos *scripts* requiere un conocimiento muy elevado de los dos lenguajes, tanto del lenguaje original, como del que queremos obtener como resultado, sin embargo, algunas de las estructuras utilizadas en el ejemplo de implementación, pueden servir de referencia e incluso ser semejantes a las que pueda necesitar el usuario.

Bibliografía

[1] Aho, A., Kernighan, B., & Weinberger, P. (1988). *The AWK Programming Language.* Pearson.

[2] Desconocido. (s.f.). *ElBlogDeG10*. Recuperado el 25 de Julio de 2017, de https://neobrr.wordpress.com/2009/02/04/filosofia-unix/

[3] Desconocido. (s.f.). *Github*. Recuperado el 19 de Junio de 2017, de https://google.github.io/styleguide/cppguide.html

[4] Desconocido. (s.f.). *Wikipedia*. Recuperado el 6 de Septiembre de 2017, de https://es.wikipedia.org/wiki/Máquina_de_Turing

[5] Desconocido. (s.f.). *Wikipedia*. Recuperado el 2 de Septiembre de 2017, de https://es.wikipedia.org/wiki/Tesis_de_Church-Turing

[6] Desconocido. (s.f.). *Wikipedia*. Recuperado el 4 de Agosto de 2017, de https://es.wikipedia.org/wiki/Teoremas_de_incompletitud_de_G%C3%B6del

[7] Desconocido. (s.f.). *Wikipedia*. Recuperado el 29 de Julio de 2017, de https://es.wikipedia.org/wiki/Programa_de_Hilbert

[8] Desconocido. (s.f.). *Wikipedia*. Recuperado el 22 de Agosto de 2017, de https://es.wikipedia.org/wiki/Problema_de_la_parada

[9] GNU. (s.f.). *GNU*. Recuperado el 6 de Julio de 2017, de https://www.gnu.org/software/gawk/manual/gawk.html

[10] Hopcroft, J. E., Motwani, R., & Ullman, J. (2002). *Introducción a la teoría de autómatas, lenguajes y computación.* Addison-Wesley.

[11] Kernighan, B., & Ritchie, D. (1988). *The C Programming Language.* Prentice Hall.

[12] Llopis, J. (s.f.). *Matesfacil*. Recuperado el 3 de Septiembre de 2017, de https://www.matesfacil.com/automatas-lenguajes/Maquina-Turing.html

[13] López, J. M. (s.f.). *Blogthinkbig*. Recuperado el 30 de Agosto de 2017, de https://blogthinkbig.com/unix-el-padre-de-los-sistemas-operativos-actuales

[14] Martiloni, D. (s.f.). *Universidad de Murcia*. Recuperado el 16 de Agosto de 2017, de http://www.um.es/docencia/barzana/DIVULGACION/INFORMATICA/Unix01.html

[15] Stroustrup, B. (2011). *The C++ Programming Language.* Addison-Wesley.

[16] University, C. M. (s.f.). *Carnegie Mellon University*. Recuperado el 9 de Junio de 2017, de https://users.ece.cmu.edu/~eno/coding/CppCodingStandard.html

[17] Venkateshmurthy, M. G. (2009). *Introduction to Unix & Shell Programming.* Pearson.

Anexos

A continuación se muestran los anexos correspondientes al código:

- Anexo I: código esqueleto de la herramienta.
- Anexo II: conversor de estilo de código C/C++.

ANEXO I. CÓDIGO ESQUELETO DE LA HERRAMIENTA

Este anexo contiene el código esqueleto de cada archivo de la herramienta, estos archivos son:

- ajustes_TC.cfg
- reglas_TC.cfg
- lanzador_TC.cfg
- lib_TC.sh
- detector_TC.sh
- corrector_TC.sh

ajustes_TC.cfg

```
1  RUTA_FICHEROS="entrada”
2  SOBRESCRIBIR=NO
3  SECUENCIA=0
```

reglas_TC.cfg

```
1  R1=SI
2  R2=SI
```

lanzador_TC.sh

```
1  #!/bin/bash
2  #MOD_REQ#
3
4  # CONSTANTES
5  # Número de reglas que generan reporte
6  export NUM_REG_REP=3 #MOD_REQ#
7  export MI_PATH=$(pwd)
8  # VARIABLES
9  export LIBRERIA="${MI_PATH}/scripts/lib_TC.sh"
10 export LOG="log_TC"
11 # FUNCIONES
12
13 ##############################################################
   ##########
14 #                    Nombre función: check_lib
          #
15 #    Descripción: comprueba que lib_TC.sh existe y tiene
   permisos    #
16 ##############################################################
   ##########
17 function check_lib()
18 {
19     if [ -d scripts ]
20     then
21         echo "[INFO] ---- Existe directorio de scripts" >>
   $LOG
22         if [ -f $LIBRERIA ]
23         then
24             echo "[INFO] ---- lib_TC.sh existe" >> $LOG
25             if [ -x $LIBRERIA ]
26             then
27                 echo "[INFO] ---- lib_TC.sh tiene permisos
   de
   ejecución" >> $LOG
28             else
29                 chmod +x $LIBRERIA
30                 echo "[INFO] ---- Permisos otorgados a
   lib_TC.sh" |
   tee -a $LOG
31             fi
32         else
33             echo "[ERROR] 002 lib_TC.sh no existe" | tee -a
   $LOG
34             exit 1;
35         fi
36     else
37         echo "[ERROR] 001 Directorio scripts no encontrado"
   | tee -a
   $LOG
38         exit 1;
39     fi
40 }
```

```
41
42  # Por si no lo hemos movido anteriormente
43  [ -f $LOG ] && rm $LOG
44
45  ##############################################################
    ##########
46  #                       Inicio de lanzador_TC.sh
    #
47  ##############################################################
    ##########
48
49  echo "
    ##############################################################
    ####" | tee -a $LOG
50  echo "             #         Herramienta de transformación de
    código
    fuente         #" | tee -a $LOG
51  echo "             #
    #" | tee -a $LOG
52  echo "             #       Trabajo de Fin de Grado de Álvaro
    Fraile
    Cáceres        #" | tee -a $LOG
53  echo "             #             Tutores:
    #" | tee -a $LOG
54  echo "             #                 - Podar Cristea,
    Smaranda
    #" | tee -a $LOG
55  echo "             #                 - Mazaeda Echevarria,
    Rogelio
    #" | tee -a $LOG
56  echo "
    ##############################################################
    ####" | tee -a $LOG
57
58  ##############################################################
    ##########
59  #                             Código
    #
60  ##############################################################
    ##########
61
62  check_lib
63
64  # source nos "pasa las variables de hijo a padre"
65  source $LIBRERIA
66
67  # Comprobamos que todos los directorios están creados
68  check_dir
69
70  # Extraemos las variables de configuración
71  check_conf
```

```
72
73  # Convertimos los archivos a tratar en formato UNIX
74  dos2unix -k -q $WORKSHOP
75
76  # Comprobamos que los scripts tienen permisos de ejecución
77  check_perm
78
79  # Comprobamos que los parámetros de ejecución son correctos
80  check_param $1 $#
81
82  # Obtenemos las reglas activas
83  check_reglas
84
85  # Creamos el directorio de la secuencia actual
86  crea_dir_seq
87
88  if [[ $1 == "D" ]]
89  then
90      . $DETECTOR
91      log "La herramienta finalizó correctamente" "TI"
92  elif [[ $1 == "DC" ]]
93  then
94      . $DETECTOR
95      . $CORRECTOR
96
97      log "La herramienta finalizó correctamente" "TI"
98      log "Los ficheros modificados se encuentran en
    ${RUTA_CORREGIDOS}" "TI"
99  elif [[ $1 == "R" ]]
100 then
101     for (( i=1; i <= (${#regla[*]} - $NUM_REG_REP); i++ ))
102     do
103         regla[$i]=0
104     done
105     log "Reglas de corrección desactivadas" "TI"
106     . $DETECTOR
107
108     log "La herramienta finalizó correctamente" "TI"
109 fi
110
111
112 exit 0;
```

lib_TC.sh

```
1  ##############################################################
   ##########
2  #                          Inicio de lib_TC.sh
          #
3  ##############################################################
   ##########
4
5  # CONSTANTES
6  export CONFIG="${MI_PATH}/config"
7  export CFG_AJUSTES="${CONFIG}/ajustes_TC.cfg"
8  export CFG_REGLAS="${CONFIG}/reglas_TC.cfg"
9  export CORRECTOR="${MI_PATH}/scripts/corrector_TC.sh"
10 export DETECTOR="${MI_PATH}/scripts/detector_TC.sh"
11 export SALIDA="${MI_PATH}/salida"
12 export ENTRADA="${MI_PATH}/entrada"
13
14 # VARIABLES
15 export REPORTE="reporte_TC"
16 export INFORME_DET="informe_detector_TC"
17 export RUTA_FICHEROS=""
18 export SOBRESCRIBIR=""
19 export WORKSHOP=""
20 export SECUENCIA=""
21 export regla
22
23 ##############################################################
   ##########
24 #                    Nombre función: log
            #
25 #     Descripción: imprime la cadena que recibe en el
   fichero         #
26 #       correspondiente según el nivel del texto a imprimir
            #
27 #     Argumentos: $1 --> cadena a imprimir
            #
28 #                 $2 --> Nivel del texto
            #
29 #                        I --> Información en log
            #
30 #                        TI -> Información en terminal e log
            #
31 #                        E --> Error
            #
32 ##############################################################
   ##########
33 function log()
34 {
35       case $2 in
36            "I")
37                  # Pinta en log_TC.txt
38                  echo "[INFO] ----" $1 >> $LOG
39                  ;;
```

```
40              "TI")
41                    # Pinta en log_TC.txt y en el terminal
42                    echo "[INFO] ----" $1 | tee -a $LOG
43                    ;;
44              "E")
45                    # Pinta en log_TC.txt y en el terminal
46                    echo "[ERROR]" $1 | tee -a $LOG
47                    exit 1;
48                    ;;
49              *)
50                    # Error en el parámetro
51                    echo "[ERROR] 000 Parámetro de nivel
    incorrecto" |
    tee -a $LOG
52                    exit 1;
53                    ;;
54        esac
55    }
56
57    ##############################################################
    ##########
58    #                    Nombre función: ayuda_lanz
              #
59    #     Descripción: muestra los posibles parámetros de
    ejecución       #
60    ##############################################################
    ##########
61    function ayuda_lanz()
62    {
63          log "Parámetros de ejecución admitidos:" "I"
64          echo "                  D --> Solo detección"
65          echo "                  DC --> Detección y corrección"
66          echo "                  R --> Solo reportes de warnings"
67          echo "                  --help --> Ayuda"
68    }
69
70    ##############################################################
    ##########
71    #                    Nombre función: check_dir
              #
72    #        Descripción: comprueba la existencia de directorios
              #
73    ##############################################################
    ##########
74    function check_dir()
75    {
76          if [ -d $ENTRADA ]
77          then
78                log "Existe directorio de entrada" "I"
79          else
80                log "Directorio de entrada creado" "TI"
81                mkdir $ENTRADA
82          fi
```

```
83
84          if [ -d $SALIDA ]
85          then
86                log "Existe directorio de salida" "I"
87          else
88                log "Directorio de salida creado" "TI"
89                mkdir $SALIDA
90          fi
91
92          if [ -d $CONFIG ]
93          then
94                log "Existe directorio de configuración" "I"
95                if [ -f $CFG_AJUSTES ]
96                then
97                      log "ajustes_TC.cfg existe" "I"
98                else
99                      log "004 ajustes_TC.cfg no existe" "E"
100               fi
101               if [ -f $CFG_REGLAS ]
102               then
103                     log "reglas_TC.cfg existe" "I"
104               else
105                     log "005 reglas_TC.cfg no existe" "E"
106               fi
107         else
108               log "003 Directorio config no encontrado" "E"
109         fi
110   }
111
112   ##############################################################
      ##########
113   #                    Nombre función: check_conf
            #
114   #  Descripción: extrae las variables necesarias del fichero
      de conf   #
115   ##############################################################
      ##########
116   function check_conf()
117   {
118         . $CFG_AJUSTES
119         if [ -z $RUTA_FICHEROS ]
120         then
121               log "006 Ruta de ficheros a modificar no
      especificada" "E"
122         else
123               if [ -d $RUTA_FICHEROS ]
124               then
125                     log "Ruta de ficheros a modificar:
      ${RUTA_FICHEROS}"
      "TI"
126                   if [[ $MI_PATH != $RUTA_FICHEROS ]]
127                   then
128                         WORKSHOP=$(find ${RUTA_FICHEROS} -
```

```
      iname "*.c" -
      or -iname "*.cpp" -or -iname "*.h" -or -iname "*.hpp")
      #MOD_REQ#
129                         if [[ -z $WORKSHOP ]]
130                         then
131                               log "009 No se encontraron
      ficheros que
      modificar" "E"
132                         else
133                               n_fich=$(find ${RUTA_FICHEROS} -
      iname
      "*.c" -or -iname "*.cpp" -or -iname "*.h" -or -iname "*.hpp"
      | wc -l)  #MOD_REQ#
134                               log "${n_fich} fichero(s) a
      modificar
      encontrado(s)" "TI"
135                         fi
136                   else
137                         log "008 La ruta de los ficheros a
      modificar no
      es válida" "E"
138                   fi
139             else
140                   log "007 Ruta de ficheros a modificar no
      existe" "E"
141             fi
142       fi
143
144       if [[ $SOBRESCRIBIR == "SI" ]]
145       then
146             log "Sobreescritura de archivos activada" "TI"
147             log "¿Seguro que quiere SOBRESCRIBIR los
      archivos?" "TI"
148             log "Escriba NO para abandonar o cualquier otra
      cadena para
      continuar" "TI"
149             read respuesta
150             log "Escribió: ${respuesta}" "I"
151             if [[ $respuesta == "NO" ]]
152             then
153                   log "Operación cancelada, abandonando
      herramienta"
      "TI"
154                   exit 2
155             fi
156       elif [[ $SOBRESCRIBIR == "NO" ]]
157       then
158             log "Sobreescritura de archivos desactivada" "TI"
159       else
160             log "010 Variable SOBRESCRIBIR debe ser SI o NO"
      "E"
161       fi
162
```

```
163         if [[ $SECUENCIA =~ ^[0-9]+$ ]]
164         then
165               log "Patrón de secuencia OK" "I"
166               SECUENCIA=$(($SECUENCIA + 1))
167               log "Número de secuencia actualizado a
    ${SECUENCIA}" "TI"
168               rm $CFG_AJUSTES
169               echo "RUTA_FICHEROS=\""$RUTA_FICHEROS"\"" >>
    $CFG_AJUSTES
170               echo "SOBRESCRIBIR="$SOBRESCRIBIR >> $CFG_AJUSTES
171               echo "SECUENCIA="$SECUENCIA >> $CFG_AJUSTES
172         else
173               log "011 Variable SECUENCIA debe de ser un
    número" "E"
174         fi
175
176   }
177
178   ##############################################################
    ##########
179   #                     Nombre función: check_perm
              #
180   #      Descripción: comprueba que los scripts tienen permisos
              #
181   ##############################################################
    ##########
182   function check_perm()
183   {
184         if [ -f $DETECTOR ]
185         then
186               log "detector_TC.sh existe" "I"
187               if [ -x $DETECTOR ]
188               then
189                     log "detector_TC.sh tiene permisos de
    ejecución" "I"
190               else
191                     chmod +x $DETECTOR
192                     log "Permisos otorgados a detector_TC.sh"
    "TI"
193               fi
194         else
195               log "012 detector_TC.sh no existe" "E"
196         fi
197
198         if [ -f $CORRECTOR ]
199         then
200               log "corrector_TC.sh existe" "I"
201               if [ -x $CORRECTOR ]
202               then
203                     log "corrector_TC.sh tiene permisos de
    ejecución" "I"
204               else
205                     chmod +x $CORRECTOR
```

```
206                 log "Permisos otorgados a corrector_TC.sh"
    "TI"
207           fi
208     else
209           log "013 corrector_TC.sh no existe" "E"
210     fi
211 }
212
213
214
215 ##############################################################
    ##########
216 #                    Nombre función: check_param
            #
217 #   Descripción: comprueba que los parámetros del lanzador
    son OK    #
218 #     Argumentos: $1 --> $1 de ejecución
            #
219 #                  $2 --> $# número de parámetros de
    ejecución         #
220 ##############################################################
    ##########
221 function check_param()
222 {
223     if [[ $2 == "1" ]]
224     then
225           log "Número de parámetros de ejecución correctos"
    "I"
226           case $1 in
227           "D")
228                 log "Lanzador en modo detección" "TI"
229                 ;;
230           "DC")
231                 log "Lanzador en modo detección y
    corrección" "TI"
232                 ;;
233           "R")
234                 log "Lanzador en modo reportes " "TI"
235                 ;;
236           "--help")
237                 log "Ayuda solicitada en ejecución" "TI"
238                 ayuda_lanz
239                 exit 1;
240                 ;;
241           *)
242                 log "014 Parámetros de ejecución
    incorrectos,
    consulte la ayuda con --help" "E"
243                 ;;
244           esac
245     else
246           log "014 Parámetros de ejecución incorrectos,
    consulte la
```

```
ayuda con --help" "E"
247         fi
248   }
249
250   ##############################################################
      ##########
251   #                     Nombre función: check_reglas
                #
252   #  Descripción: extrae las variables necesarias del fichero
      de conf  #
253   ##############################################################
      ##########
254   function check_reglas()
255   {
256         i=0
257         for r in $(cat ${CFG_REGLAS})
258         do
259               i=$(($i + 1))
260               if [[ $r =~ R[0-9]+\=NO$ ]]
261               then
262                     regla[$i]=0
263               elif [[ $r =~ R[0-9]+\=SI$ ]]
264               then
265                     regla[$i]=1
266                     log "Regla ${i} activa" "TI"
267               else
268                     log "015 Error en reglas_TC.cfg: las reglas
      deben
      estar a SI o NO" "E"
269               fi
270         done
271   }
272
273   ##############################################################
      ##########
274   #                     Nombre función: crea_dir_seq
                #
275   #  Descripción: crea el directorio con la secuencia
      correspondiente  #
276   ##############################################################
      ##########
277   function crea_dir_seq()
278   {
279         # Directorio de la salida para esta secuencia
280         export SAL_SEQ="${SALIDA}/${SECUENCIA}"
281
282         # Sobreescribimos en caso de que ya haya una carpeta
      con la
      secuencia actual
283         [ -d $SAL_SEQ ] && rm -rf $SAL_SEQ && log "Directorio
      de salida
      sobreescrito (n° de secuencia repetido)" "TI"
284
```

```
285       # Creamos el directorio donde se guardarán los
    archivos
286       mkdir $SAL_SEQ
287
288       # Movemos el log a su nueva ubicación
289       [ -f $LOG ] && mv $LOG
    "${SAL_SEQ}/${LOG}_${SECUENCIA}.txt"
290
291       # Incluímos el PATH en el nombre de los archivos
292       LOG="${SAL_SEQ}/${LOG}_${SECUENCIA}.txt"
293       INFORME_DET="${SAL_SEQ}/${INFORME_DET}_${SECUENCIA}.cs
    v"
294       REPORTE="${SAL_SEQ}/${REPORTE}_${SECUENCIA}.csv"
295   }
296
297   ##############################################################
    ##########
298   #                    Nombre función: crea_dir_correg
              #
299   #  Descripción: crea los directorios de la ruta de ficheros
    escogida #
300   ##############################################################
    ##########
301   function crea_dir_correg
302   {
303       if [[ $SOBRESCRIBIR == "SI" ]]
304       then
305             RUTA_CORREGIDOS=$RUTA_FICHEROS"_copy"
306             mkdir $RUTA_CORREGIDOS
307             log "Directorio ${RUTA_CORREGIDOS} creado" "I"
308
309             for i in $WORKSHOP
310             do
311                   fich="${i#${RUTA_FICHEROS}}"
312                   dir="${fich%/*}"
313
314                   mkdir -p "${RUTA_CORREGIDOS}/${dir}"
315             done
316       elif [[ $SOBRESCRIBIR == "NO" ]]
317       then
318             export CORREGIDOS="${SAL_SEQ}/corregidos"
319             mkdir $CORREGIDOS
320             log "Directorio ${CORREGIDOS} creado" "I"
321             export ORIGINALES="${SAL_SEQ}/originales"
322             mkdir  $ORIGINALES
323             log "Directorio ${ORIGINALES} creado" "I"
324
325             for i in $WORKSHOP
326             do
327                   fich="${i#${RUTA_FICHEROS}}"
328                   dir="${fich%/*}"
329
330                   mkdir -p "${CORREGIDOS}/${dir}"
```

```
331                 mkdir -p "${ORIGINALES}/${dir}"
332                 cp -ra $i "${ORIGINALES}${fich}"
333           done
334
335           RUTA_CORREGIDOS=$CORREGIDOS
336     fi
337 }
338
339 ##############################################################
    ##########
340 #                   Nombre función: check_informe_det
          #
341 #  Descripción: comprueba que el informe del detector trae
    al menos  #
342 #                   1 error
          #
343 ##############################################################
    ##########
344 function check_informe_det
345 {
346     if [[ $(cat ${INFORME_DET}) =~ [0-9] ]]
347     then
348           log "Los ficheros contienen errores" "TI"
349           return 1
350     else
351           log "Los ficheros no contienen errores" "TI"
352           return 0
353     fi
354
355 }
356
357 # FUNCIONES
358 export -f log
359 export -f ayuda_lanz
360 export -f check_dir
361 export -f check_conf
362 export -f check_perm
363 export -f check_param
364 export -f check_reglas
365 export -f crea_dir_seq
366 export -f crea_dir_correg
367 export -f check_informe_det
```

detector_TC.sh

```
1  ##############################################################
   ##########
2  #                        Inicio de detector_TC.sh
           #
3  ##############################################################
   ##########
4
5
6  log "INICIALIZANDO DETECTOR" "TI"
7
8  echo "Fichero;Linea;Regla;Subregla" >> $INFORME_DET
9  echo "Fichero;Linea;Regla;Regla Reporte;Incidencia" >>
   $REPORTE
10
11 awk -b -v reporte=$REPORTE -v informe_det=$INFORME_DET -v
   fich_log=$LOG -v reglas="${regla[*]}" -v
   num_reg_rep=$NUM_REG_REP '
12 ##############################################################
   ##########
13 #                     Nombre función: log_awk
           #
14 #     Descripción: imprime la cadena que recibe en el
   fichero         #
15 #       correspondiente según el nivel del texto a imprimir
           #
16 #     Argumentos: texto --> cadena a imprimir
           #
17 #                   nivel --> Nivel del texto
           #
18 #                         I --> Información en log
           #
19 #                         TI -> Información en terminal e log
           #
20 #                         ID -> Arreglos en
   informe_detector.csv          #
21 #                         R --> Reporte generado por regla
           #
22 ##############################################################
   ##########
23 function log_awk(texto,nivel)
24 {
25       if (nivel == "I") {
26             # Pinta en log_TC.txt
27             print "[INFO] ---- " texto >> fich_log
28       } else if (nivel == "TI") {
29             # Pinta en log_TC.txt y en el terminal
30             print "[INFO] ---- " texto >> fich_log
31             print "[INFO] ---- " texto
32       } else if (nivel == "E") {
33             # Pinta en log_TC.txt y en el terminal
34             print "[ERROR]" texto >> fich_log
35             print "[ERROR]" texto
```

```
36              exit 1
37        } else if (nivel == "D") {
38              # Pinta en informe_detector_TC.csv
39              print texto >> informe_det
40        } else if (nivel == "R") {
41              # Pinta en reportes_TC.csv
42              if (texto ~ /^[^;]+;[^;]+;[^;]+;1;$/) {
43                    texto=texto ""
44              }
45              print texto >> reporte
46        }
47  }
48
49
50  BEGIN{
51        # Las reglas a 1 y 0 están en regla
52        n_reglas=split(reglas,regla," ")
53
54        # Inicializamos el array con lo errores de cada regla
55        for (i in regla) {
56              n_errores[i]=0
57        }
58        n_errores_totales=0
59  }{
60
61        ##########################################################
    ############
62        #                              REGLA 1
    #
63        ##########################################################
    ############
64        if (regla[1] == 1) {
65
66        }
67
68        ##########################################################
    ############
69        #                              REGLA 2
    #
70        ##########################################################
    ############
71        if (regla[2] == 1) {
72
73        }
74
75  }
76  END {
77        log_awk("Estadísticas de detección","TI")
78        for (i=1; i <= n_reglas; i++) {
79              if (regla[i] == 1) {
80                    if (i <= n_reglas-num_reg_rep) {
81                          log_awk("N° errores regla " i ": "
    n_errores[i],"TI")
```

```
82                  } else {
83                       log_awk("N° reportes regla " i ": "
    n_errores[i],"TI")
84                  }
85                  n_errores_totales+=n_errores[i]
86             }
87        }
88        log_awk("N° errores totales: " n_errores_totales,"TI")
89
90   }' $WORKSHOP
91
92   err_detector=$?
93   if [[ $err_detector == 0 ]]
94   then
95        log "DETECTOR FINALIZÓ SIN INCIDENCIAS" "TI"
96   else
97        log "016 DETECTOR NO FINALIZÓ CORRECTAMENTE" "E"
98   fi
```

corrector_TC.sh

```
1  ##############################################################
   ##########
2  #                        Inicio de corrector_TC.sh
   #
3  ##############################################################
   ##########
4
5  export RUTA_CORREGIDOS=""
6
7  crea_dir_correg
8
9  check_informe_det
10 ret_corrector=$?
11
12 if [[ $ret_corrector == 1 ]]
13 then
14       log "INICIALIZANDO CORRECTOR" "TI"
15 elif [[ $ret_corrector == 0 ]]
16 then
17       log "ABANDONANDO CORRECTOR" "TI"
18       return
19 else
20       log "017 Error de retorno de la función
   check_informe_det" "E"
21 fi
22
23 awk -b -v informe_det=$INFORME_DET -v fich_log=$LOG -v
   ruta_corregidos=$RUTA_CORREGIDOS -v
   ruta_ficheros=$RUTA_FICHEROS '
24 ##############################################################
   ##########
25 #               Nombre función: genera_array_arreglos
            #
26 #     Descripción: genera un array con las expresiones de
   matcheo    #
27 #                        y sustitución de cada subregla
            #
28 ##############################################################
   ##########
29 function genera_array_arreglos()
30 {
31       ##########################################################
   ##########
   ##
32       #                                REGLA 1
             #
33       ##########################################################
   ############
34       arreglos["0101"][0]["regexp"]=""
35       arreglos["0101"][0]["replac"]=""
36       arreglos["0101"][1]["regexp"]=""
37       arreglos["0101"][1]["replac"]=""
```

```
38
39        arreglos["0102"][0]["regexp"]=""
40        arreglos["0102"][0]["replac"]=""
41
42        ##########################################################
    ############
43        #                                REGLA 2
                #
44        ##########################################################
    ############
45        arreglos["0201"][0]["regexp"]=""
46        arreglos["0201"][0]["replac"]=""
47        arreglos["0201"][1]["regexp"]=""
48        arreglos["0201"][1]["replac"]=""
49
50  }
51  ##############################################################
    ###########
52  #                     Nombre función: log_awk
              #
53  #      Descripción: imprime la cadena que recibe en el
    fichero          #
54  #         correspondiente según el nivel del texto a imprimir
              #
55  #      Argumentos: texto --> cadena a imprimir
              #
56  #                     nivel --> Nivel del texto
              #
57  #                           I --> Información en log
              #
58  #                           TI -> Información en terminal e log
              #
59  ##############################################################
    ###########
60  function log_awk(texto,nivel)
61  {
62        if (nivel == "I") {
63              # Pinta en log.txt
64              print "[INFO] ---- " texto >> fich_log
65        } else if (nivel == "TI") {
66              # Pinta en log.txt y en el terminal
67              print "[INFO] ---- " texto >> fich_log
68              print "[INFO] ---- " texto
69        } else if (nivel == "E") {
70              # Pinta en log.txt y en el terminal
71              print "[ERROR]" texto >> fich_log
72              print "[ERROR]" texto
73              exit 1
74        }
75  }
76
77  ##############################################################
    ###########
```

```
78  #               Nombre función: carga_informe_detector
          #
79  #      Descripción: carga el informe del detector en un array
          #
80  ##############################################################
    ##########
81  function carga_informe_detector()
82  {
83        getline error < informe_det
84        fichero_anterior=""
85        linea_anterior=""
86
87        while (getline error < informe_det) {
88              split(error,err_campos,";")
89              fichero=err_campos[1]
90              linea=err_campos[2]
91              subregla=err_campos[4]
92              contador_arreglos=0
93
94              if (fichero in array_errores && linea in
    array_errores[fichero]) {
95        contador_arreglos=length(array_errores[fichero][linea]
    )
96              }
97              fichero_anterior=fichero
98              linea_anterior=linea
99
100             array_errores[fichero][linea][contador_arreglos]=
    subregla
101       }
102       close(informe_det)
103 }
104 ##############################################################
    ##########
105 #                   Nombre función: aplica_reglas
            #
106 #      Descripción: modifica la línea del fichero aplicando
    la          #
107 #                          subregla correspondiente
            #
108 #      Argumentos: linea  --> línea a tratar
            #
109 #                   fichero  --> fichero al que pertenece la
    linea      #
110 #                   num_linea  --> número de línea de línea
            #
111 ##############################################################
    ##########
112 function aplica_reglas(linea, fichero, num_linea)
113 {
114       linea_fix[1]=linea
115
116       if(fichero in array_errores && num_linea in
```

```
    array_errores[fichero]) {
117             for(num_subreglas in
    array_errores[fichero][num_linea]) {
118                 subregla =
    array_errores[fichero][num_linea][num_subreglas]
119                 if (subregla in arreglos){
120
121                     for(indice_subregla in
    arreglos[subregla]) {
122                         linea_fix[1] =
    gensub(arreglos[subregla][indice_subregla]["regexp"],
    arreglos[subregla][indice_subregla]["replac"], "g",
    linea_fix[1])
123                     }
124                 } else {
125                     log_awk("Subregla " subregla " no
    definida en
    corrector","TI")
126                 }
127                 n_correcciones++
128             }
129             split(linea_fix[1],linea_fix,"\n")
130         }
131
132         for (ind in linea_fix) {
133             escribe_linea(linea_fix[ind], fichero)
134             delete linea_fix[ind]
135         }
136     }
137
138     ##########################################################
    ##########
139     #                 Nombre función: escribe_linea
            #
140     #     Descripción: escribe la línea en el fichero
    correspondiente    #
141     #     Argumentos: linea  --> línea a escribir
            #
142     #                 fichero  --> fichero en el que escribir
            #
143     #                 identacion --> número de espacios de
    identación    #
144     ##########################################################
    ##########
145     function escribe_linea(linea, fichero)
146     {
147         sub(ruta_ficheros"\\/","",fichero)
148         fichero=ruta_corregidos"/"fichero
149
150         if (linea !~ /^\s*_NO-IMPRIMIR_$/) {
151             if (linea ~ /_ELIM-RETORNO_/) {
152                 sub(/\s*_ELIM-RETORNO_/," ",linea)
153                 printf "%s" , linea >> fichero
```

```
154             } else {
155                 print linea >> fichero
156             }
157         }
158     }
159
160     BEGIN {
161         n_correcciones=0
162
163         carga_informe_detector()
164         genera_array_arreglos()
165     }{
166         aplica_reglas($0, FILENAME, FNR)
167     }
168
169     END {
170         log_awk("Se han realizado un total de " n_correcciones
    "
171     correcciones","TI")
172     }' $WORKSHOP
173
174
175     ret_corrector=$?
176     if [[ $ret_corrector == 0 ]]
177     then
178         log "CORRECTOR FINALIZÓ SIN INCIDENCIAS" "TI"
179         if [[ $SOBRESCRIBIR == "SI" ]]
180         then
181             rm -rf $RUTA_FICHEROS
182             log "${RUTA_FICHEROS} eliminado" "I"
183             mv $RUTA_CORREGIDOS $RUTA_FICHEROS
184             log "${RUTA_CORREGIDOS} es ahora
    ${RUTA_FICHEROS}" "I"
185             RUTA_CORREGIDOS=$RUTA_FICHEROS
186         fi
187     else
188         log "018 CORRECTOR NO FINALIZÓ CORRECTAMENTE" "E"
189     fi
```

ANEXO II. CONVERSOR DE ESTILO DE CÓDIGO C/C++

Este anexo contiene el código de cada archivo de la herramienta, correspondiente a la implementación del conversor de estilo de C/C++, estos archivos son:

- ajustes_TC.cfg
- reglas_TC.cfg
- lanzador_TC.cfg
- lib_TC.sh
- detector_TC.sh
- corrector_TC.sh

ajustes_TC.cfg

```
1  RUTA_FICHEROS="entrada/correg_orig"
2  SOBRESCRIBIR=NO
3  LONG_MAX=120
4  SECUENCIA=131
```

reglas_TC.cfg

```
1  R1=SI
2  R2=SI
3  R3=SI
4  R4=SI
5  R5=SI
6  R6=SI
7  R7=SI
8  R8=SI
9  R9=SI
10 R10=SI
11 R11=SI
```

lanzador_TC.sh

```
1  #!/bin/bash
2  #MOD_REQ#
3
4  # CONSTANTES
5  # Número de reglas que generan reporte
6  export NUM_REG_REP=3 #MOD_REQ#
7  export MI_PATH=$(pwd)
8  # VARIABLES
9  export LIBRERIA="${MI_PATH}/scripts/lib_TC.sh"
10 export LOG="log_TC"
11 # FUNCIONES
12
13 ##############################################################
   ##########
14 #                    Nombre función: check_lib
           #
15 #    Descripción: comprueba que lib_TC.sh existe y tiene
   permisos    #
16 ##############################################################
   ##########
17 function check_lib()
18 {
19       if [ -d scripts ]
20       then
21             echo "[INFO] ---- Existe directorio de scripts"
   >> $LOG
22             if [ -f $LIBRERIA ]
23             then
24                   echo "[INFO] ---- lib_TC.sh existe" >> $LOG
25                   if [ -x $LIBRERIA ]
26                   then
27                         echo "[INFO] ---- lib_TC.sh tiene
   permisos de ejecución" >> $LOG
28                   else
29                         chmod +x $LIBRERIA
30                         echo "[INFO] ---- Permisos otorgados a
   lib_TC.sh" | tee -a $LOG
31                   fi
32             else
33                   echo "[ERROR] 002 lib_TC.sh no existe" | tee
   -a $LOG
34                   exit 1;
35             fi
36       else
37             echo "[ERROR] 001 Directorio scripts no
   encontrado" | tee -a $LOG
38             exit 1;
39       fi
40 }
41
42 # Por si no lo hemos movido anteriormente
43 [ -f $LOG ] && rm $LOG
```

```
44
45  ##############################################################
    ##########
46  #                       Inicio de lanzador_TC.sh
            #
47  ##############################################################
    ##########
48
49  echo "
    ##############################################################
    ####" | tee -a $LOG
50  echo "             #         Herramienta de transformación de
    código fuente         #" | tee -a $LOG
51  echo "             #
                       #" | tee -a $LOG
52  echo "             #      Ejemplo de aplicación: código C/C++
                       #" | tee -a $LOG
53  echo "             #                   a código C/C++ con un
    estilo concreto          #" | tee -a $LOG
54  echo "             #
                       #" | tee -a $LOG
55  echo "             #        Trabajo de Fin de Grado de Álvaro
    Fraile Cáceres        #" | tee -a $LOG
56  echo "             #                Tutores:
                       #" | tee -a $LOG
57  echo "             #                    - Podar Cristea,
    Smaranda                    #" | tee -a $LOG
58  echo "             #                    - Mazaeda Echevarria,
    Rogelio                  #" | tee -a $LOG
59  echo "
    ##############################################################
    ####" | tee -a $LOG
60
61  ##############################################################
    ##########
62  #                             Código
            #
63  ##############################################################
    ##########
64
65  check_lib
66
67  # source nos "pasa las variables de hijo a padre"
68  source $LIBRERIA
69
70  # Comprobamos que todos los directorios están creados
71  check_dir
72
73  # Extraemos las variables de configuración
74  check_conf
75
76  # Convertimos los archivos a tratar en formato UNIX
77  dos2unix -k -q $WORKSHOP
```

```
78
79  # Comprobamos que los scripts tienen permisos de ejecución
80  check_perm
81
82  # Comprobamos que los parámetros de ejecución son correctos
83  check_param $1 $#
84
85  # Obtenemos las reglas activas
86  check_reglas
87
88  # Creamos el directorio de la secuencia actual
89  crea_dir_seq
90
91  if [[ $1 == "D" ]]
92  then
93        . $DETECTOR
94        log "La herramienta finalizó correctamente" "TI"
95  elif [[ $1 == "DC" ]]
96  then
97        . $DETECTOR
98        . $CORRECTOR
99
100       log "La herramienta finalizó correctamente" "TI"
101       log "Los ficheros modificados se encuentran en
    ${RUTA_CORREGIDOS}" "TI"
102 elif [[ $1 == "R" ]]
103 then
104       for (( i=1; i <= (${#regla[*]} - $NUM_REG_REP); i++ ))
105       do
106             regla[$i]=0
107       done
108       log "Reglas de corrección desactivadas" "TI"
109       . $DETECTOR
110
111       log "La herramienta finalizó correctamente" "TI"
112 fi
113
114
115 exit 0;
```

lib_TC.sh

```
1  ##############################################################
   ##########
2  #                          Inicio de lib_TC.sh
           #
3  ##############################################################
   ##########
4  # CONSTANTES
5  export CONFIG="${MI_PATH}/config"
6  export CFG_AJUSTES="${CONFIG}/ajustes_TC.cfg"
7  export CFG_REGLAS="${CONFIG}/reglas_TC.cfg"
8  export CORRECTOR="${MI_PATH}/scripts/corrector_TC.sh"
9  export DETECTOR="${MI_PATH}/scripts/detector_TC.sh"
10 export SALIDA="${MI_PATH}/salida"
11 export ENTRADA="${MI_PATH}/entrada"
12
13 # VARIABLES
14 export REPORTE="reporte_TC"
15 export INFORME_DET="informe_detector_TC"
16 export RUTA_FICHEROS=""
17 export SOBRESCRIBIR=""
18 export LONG_MAX=""
19 export WORKSHOP=""
20 export SECUENCIA=""
21 export regla
22
23 ##############################################################
   ##########
24 #                    Nombre función: log
           #
25 #     Descripción: imprime la cadena que recibe en el
   fichero         #
26 #       correspondiente según el nivel del texto a imprimir
           #
27 #     Argumentos: $1 --> cadena a imprimir
           #
28 #                 $2 --> Nivel del texto
           #
29 #                       I --> Información en log
           #
30 #                       TI -> Información en terminal e log
           #
31 #                       E --> Error
           #
32 ##############################################################
   ##########
33 function log()
34 {
35       case $2 in
36            "I")
37                  # Pinta en log_TC.txt
38                  echo "[INFO] ----" $1 >> $LOG
39                  ;;
```

```
40              "TI")
41                    # Pinta en log_TC.txt y en el terminal
42                    echo "[INFO] ----" $1 | tee -a $LOG
43                    ;;
44              "E")
45                    # Pinta en log_TC.txt y en el terminal
46                    echo "[ERROR]" $1 | tee -a $LOG
47                    exit 1;
48                    ;;
49              *)
50                    # Error en el parámetro
51                    echo "[ERROR] 000 Parámetro de nivel
    incorrecto" | tee -a $LOG
52                    exit 1;
53                    ;;
54        esac
55  }
56
57  ##############################################################
    ##########
58  #                    Nombre función: ayuda_lanz
              #
59  #     Descripción: muestra los posibles parámetros de
    ejecución       #
60  ##############################################################
    ##########
61  function ayuda_lanz()
62  {
63        log "Parámetros de ejecución admitidos:" "I"
64        echo "                  D --> Solo detección"
65        echo "                  DC --> Detección y corrección"
66        echo "                  R --> Solo reportes de warnings"
67        echo "                  --help --> Ayuda"
68  }
69
70  ##############################################################
    ##########
71  #                    Nombre función: check_dir
              #
72  #        Descripción: comprueba la existencia de directorios
              #
73  ##############################################################
    ##########
74  function check_dir()
75  {
76        if [ -d $ENTRADA ]
77        then
78              log "Existe directorio de entrada" "I"
79        else
80              log "Directorio de entrada creado" "TI"
81              mkdir $ENTRADA
82        fi
83
```

```
84          if [ -d $SALIDA ]
85          then
86                log "Existe directorio de salida" "I"
87          else
88                log "Directorio de salida creado" "TI"
89                mkdir $SALIDA
90          fi
91
92          if [ -d $CONFIG ]
93          then
94                log "Existe directorio de configuración" "I"
95                if [ -f $CFG_AJUSTES ]
96                then
97                      log "ajustes_TC.cfg existe" "I"
98                else
99                      log "004 ajustes_TC.cfg no existe" "E"
100               fi
101               if [ -f $CFG_REGLAS ]
102               then
103                     log "reglas_TC.cfg existe" "I"
104               else
105                     log "005 reglas_TC.cfg no existe" "E"
106               fi
107         else
108               log "003 Directorio config no encontrado" "E"
109         fi
110   }
111
112   ##############################################################
      ##########
113   #                    Nombre función: check_conf
            #
114   #  Descripción: extrae las variables necesarias del fichero
      de conf   #
115   ##############################################################
      ##########
116   function check_conf()
117   {
118         . $CFG_AJUSTES
119         if [ -z $RUTA_FICHEROS ]
120         then
121               log "006 Ruta de ficheros a modificar no
      especificada" "E"
122         else
123               if [ -d $RUTA_FICHEROS ]
124               then
125                     log "Ruta de ficheros a modificar:
      ${RUTA_FICHEROS}" "TI"
126                     if [[ $MI_PATH != $RUTA_FICHEROS ]]
127                     then
128                           WORKSHOP=$(find ${RUTA_FICHEROS} -
      iname "*.c" -or -iname "*.cpp" -or -iname "*.h" -or -iname
      "*.hpp") #MOD_REQ#
```

```
129                     if [[ -z $WORKSHOP ]]
130                     then
131                         log "009 No se encontraron
    ficheros que modificar" "E"
132                     else
133                         n_fich=$(find ${RUTA_FICHEROS} -
    iname "*.c" -or -iname "*.cpp" -or -iname "*.h" -or -iname
    "*.hpp" | wc -l) #MOD_REQ#
134                         log "${n_fich} fichero(s) a
    modificar encontrado(s)" "TI"
135                     fi
136                 else
137                     log "008 La ruta de los ficheros a
    modificar no es válida" "E"
138                 fi
139             else
140                 log "007 Ruta de ficheros a modificar no
    existe" "E"
141             fi
142         fi
143
144         if [[ $SOBRESCRIBIR == "SI" ]]
145         then
146             log "Sobreescritura de archivos activada" "TI"
147             log "¿Seguro que quiere SOBRESCRIBIR los
    archivos?" "TI"
148             log "Escriba NO para abandonar o cualquier otra
    cadena para continuar" "TI"
149             read respuesta
150             log "Escribió: ${respuesta}" "I"
151             if [[ $respuesta == "NO" ]]
152             then
153                 log "Operación cancelada, abandonando
    herramienta" "TI"
154                 exit 2
155             fi
156         elif [[ $SOBRESCRIBIR == "NO" ]]
157         then
158             log "Sobreescritura de archivos desactivada" "TI"
159         else
160             log "010 Variable SOBRESCRIBIR debe ser SI o NO"
    "E"
161         fi
162
163         if [[ $LONG_MAX =~ ^[0-9]+$ ]]
164         then
165             log "Patrón de longitud máxima OK" "I"
166         else
167             log "019 Variable LONG_MAX debe de ser un número"
    "E"
168         fi
169
170         if [[ $SECUENCIA =~ ^[0-9]+$ ]]
```

```
171         then
172               log "Patrón de secuencia OK" "I"
173               SECUENCIA=$(($SECUENCIA + 1))
174               log "Número de secuencia actualizado a
    ${SECUENCIA}" "TI"
175               rm $CFG_AJUSTES
176               echo "RUTA_FICHEROS=\""$RUTA_FICHEROS"\"" >>
    $CFG_AJUSTES
177               echo "SOBRESCRIBIR="$SOBRESCRIBIR >> $CFG_AJUSTES
178               echo "LONG_MAX="$LONG_MAX >> $CFG_AJUSTES
179               echo "SECUENCIA="$SECUENCIA >> $CFG_AJUSTES
180         else
181               log "011 Variable SECUENCIA debe de ser un
    número" "E"
182         fi
183
184   }
185
186   ##############################################################
    ##########
187   #                    Nombre función: check_perm
              #
188   #      Descripción: comprueba que los scripts tienen permisos
              #
189   ##############################################################
    ##########
190   function check_perm()
191   {
192         if [ -f $DETECTOR ]
193         then
194               log "detector_TC.sh existe" "I"
195               if [ -x $DETECTOR ]
196               then
197                     log "detector_TC.sh tiene permisos de
    ejecución" "I"
198               else
199                     chmod +x $DETECTOR
200                     log "Permisos otorgados a detector_TC.sh"
    "TI"
201               fi
202         else
203               log "012 detector_TC.sh no existe" "E"
204         fi
205
206         if [ -f $CORRECTOR ]
207         then
208               log "corrector_TC.sh existe" "I"
209               if [ -x $CORRECTOR ]
210               then
211                     log "corrector_TC.sh tiene permisos de
    ejecución" "I"
212               else
213                     chmod +x $CORRECTOR
```

```
214                 log "Permisos otorgados a corrector_TC.sh"
    "TI"
215           fi
216     else
217           log "013 corrector_TC.sh no existe" "E"
218     fi
219 }
220
221
222
223 ##############################################################
    ##########
224 #                    Nombre función: check_param
            #
225 #   Descripción: comprueba que los parámetros del lanzador
    son OK    #
226 #     Argumentos: $1 --> $1 de ejecución
            #
227 #                  $2 --> $# número de parámetros de
    ejecución         #
228 ##############################################################
    ##########
229 function check_param()
230 {
231     if [[ $2 == "1" ]]
232     then
233           log "Número de parámetros de ejecución correctos"
    "I"
234           case $1 in
235           "D")
236                 log "Lanzador en modo detección" "TI"
237                 ;;
238           "DC")
239                 log "Lanzador en modo detección y
    corrección" "TI"
240                 ;;
241           "R")
242                 log "Lanzador en modo reportes " "TI"
243                 ;;
244           "--help")
245                 log "Ayuda solicitada en ejecución" "TI"
246                 ayuda_lanz
247                 exit 1;
248                 ;;
249           *)
250                 log "014 Parámetros de ejecución
    incorrectos, consulte la ayuda con --help" "E"
251                 ;;
252           esac
253     else
254           log "014 Parámetros de ejecución incorrectos,
    consulte la ayuda con --help" "E"
255     fi
```

```
256 }
257
258 ##############################################################
    ##########
259 #                    Nombre función: check_reglas
            #
260 #  Descripción: extrae las variables necesarias del fichero
    de conf  #
261 ##############################################################
    ##########
262 function check_reglas()
263 {
264       i=0
265       for r in $(cat ${CFG_REGLAS})
266       do
267             i=$(($i + 1))
268             if [[ $r =~ R[0-9]+\=NO$ ]]
269             then
270                   regla[$i]=0
271             elif [[ $r =~ R[0-9]+\=SI$ ]]
272             then
273                   regla[$i]=1
274                   log "Regla ${i} activa" "TI"
275             else
276                   log "015 Error en reglas_TC.cfg: las reglas
    deben estar a SI o NO" "E"
277             fi
278       done
279 }
280
281 ##############################################################
    ##########
282 #                    Nombre función: crea_dir_seq
            #
283 #  Descripción: crea el directorio con la secuencia
    correspondiente  #
284 ##############################################################
    ##########
285 function crea_dir_seq()
286 {
287       # Directorio de la salida para esta secuencia
288       export SAL_SEQ="${SALIDA}/${SECUENCIA}"
289
290       # Sobreescribimos en caso de que ya haya una carpeta
    con la secuencia actual
291       [ -d $SAL_SEQ ] && rm -rf $SAL_SEQ && log "Directorio
    de salida sobreescrito (n° de secuencia repetido)" "TI"
292
293       # Creamos el directorio donde se guardarán los
    archivos
294       mkdir $SAL_SEQ
295
296       # Movemos el log a su nueva ubicación
```

```
297         [ -f $LOG ] && mv $LOG
    "${SAL_SEQ}/${LOG}_${SECUENCIA}.txt"
298
299         # Incluímos el PATH en el nombre de los archivos
300         LOG="${SAL_SEQ}/${LOG}_${SECUENCIA}.txt"
301         INFORME_DET="${SAL_SEQ}/${INFORME_DET}_${SECUENCIA}.cs
    v"
302         REPORTE="${SAL_SEQ}/${REPORTE}_${SECUENCIA}.csv"
303   }
304
305   ##############################################################
    ##########
306   #                    Nombre función: crea_dir_correg
              #
307   #  Descripción: crea los directorios de la ruta de ficheros
    escogida #
308   ##############################################################
    ##########
309   function crea_dir_correg
310   {
311         if [[ $SOBRESCRIBIR == "SI" ]]
312         then
313               RUTA_CORREGIDOS=$RUTA_FICHEROS"_copy"
314               mkdir $RUTA_CORREGIDOS
315               log "Directorio ${RUTA_CORREGIDOS} creado" "I"
316
317               for i in $WORKSHOP
318               do
319                     fich="${i#${RUTA_FICHEROS}}"
320                     dir="${fich%/*}"
321
322                     mkdir -p "${RUTA_CORREGIDOS}/${dir}"
323               done
324         elif [[ $SOBRESCRIBIR == "NO" ]]
325         then
326               export CORREGIDOS="${SAL_SEQ}/corregidos"
327               mkdir $CORREGIDOS
328               log "Directorio ${CORREGIDOS} creado" "I"
329               export ORIGINALES="${SAL_SEQ}/originales"
330               mkdir  $ORIGINALES
331               log "Directorio ${ORIGINALES} creado" "I"
332
333               for i in $WORKSHOP
334               do
335                     fich="${i#${RUTA_FICHEROS}}"
336                     dir="${fich%/*}"
337
338                     mkdir -p "${CORREGIDOS}/${dir}"
339                     mkdir -p "${ORIGINALES}/${dir}"
340                     cp -ra $i "${ORIGINALES}${fich}"
341               done
342
343               RUTA_CORREGIDOS=$CORREGIDOS
```

```
344         fi
345   }
346
347   ##############################################################
      ##########
348   #                    Nombre función: check_informe_det
            #
349   #  Descripción: comprueba que el informe del detector trae
      al menos  #
350   #                    1 error
            #
351   ##############################################################
      ##########
352   function check_informe_det
353   {
354         if [[ $(cat ${INFORME_DET}) =~ [0-9] ]]
355         then
356               log "Los ficheros contienen errores" "TI"
357               return 1
358         elif (( regla[1] == 1 ))
359         then
360               log "Los ficheros no contienen errores, R1
      activa" "TI"
361               return 1
362         else
363               log "Los ficheros no contienen errores" "TI"
364               return 0
365         fi
366
367   }
368
369   # FUNCIONES
370   export -f log
371   export -f ayuda_lanz
372   export -f check_dir
373   export -f check_conf
374   export -f check_perm
375   export -f check_param
376   export -f check_reglas
377   export -f crea_dir_seq
378   export -f crea_dir_correg
379   export -f check_informe_det
```

detector_TC.sh

```
1  ##############################################################
   ##########
2  #                        Inicio de detector_TC.sh
           #
3  ##############################################################
   ##########
4
5
6  log "INICIALIZANDO DETECTOR" "TI"
7
8  echo "Fichero;Linea;Regla;Subregla" >> $INFORME_DET
9  echo "Fichero;Linea;Regla;Regla Reporte;Incidencia" >>
   $REPORTE
10
11 awk -b -v reporte=$REPORTE -v informe_det=$INFORME_DET -v
   fich_log=$LOG -v reglas="${regla[*]}" -v
   num_reg_rep=$NUM_REG_REP -v long_max=$LONG_MAX '
12 ##############################################################
   ##########
13 #                     Nombre función: log_awk
           #
14 #     Descripción: imprime la cadena que recibe en el
   fichero         #
15 #       correspondiente según el nivel del texto a imprimir
           #
16 #     Argumentos: texto --> cadena a imprimir
           #
17 #                   nivel --> Nivel del texto
           #
18 #                         I --> Información en log
           #
19 #                         TI -> Información en terminal e log
           #
20 #                         ID -> Arreglos en
   informe_detector.csv          #
21 #                         R --> Reporte generado por regla
           #
22 ##############################################################
   ##########
23 function log_awk(texto,nivel)
24 {
25       if (nivel == "I") {
26             # Pinta en log_TC.txt
27             print "[INFO] ---- " texto >> fich_log
28       } else if (nivel == "TI") {
29             # Pinta en log_TC.txt y en el terminal
30             print "[INFO] ---- " texto >> fich_log
31             print "[INFO] ---- " texto
32       } else if (nivel == "E") {
33             # Pinta en log.txt y en el terminal
34             print "[ERROR]" texto >> fich_log
35             print "[ERROR]" texto
```

```
36              exit 1
37        } else if (nivel == "D") {
38              # Pinta en informe_detector.csv
39              print texto >> informe_det
40        } else if (nivel == "R") {
41              # Pinta en reportes.csv
42              if (texto ~ /^[^;]+;[^;]+;[^;]+;1;$/) {
43                    texto=texto "Variable no inicializada"
44              } else if (texto ~ /^[^;]+;[^;]+;[^;]+;2;$/) {
45                    texto=texto "La linea sobrepasa el limite de
    longitud de " long_max
46              } else if (texto ~ /^[^;]+;[^;]+;[^;]+;3;$/) {
47                    texto=texto "Funcion sin comentario
    explicativo"
48              }
49              print texto >> reporte
50        }
51  }
52  ##############################################################
    ##########
53  #                   Nombre función: elimina_coment
              #
54  #      Descripción: elimina los comentarios de línea
              #
55  #      Argumentos: linea  --> línea a tratar
              #
56  #      Return: línea modificada, ahora sin comentarios
              #
57  ##############################################################
    ##########
58  function elimina_coment(linea)
59  {
60        if (linea ~ /\/\//) {
61              sub(/\s*\/\/.*/,"",linea)
62        }
63
64        if (linea ~ /\".+\"/) {
65              sub(/\".+\"/,"",linea)
66        }
67
68        if (linea ~ /\/\*/ || coment_rang) {
69              if (linea ~ /\/\*/ && linea ~ /\*\//) {
70                    sub(/\/\*.*\*\//,"",linea)
71                    coment_rang=0
72              } else if (linea ~ /\/\*/) {
73                    sub(/\/\*.*/,"",linea)
74                    coment_rang=1
75              } else if (linea ~ /\*\//) {
76                    sub(/.*\*\//,"",linea)
77                    coment_rang=0
78              } else {
79                    sub(/^.*$/,"",linea)
80              }
```

```
81        }
82
83        return linea
84  }
85  ##############################################################
    ###########
86  #                   Nombre función: cuenta_ocurrencias
              #
87  #      Descripción: devuelve el nº de ocurrencias que encajan
     con      #
88  #                           regexp
              #
89  #      Argumentos: regexp --> expresión regular
              #
90  #                   linea  --> línea a tratar
              #
91  #      Return: número de ocurrencias
              #
92  ##############################################################
    ###########
93  function cuenta_ocurrencias(regexp,linea)
94  {
95        linea_aux=linea
96        for (iterador=0; linea_aux ~ regexp; iterador++) {
97              sub(regexp,"",linea_aux)
98        }
99        return iterador
100 }
101 ##############################################################
    ###########
102 #                     Nombre función: mi_getline
              #
103 #      Descripción: devuelve la cadena correspondiente al
    fichero      #
104 #                           y linea dados
              #
105 #      Argumentos: fichero --> nombre del fichero
              #
106 #                   n_linea --> nº de línea
              #
107 #      Return: línea leída
              #
108 ##############################################################
    ###########
109 function mi_getline(fichero, n_linea)
110 {
111           cont=0
112           do {
113              cont++
114              getline linea_getline < fichero
115           } while (cont<n_linea)
116              close(fichero)
117              linea_getline=elimina_coment(linea_getline)
```

```
118             return linea_getline
119     }
120
121     BEGIN{
122           flag_R7=0
123           linea_anterior=""
124
125           # Expresión regular que matchea con los tipos de datos
      que puede ser una variable
126           tipos_var="^\\s*(const)?\\s*(unsigned\\s+|signed\\s+|c
      har\\s+|auto\\s+|bool\\s+|short\\s+|long\\s+|int\\s+|float\\
      s+|double\\s+)\\s*(*|&)?\\s*"
127
128           # Expresión regular que matchea con los tipos de datos
      que puede devolver una función
129           tipos_func="^\\s*(const|inline)?\\s*(unsigned\\s+|sign
      ed\\s+|char\\s+|auto\\s+|bool\\s+|short\\s+|long\\s+|int\\s+
      |float\\s+|double\\s+|void\\s+)\\s*(*|&)?\\s*"
130
131           # Encaja con paréntesis que tengan dentro de ellos
      paréntesis emparejados
132           parent_par="\\s*\\([^\\(\\)]*(\\([^\\(\\)]*\\)[^\\(\\)
      ]*)*\\)"
133
134           # Las reglas a 1 y 0 están en regla
135           n_reglas=split(reglas,regla," ")
136
137           # Inicializamos el array con lo errores de cada regla
138           for (i in regla) {
139                 n_errores[i]=0
140           }
141           n_errores_totales=0
142     }{
143           linea_mod=elimina_coment($0)
144
145           ##################################################
      ############
146           #                              REGLA 1
                  #
147           ##################################################
      ############
148           if (regla[1] == 1) {
149                 if ($0 ~ /^\s*{(\s*{\s*)+$/) {
150                       log_awk(FILENAME";"FNR";1;0101;","D")
151                       n_errores[1]++
152                 }
153                 if ($0 ~ /^\s*}(\s*}\s*)+$/) {
154                       log_awk(FILENAME";"FNR";1;0102;","D")
155                       n_errores[1]++
156                 }
157           }
158
159           ##################################################
```

```
############
      #                                 REGLA 2
            #
      ##########################################################
############
      if (regla[2] == 1) {
            if ($0 ~ /\t/) {
                  log_awk(FILENAME";"FNR";2;0201;","D")
                  n_errores[2]++
            }
      }

      ##########################################################
############
      #                                 REGLA 3
            #
      ##########################################################
############
      if (regla[3] == 1) {
      regexp_bucles="((if|for|while)\\s*\\(.*|}?\\s*else\\s*
(if\\s*\\(.*)?|do)(\\s+|$)"
            if (linea_mod ~ "^\\s*{?\\s*"regexp_bucles &&
linea_mod !~ "(\\)|else\\s*|do)\\s*{") {

      n_parent=cuenta_ocurrencias("\\(",linea_mod)-
cuenta_ocurrencias("\\)",linea_mod)
                  if (n_parent != 0) {
                        a=0
                        do {
                              a++

      linea_getl=mi_getline(FILENAME,FNR+a)

      n_parent+=cuenta_ocurrencias("\\(",linea_getl)-
cuenta_ocurrencias("\\)",linea_getl)
                              if (a >= 1000 ) {
                                    log_awk(" 020  Error en
detección de la regla 3 para el fichero "FILENAME,"E")
                              }
                        } while (n_parent != 0);
                        num_linea=FNR+a
                        linea_bucle=linea_getl
                  } else {
                        num_linea=FNR
                        linea_bucle=linea_mod
                  }


                  if (linea_bucle ~ /;\s*$/ && linea_bucle !~
/{/) {
                        if (linea_bucle !~
"while\\s*"parent_par"\\s*;") {
```

```
      log_awk(FILENAME";"num_linea";3;0304;","D")
197                         a=1
198                         do {
199
      linea_getl=mi_getline(FILENAME,num_linea+a)
200                               if (linea_getl ~
/^\s*(else|while\s*\(.*\s*;)/ && linea_mod ~
/((else\s*)?if\s*\(.*|do)/) {
201
            log_awk(FILENAME";"num_linea+a";3;0303;","D")
202                                     break
203                               } else if (linea_getl !~
/^\s*$/) {
204
      log_awk(FILENAME";"num_linea";3;0302;","D")
205                                     break
206                               }
207                               a++
208                               if (a >= 1000 ) {
209                                     log_awk(" 020  Error
en detección de la regla 3 para el fichero "FILENAME,"E")
210                               }
211                         } while (linea_getl ~ /^\s*$/);
212                         n_errores[3]++
213                   }
214                } else if (linea_bucle !~ /\)\s*{/ &&
linea_bucle !~ /else\s*{/ && linea_bucle !~ /do\s*{/) {
215                   i=1
216                   # Buscar corchete
217                   do {
218
      linea_getl=mi_getline(FILENAME,num_linea+i)
219                         if (linea_getl ~ /(^\s*{|{\s*$)/
&& ( linea_getl !~ "^\\s*{?\\s*"regexp_bucles || linea_getl
~ "^\\s*{\\s*"regexp_bucles)) {
220                               break
221                         } else if (linea_mod ~
/^\s*}?\s*else\s*$/ && linea_getl ~ /^\s*if\s*\(.*/) {
222                               for (a=0; a < i; a++) {
223
      log_awk(FILENAME";"FNR+a";3;0305;","D")
224                               }
225
      log_awk(FILENAME";"FNR+i";1;0103;","D")
226                               n_errores[3]++
227                               break
228                         } else if (linea_getl !~ /^\s*$/)
 {
229                               j=i
230                               # Ver si tenemos un bucle o
es una sentencia
231                               do {
232
```

```
        linea_getl=mi_getline(FILENAME,num_linea+j)
233                                     if (linea_getl ~
    /(;|})\s*$/) {
234
        log_awk(FILENAME";"num_linea";3;0301;","D")
235                                         a=j+1
236                                         do {
237
        linea_getl=mi_getline(FILENAME,num_linea+a)
238                                             if
    (linea_getl ~ /^\s*(else|while\s*\(.*\s*;)/ && linea_mod ~
    /((else\s*)?if\s*\(.*|do)/) {
239
        log_awk(FILENAME";"num_linea+a";3;0303;","D")
240                                                 break
241                                             } else if
    (linea_getl !~ /^\s*$/) {
242
        log_awk(FILENAME";"num_linea+j";3;0302;","D")
243                                                 break
244                                             }
245                                             a++
246                                             if (a >=
    1000 ) {
247
        log_awk(" 020 Error en detección de la regla 3 para el
    fichero "FILENAME,"E")
248                                             }
249                                         } while (1);
250                                         break
251                                     } else if (linea_getl
    ~ /^\s*$/ || linea_getl ~ "^\\s*{?\\s*"regexp_bucles) {
252                                         k=j
253                                         # Buscamos fin
    del bucle
254                                         do {
255
        linea_getl=mi_getline(FILENAME,num_linea+k)
256                                             if
    (linea_getl ~ /(^\s*{|{\s*$)/) {
257                                                 l=k
258
        n_llaves=0
259                                                 #
    Buscamos fin de bucle con llave
260                                                 do {
261
        linea_getl=mi_getline(FILENAME,num_linea+l)
262
        n_llaves+=cuenta_ocurrencias("{",linea_getl)-
    cuenta_ocurrencias("}",linea_getl)
263
        if (n_llaves <= 0) {
```

```
264
        log_awk(FILENAME";"num_linea";3;0301;","D")
265
        a=l+1
266
        do {
267
              linea_getl=mi_getline(FILENAME,num_linea+a)
268
              if (linea_getl ~ /^\s*(else|while\s*\(.*\s*;)/ &&
    linea_mod ~ /((else\s*)?if\s*\(.*|do)/) {
269
        log_awk(FILENAME";"num_linea+a";3;0303;","D")
270
                    break
271
              } else if (linea_getl !~ /^\s*$/) {
272
        log_awk(FILENAME";"num_linea+l";3;0302;","D")
273
                    break
274
              }
275
              a++
276
              if (a >= 1000 ) {
277
                    log_awk(" 020 Error en detección de la regla
    3 para el fichero "FILENAME,"E")
278
              }
279
        } while (1);
280
        break
281                                                                  }
282
        l++
283
        if (l >= 1000 ) {
284
        log_awk(" 020 Error en detección de la regla 3 para el
    fichero "FILENAME,"E")
285                                                                  }
286                                                               }
    while (n_llaves >= 0);
287                                                                 break
288                                                            } else if
    (linea_getl ~ /(;|})\s*$/) {
289
```

```
        log_awk(FILENAME";"num_linea";3;0301;","D")
                                        a=k+1
                                        do {
        linea_getl=mi_getline(FILENAME,num_linea+a)
        if (linea_getl ~ /^\s*(else|while\s*\(.*\s*;)/ && linea_mod ~ /((else\s*)?if\s*\(.*|do)/) {
        log_awk(FILENAME";"num_linea+a";3;0303;","D")
        break
                                            }
    else if (linea_getl !~ /^\s*$/) {
        log_awk(FILENAME";"num_linea+k";3;0302;","D")
        break
                                            }
        a++
        if (a >= 1000 ) {
        log_awk(" 020 Error en detección de la regla 3 para el fichero "FILENAME,"E")
                                            }
                                        }
    while (1);
                                        break
                                    }
                                    k++
                                    if (k >= 1000 ) {
        log_awk(" 020 Error en detección de la regla 3 para el fichero "FILENAME,"E")
                                    }
                                } while (1);
                                break
                            }
                            j++
                            if (j >= 1000 ) {
                                log_awk(" 020 Error en detección de la regla 3 para el fichero "FILENAME,"E")
                            }
                        } while (1);
                        n_errores[3]++
                        break
                    }
                    i++
                    if (i >= 1000 ) {
```

```
324                                     log_awk(" 020 Error en
detección de la regla 3 para el fichero "FILENAME,"E")
325                                }
326                           } while (1);
327                     }
328               }
329          }
330
331          ##################################################
#############
332          #                                REGLA 4
                  #
333          ##################################################
#############
334          if (regla[4] == 1) {
335
             exp_reg_apert="((if|for|while|catch|switch)\\s*\\(.*|e
lse\\s*(if\\s*\\(.*)?|do\\s*{?\\s*$|try\\s*{?\\s*$|case.*:|d
efault:)"
336
             exp_reg_cierre="^\\s*}?\\s*(catch\\s*\\(.*|while\\s*\\
(.*\\)\\s*;|else\\s*(if\\s*\\(.*)?)"
337
338               if (linea_mod ~
"^\\s*{?}?\\s*"exp_reg_apert".*{.*}") {
339                    log_awk(FILENAME";"FNR";4;0405;","D")
340                    n_errores[4]++
341               }
342
343               if (linea_mod ~
"^\\s*{?}?\\s*"exp_reg_apert"(\\s+|{|$)" && linea_mod !~
exp_reg_apert" {" && linea_mod !~
"while\\s*"parent_par"\\s*;") {
344
             n_parent=cuenta_ocurrencias("\\(",linea_mod)-
cuenta_ocurrencias("\\)",linea_mod)
345                    if (n_parent != 0) {
346                         a=0
347                         do {
348                              a++
349
             linea_getl=mi_getline(FILENAME,FNR+a)
350
             n_parent+=cuenta_ocurrencias("\\(",linea_getl)-
cuenta_ocurrencias("\\)",linea_getl)
351                              if (a >= 1000 ) {
352                                   log_awk(" 020 Error en
detección de
la regla 4 para el fichero "FILENAME,"E")
353                              }
354                         } while (n_parent != 0);
355                         num_linea=FNR+a
356                         linea_bucle=linea_getl
```

```
357                     } else {
358                          num_linea=FNR
359                          linea_bucle=linea_mod
360                     }
361
362                     if (linea_bucle ~ /;\s*$/ && linea_bucle ~
    /(^\s*{|{\s*$)/) {
363
          log_awk(FILENAME";"num_linea";4;0403;","D")
364
          log_awk(FILENAME";"num_linea";4;0401;","D")
365                          n_errores[4]++
366                     } else {
367                          if (linea_bucle ~ /{\s*$/) {
368
          log_awk(FILENAME";"num_linea";4;0401;","D")
369                               n_errores[4]++
370                          } else {
371                               i=1
372                               do {
373
          linea_getl=mi_getline(FILENAME,num_linea+i)
374                                    if (linea_getl ~ /^\s*{/) {
375
          log_awk(FILENAME";"num_linea";4;0401;","D")
376
          log_awk(FILENAME";"num_linea+i";4;0403;","D")
377                                         n_errores[4]++
378                                         break
379                                    } else if (linea_getl !~
    /^\s*$/) {
380                                         break
381                                    }
382                                    i++
383                                    if (i >= 1000 ) {
384                                         log_awk(" 020 Error en
    detección de la regla 4 para el fichero "FILENAME,"E")
385                                    }
386                               } while (linea_getl ~ /^\s*$/);
387                          }
388                     }
389               }
390
391               if (linea_mod ~ "^\\s*}?\\s*"exp_reg_cierre &&
    linea_mod !~ "} "exp_reg_cierre) {
392                     if (linea_mod ~ /^\s*}/) {
393                          log_awk(FILENAME";"FNR";4;0402;","D")
394                          n_errores[4]++
395                     } else {
396                          i=1
397                          do {
398
```

```
      linea_getl=mi_getline(FILENAME,FNR-i)
399                         if (linea_getl ~ /(^\s*}|}\s*$)/)
    {
400                               j=0
401                               do {
402                                     j++
403
      linea_getl=mi_getline(FILENAME,FNR-j)
404
405                               if (linea_getl ~
    /(^\s*}|}\s*$)/) {
406                                     k=j
407
      n_llaves=cuenta_ocurrencias("{",linea_getl)-
    cuenta_ocurrencias("}",linea_getl)
408                               do {
409                                     k++
410                                     if (k >=
    1000 ) {
411
      log_awk(" 020 Error en detección de la regla 4 para el
    fichero "FILENAME,"E")
412                                     }
413
      linea_getl=mi_getline(FILENAME,FNR-k)
414                                     if (
    n_llaves !=
    0) {
415
      n_llaves+=cuenta_ocurrencias("{",linea_getl)-
    cuenta_ocurrencias("}",linea_getl)
416                                     }
417                                     if
    (linea_getl ~
    "^\\s*}?{\\s*"exp_reg_apert && n_llaves == 0) {
418                                           do {
419
      k++
420
      linea_getl=mi_getline(FILENAME,FNR-k)
421
      if (k >= 1000 ) {
422
      log_awk(" 020 Error en detección de la regla 4 para el
    fichero "FILENAME,"E")
423                                                 }
424                                           }
    while (linea_getl ~ /^\s*$/);
425                                     }
426                               } while
    (linea_getl !~
    "^\\s*{?}?\\s*"exp_reg_apert || n_llaves != 0);
427                               if (linea_getl ~
```

```
/((else\s*)?if\s*\(.*|do|try)/) {
428
      log_awk(FILENAME";"FNR";4;0402;","D")
429
      log_awk(FILENAME";"FNR-i";4;0404;","D")
430
      n_errores[4]++
431                                                   break
432                                               } else if
(linea_getl !~ "^\\s*{?}?\\s*"exp_reg_apert) {
433                                                   break
434                                               }
435                                         }
436
437                                         if (j >= 1000 ) {
438                                               log_awk(" 020
Error en
detección de la regla 4 para el fichero "FILENAME,"E")
439                                         }
440                                     } while (n_llaves >= 0);
441
442                                     break
443                               } else if (linea_getl !~ /^\s*$/)
{
444                                     break
445                               }
446                               i++
447                               if (i >= 1000 ) {
448                                     log_awk(" 020 Error en
detección de la regla 4 para el fichero "FILENAME,"E")
449                               }
450                         } while (1);
451                   }
452             }
453       }
454
455       ##################################################
##########
456       #                              REGLA 5
               #
457       ##################################################
##########
458       if (regla[5] == 1) {
459             if ( (linea_mod ~
/\s*(class|namespace|struct|enum)\s+/ || linea_mod ~
tipos_func"[^\\(]+\\(.*\\)") && linea_mod ~ /{\s*$/) {
460                   log_awk(FILENAME";"FNR";5;0501;","D")
461                   n_errores[5]++
462             }
463       }
464
465       ##################################################
##########
```

```
466       #                                REGLA 6
                #
467       ##################################################################
          ############
468       if (regla[6] == 1) {
469             if (linea_mod ~
          tipos_var"[^,;=\\(]+(=[^,;=]+)?,([^,;=\\(]+(=[^,;=]+)?,?)+\\
          s*;\\s*$" && linea_mod !~ /\(/ && linea_mod !~ /\)/ && $0 !~
          /\/\/.*,/) {
470                   linea_aux=linea_mod
471                   sub(/{.*}/,"{}", linea_aux)
472                   if (linea_aux ~
          tipos_var"[^,;=\\(]+(=[^,;=]+)?,([^,;=\\(]+(=[^,;=]+)?,?)+\\
          s*;\\s*$") {
473                         log_awk(FILENAME";"FNR";6;0601;","D")
474                         n_errores[6]++
475                   }
476             } else if (linea_mod ~
          tipos_var"([^,;=\\(]+(=[^,;=]+)?,)+[^,;=\\(]+(=[^,;=]+)?(,|[
          ^,;=\\)])\\s*$" && linea_mod !~ /\(/ && linea_mod !~ /\)/ &&
          $0 !~ /\/\/.*,/) {
477                   linea_aux=linea_mod
478                   flag_R6=cuenta_ocurrencias("{",linea_mod)-
          cuenta_ocurrencias("}",linea_mod)
479                   if (flag_R6=0) {
480                         sub(/{.*}/,"{}", linea_aux)
481                   } else {
482                         sub(/{.*/,"{", linea_aux)
483                   }
484                   if (linea_aux ~
          tipos_var"([^,;=\\(]+(=[^,;=]+)?,)+[^,;=\\(]+(=[^,;=]+)?,?\\
          s*$") {
485                         log_awk(FILENAME";"FNR";6;0601;","D")
486                         i=1
487                         do {
488
          linea_getl=mi_getline(FILENAME,FNR+i)
489                               if (flag_R6 != 0) {
490
          flag_R6+=cuenta_ocurrencias("{",linea_getl)-
          cuenta_ocurrencias("}",linea_getl)
491                                     if (flag_R6 == 0) {
492                                           sub(/.*}/,"}",
          linea_getl)
493                                     } else {
494                                           sub(/.*/,"",
          linea_getl)
495                                     }
496                               }
497
498                               if (linea_getl !~ /^\s*,/) {
499
          log_awk(FILENAME";"FNR+i";6;0603;","D")
```

```
500                               }
501
502
      log_awk(FILENAME";"FNR+i";6;0602;","D")
503
504                               if (linea_getl !~ /;\s*$/ &&
linea_getl !~ /,\s*$/) {
505
      log_awk(FILENAME";"FNR+i";6;0604;","D")
506                               }
507
508                               if (linea_getl ~ /;\s*$/) {
509                                     break
510                               }
511                               i++
512                               if (i >= 1000 ) {
513                                     log_awk(" 020 Error en
detección de la regla 6 para el fichero "FILENAME,"E")
514                               }
515                         } while (linea_getl !~ /;\s*$/);
516                         n_errores[6]++
517                   }
518             }
519       }
520
521       ############################################################
############
522       #                                 REGLA 7
                #
523       ############################################################
############
524       if (regla[7] == 1) {
525             if ($0 ~ /\/\/+[^\/]+/ && $0 !~ /\/\/+ [^\s]+/) {
526                   log_awk(FILENAME";"FNR";7;0701;","D")
527                   n_errores[7]++
528             }
529             if ($0 ~ /([^\/]|^)\/\*.*/ && $0 !~ /\*\//) {
530                   flag_R7=1
531                   if ($0 ~ /\/(\*)+\s*$/) {
532                         log_awk(FILENAME";"FNR";7;0702;","D")
533                   } else {
534                         log_awk(FILENAME";"FNR";7;0703;","D")
535                   }
536                   n_errores[7]++
537             } else if ($0 ~ /.*\*\// && $0 !~ /\/\*/) {
538                   flag_R7=0
539                   if ($0 ~ /^\s*(\*)+\//) {
540                         log_awk(FILENAME";"FNR";7;0704;","D")
541                   } else {
542                         log_awk(FILENAME";"FNR";7;0705;","D")
543                   }
544             } else if (flag_R7) {
545                   log_awk(FILENAME";"FNR";7;0706;","D")
```

```
546             }
547       }
548
549       ##########################################################
          ############
550       #                                REGLA 8
                  #
551       ##########################################################
          ############
552       if (regla[8] == 1) {
553             if (linea_mod ~ /^.*((\|\||&&).*)+$/ && linea_mod
          !~ /^.*(\s(\|\||&&)\s.*)+$/) {
554                   log_awk(FILENAME";"FNR";8;0801;","D")
555                   n_errores[8]++
556             }
557       }
558
559       ##########################################################
          ############
560       #                           REGLA REPORTE 1
                  #
561       ##########################################################
          ############
562       if (regla[9] == 1) {
563             if (linea_mod ~
          tipos_var"([^,;=\\(]+(=[^,;=]+)?,?)+\\s*;\\s*$" && linea_mod
          !~ tipos_var"([^,;=\\(]+=[^,;=]+,?)+\\s*;\\s*$" && linea_mod
          !~ /\(/ && linea_mod !~ /\)/ && $0 !~ /\/\/.*,/) {
564                   linea_aux=linea_mod
565                   sub(/{.*}/,"{}", linea_aux)
566                   if (linea_aux ~
          tipos_var"([^,;=\\(]+(=[^,;=]+)?,?)+\\s*;\\s*$" && linea_aux
          !~ tipos_var"([^,;=\\(]+=[^,;=]+,?)+\\s*;\\s*$") {
567                         log_awk(FILENAME";"FNR";9;1;","R")
568                         n_errores[9]++
569                   }
570             } else if (linea_mod ~
          tipos_var"([^,;=\\(]+(=[^,;=]+)?,)+[^,;=\\(]+(=[^,;=]+)?,?\\
          s*$" && linea_mod !~ /\(/ && linea_mod !~ /\)/ && $0 !~
          /\/\/.*,/) {
571                   linea_aux=linea_mod
572                   flag_R8=cuenta_ocurrencias("{",linea_mod)-
          cuenta_ocurrencias("}",linea_mod)
573                   if (flag_R8=0) {
574                         sub(/{.*}/,"{}", linea_aux)
575                   } else {
576                         sub(/{.*/,"{", linea_aux)
577                   }
578                   if (linea_aux ~
          tipos_var"([^,;=\\(]+(=[^,;=]+)?,)+[^,;=\\(]+(=[^,;=]+)?,?\\
          s*$") {
579                         if (linea_aux !~
          tipos_var"([^,;=\\(]+=[^,;=]+,)+[^,;=\\(]+=[^,;=]+,?\\s*$")
```

```
{
580
      log_awk(FILENAME";"FNR";9;1;","R")
581                         n_errores[9]++
582                   }
583                   i=1
584                   do {
585
      linea_getl=mi_getline(FILENAME,FNR+i)
586                         if (flag_R8 != 0) {
587
      flag_R8+=cuenta_ocurrencias("{",linea_getl)-
cuenta_ocurrencias("}",linea_getl)
588                               if (flag_R8 == 0) {
589                                     sub(/.*}/,"}",
linea_getl)
590                               } else {
591                                     sub(/.*/,"",
linea_getl)
592                               }
593                         }
594                         if (linea_getl !~
/^\s*,?([^,;=]+=[^,;=]+,)*[^,;=]+=[^,;=]+,?\s*$/) {
595
      log_awk(FILENAME";"FNR+i";9;1;","R")
596                               n_errores[9]++
597                         }
598                         i++
599                         if (i >= 1000 ) {
600                               log_awk(" 020 Error en
detección de la regla reporte 1 para el fichero
"FILENAME,"E")
601                         }
602                   } while (linea_getl !~ /;\s*$/);
603             }
604       }
605 }
606
607 ##########################################################
############
608 #                         REGLA REPORTE 2
          #
609 ##########################################################
############
610 if (regla[10] == 1) {
611       if (length($0) > long_max) {
612             log_awk(FILENAME";"FNR";10;2;","R")
613             n_errores[10]++
614       }
615 }
616
617 ##########################################################
############
```

```
618         #                              REGLA REPORTE 3
                  #
619         ##########################################################
    ############
620         if (regla[11] == 1) {
621               if (linea_mod ~ tipos_func"[^\\(]+\\(.*\\)") {
622                     if (linea_mod ~ /\)\s*(const)?\s*{\s*$/) {
623                           if (linea_anterior !~ /(\/\/|\*\/)/) {
624
          log_awk(FILENAME";"FNR";11;3;","R")
625                                       n_errores[11]++
626                           }
627                     }
628               }
629         }
630
631
632         if ($0 !~ /^\s*$/) {
633               linea_anterior=$0
634         }
635   }
636   END {
637         log_awk("Estadísticas de detección","TI")
638         for (i=1; i <= n_reglas; i++) {
639               if (regla[i] == 1) {
640                     if (i <= n_reglas-num_reg_rep) {
641                           log_awk("N° errores regla " i ": "
    n_errores[i],"TI")
642                     } else {
643                           log_awk("N° reportes regla " i ": "
    n_errores[i],"TI")
644                     }
645                     n_errores_totales+=n_errores[i]
646               }
647         }
648         log_awk("N° errores totales: " n_errores_totales,"TI")
649
650   }' $WORKSHOP
651
652   err_detector=$?
653   if [[ $err_detector == 0 ]]
654   then
655         log "DETECTOR FINALIZÓ SIN INCIDENCIAS" "TI"
656   else
657         log "016 DETECTOR NO FINALIZÓ CORRECTAMENTE" "E"
658   fi
```

corrector_TC.sh

```
1  ##############################################################
   ##########
2  #                        Inicio de corrector_TC.sh
          #
3  ##############################################################
   ##########
4
5  export RUTA_CORREGIDOS=""
6
7  crea_dir_correg
8
9  check_informe_det
10 ret_corrector=$?
11
12 if [[ $ret_corrector == 1 ]]
13 then
14       log "INICIALIZANDO CORRECTOR" "TI"
15 elif [[ $ret_corrector == 0 ]]
16 then
17       log "ABANDONANDO CORRECTOR" "TI"
18       return
19 else
20       log "017 Error de retorno de la función
   check_informe_det" "E"
21 fi
22
23 awk -b -v informe_det=$INFORME_DET -v fich_log=$LOG -v
   ruta_corregidos=$RUTA_CORREGIDOS -v
   ruta_ficheros=$RUTA_FICHEROS -v regla1="${regla[1]}" '
24 ##############################################################
   ##########
25 #               Nombre función: genera_array_arreglos
          #
26 #     Descripción: genera un array con las expresiones de
   matcheo    #
27 #                        y sustitución de cada subregla
          #
28 ##############################################################
   ##########
29 function genera_array_arreglos()
30 {
31       ##########################################################
   ############
32       #                 Expresiones regulares constantes
             #
33       ##########################################################
   ############
34
35       # Encaja con paréntesis que tengan dentro de ellos
   paréntesis emparejados
36       parent_par="\\s*\\([^\\(\\)]*(\\([^\\(\\)]*\\)[^\\(\\)
   ]*)*\\)"
```

```
37        parent_par="\\s*\\([^\\)]*(\\([^\\(\\)]*\\)[^\\(]*)*\\
    )"
38
39        tipos_var="^\\s*(const\\s+|signed\\s+|unsigned\\s+|cha
    r\\s+|short\\s+|long\\s+|int\\s+|float\\s+|double\\s+|auto\\
    s+|bool\\s+)"
40
41        ##########################################################
    ############
42        #                                REGLA 1
                #
43        ##########################################################
    ############
44        arreglos["0101"][0]["regexp"]="\\s*({)"
45        arreglos["0101"][0]["replac"]="\n\\1"
46        arreglos["0101"][1]["regexp"]="^\n"
47        arreglos["0101"][1]["replac"]=""
48
49        arreglos["0102"][0]["regexp"]="\\s*(})"
50        arreglos["0102"][0]["replac"]="\n\\1"
51        arreglos["0102"][1]["regexp"]="^\n"
52        arreglos["0102"][1]["replac"]=""
53
54        arreglos["0103"][0]["regexp"]="^\\s*"
55        arreglos["0103"][0]["replac"]=""
56
57        ##########################################################
    ############
58        #                                REGLA 2
                #
59        ##########################################################
    ############
60        arreglos["0201"][0]["regexp"]="^\\s*(.*)"
61        arreglos["0201"][0]["replac"]="\\1"
62        arreglos["0201"][1]["regexp"]="\\t"
63        arreglos["0201"][1]["replac"]="    "
64
65        ##########################################################
    ############
66        #                                REGLA 3
                #
67        ##########################################################
    ############
68        arreglos["0301"][0]["regexp"]="^\\s*"
69        arreglos["0301"][0]["replac"]=""
70        arreglos["0301"][1]["regexp"]="(\\)|else|do)(\\s*(\\/\
    \/.*|\\/\\*.*)?)$"
71        arreglos["0301"][1]["replac"]="\\1 {\\2"
72
73
74        arreglos["0302"][0]["regexp"]="^\\s*(.*)"
75        arreglos["0302"][0]["replac"]="\\1\n}"
76
```

```
77        arreglos["0303"][0]["regexp"]="^\\s*(else|while)"
78        arreglos["0303"][0]["replac"]="} \\1"
79
80        arreglos["0304"][0]["regexp"]="\\s*(.*)((if|for|while)
"parent_par"|else(\\s*if"parent_par")?|\\sdo)\\s*"
81        arreglos["0304"][0]["replac"]="\\1\\2 {\n"
82
83        arreglos["0305"][0]["regexp"]="^\\s*"
84        arreglos["0305"][0]["replac"]=""
85        arreglos["0305"][1]["regexp"]="\\/\\/(.*)"
86        arreglos["0305"][1]["replac"]="\\/\\*\\1\\*\\/"
87        arreglos["0305"][2]["regexp"]="(.*)\\s*(\\/\\/|\\/\\*|
)"
88        arreglos["0305"][2]["replac"]="\\1_ELIM-RETORNO_\\2"
89
90        ##########################################################
############
91        #                                REGLA 4
              #
92        ##########################################################
############
93        arreglos["0401"][1]["regexp"]="^\\s*"
94        arreglos["0401"][1]["replac"]=""
95        arreglos["0401"][0]["regexp"]="(\\)|else|do|try|case.*
:|default:)(\\s*){?(\\s*(\\/\\/.*|\\/\\*.*)?)$"
96        arreglos["0401"][0]["replac"]="\\1 {\\2\\3"
97
98
99        arreglos["0402"][0]["regexp"]="^\\s*}?\\s*(catch"paren
t_par"|while"parent_par"\\s*;|else(\\s*if"parent_par")?)"
100       arreglos["0402"][0]["replac"]="} \\1"
101
102       arreglos["0403"][0]["regexp"]="^\\s*"
103       arreglos["0403"][0]["replac"]=""
104       arreglos["0403"][1]["regexp"]="^\\s*{\\s*"
105       arreglos["0403"][1]["replac"]=""
106       arreglos["0403"][2]["regexp"]="^\\s*$"
107       arreglos["0403"][2]["replac"]="_NO-IMPRIMIR_"
108
109       arreglos["0404"][0]["regexp"]="^\\s*"
110       arreglos["0404"][0]["replac"]=""
111       arreglos["0404"][1]["regexp"]="\\s*}\\s*$"
112       arreglos["0404"][1]["replac"]=""
113       arreglos["0404"][2]["regexp"]="^\\s*$"
114       arreglos["0404"][2]["replac"]="_NO-IMPRIMIR_"
115
116       arreglos["0405"][0]["regexp"]="^\\s*(}\\s*)?((if|for|w
hile|catch|switch)"parent_par"|else(\\s*if"parent_par")?|do|
try|case.*:|default:)\\s*{\\s*"
117       arreglos["0405"][0]["replac"]="\\1\\2 {\n"
118       arreglos["0405"][1]["regexp"]=";\\s*}"
119       arreglos["0405"][1]["replac"]=";\n}"
120
```

```
121       ##########################################################
          ###########
122       #                                REGLA 5
                #
123       ##########################################################
          ############
124       arreglos["0501"][0]["regexp"]="^\\s*"
125       arreglos["0501"][0]["replac"]=""
126       arreglos["0501"][1]["regexp"]="\\s*{"
127       arreglos["0501"][1]["replac"]="\n{"
128       arreglos["0501"][2]["regexp"]="{\\s*(.+)"
129       arreglos["0501"][2]["replac"]="{\n\\1"
130
131       ##########################################################
          ############
132       #                                REGLA 6
                #
133       ##########################################################
          ############
134       arreglos["0601"][0]["regexp"]="^\\s*"
135       arreglos["0601"][0]["replac"]=""
136       arreglos["0601"][1]["regexp"]=",\\s*$"
137       arreglos["0601"][1]["replac"]=";"
138       arreglos["0601"][2]["regexp"]="\\s*,\\s*"
139       arreglos["0601"][2]["replac"]=";\n"
140       arreglos["0601"][3]["regexp"]="^\\s*;\n\\s*"
141       arreglos["0601"][3]["replac"]=""
142
143       arreglos["0602"][0]["regexp"]="^\\s*"
144       arreglos["0602"][0]["replac"]=""
145       arreglos["0602"][1]["regexp"]=",\\s*$"
146       arreglos["0602"][1]["replac"]=";"
147       arreglos["0602"][2]["regexp"]="\\s*,\\s*"
148       arreglos["0602"][2]["replac"]=";\n"
149       arreglos["0602"][3]["regexp"]="^\\s*;\n\\s*"
150       arreglos["0602"][3]["replac"]=""
151
152       arreglos["0603"][0]["regexp"]="(.*)"
153       arreglos["0603"][0]["replac"]=",\\1"
154
155       arreglos["0604"][0]["regexp"]="(.*)"
156       arreglos["0604"][0]["replac"]="\\1;"
157
158       ##########################################################
          ############
159       #                                REGLA 7
                #
160       ##########################################################
          ############
161       arreglos["0701"][0]["regexp"]="^\\s*"
162       arreglos["0701"][0]["replac"]=""
163       arreglos["0701"][1]["regexp"]="\\/(\\/+)\\s*"
164       arreglos["0701"][1]["replac"]="\\/\\1 "
```

```
165
166         arreglos["0702"][0]["regexp"]="^\\s*(.*)\\/(\\*)+"
167         arreglos["0702"][0]["replac"]="\\1\n\\/\\*"
168
169         arreglos["0703"][0]["regexp"]="^\\s*(.*)\\/(\\*)+\\s*(
    .*)"
170         arreglos["0703"][0]["replac"]="\\1\n\\/\\*\n \\* \\3"
171
172         arreglos["0704"][0]["regexp"]="^\\s*(\\*)+\\/\\s*(.*)"
173         arreglos["0704"][0]["replac"]=" \\*\\/\n\\2"
174
175         arreglos["0705"][0]["regexp"]="(\\*)+\\/"
176         arreglos["0705"][0]["replac"]="\\*\\/"
177         arreglos["0705"][1]["regexp"]="^\\s*(.*)\\s*(\\*)+\\/(
    .*)$"
178         arreglos["0705"][1]["replac"]=" \\* \\1\n \\*\\/\n\\3"
179
180         arreglos["0706"][0]["regexp"]="^\\s*(\\**\\s*)(.*)\\s*
    "
181         arreglos["0706"][0]["replac"]=" \\* \\2"
182
183         ############################################################
    ############
184         #                                REGLA 8
              #
185         ############################################################
    ############
186         arreglos["0801"][0]["regexp"]="^\\s*"
187         arreglos["0801"][0]["replac"]=""
188         arreglos["0801"][1]["regexp"]="\\s*(\\|\\||&&)\\s*"
189         arreglos["0801"][1]["replac"]=" \\1 "
190   }
191   ##############################################################
    ##########
192   #                    Nombre función: log_awk
            #
193   #     Descripción: imprime la cadena que recibe en el
    fichero         #
194   #       correspondiente según el nivel del texto a imprimir
            #
195   #     Argumentos: texto --> cadena a imprimir
            #
196   #                   nivel --> Nivel del texto
            #
197   #                         I --> Información en log
            #
198   #                         TI -> Información en terminal e log
            #
199   ##############################################################
    ##########
200   function log_awk(texto,nivel)
201   {
202         if (nivel == "I") {
```

```
203             # Pinta en log.txt
204             print "[INFO] ---- " texto >> fich_log
205         } else if (nivel == "TI") {
206             # Pinta en log.txt y en el terminal
207             print "[INFO] ---- " texto >> fich_log
208             print "[INFO] ---- " texto
209         } else if (nivel == "E") {
210             # Pinta en log.txt y en el terminal
211             print "[ERROR]" texto >> fich_log
212             print "[ERROR]" texto
213             exit 1
214         }
215     }
216     ##############################################################
        ##########
217     #                     Nombre función: mi_getline
                #
218     #     Descripción: devuelve la cadena correspondiente al
        fichero     #
219     #                         y linea dados
                #
220     #     Argumentos: fichero --> nombre del fichero
                #
221     #                 n_linea --> nº de línea
                #
222     #     Return: línea leída
                #
223     ##############################################################
        ##########
224     function mi_getline(fichero, n_linea)
225     {
226             cont=0
227             do {
228                cont++
229                getline linea_getline < fichero
230             } while (cont<n_linea)
231                close(fichero)
232             return linea_getline
233     }
234
235     ##############################################################
        ##########
236     #                   Nombre función: cuenta_ocurrencias
                #
237     #     Descripción: devuelve el nº de ocurrencias que encajan
        con      #
238     #                         regexp
                #
239     #     Argumentos: regexp --> expresión regular
                #
240     #                   linea  --> línea a tratar
                #
241     #     Return: número de ocurrencias
```

```
          #
242 ##########################################################
    ##########
243 function cuenta_ocurrencias(regexp,linea)
244 {
245       linea_aux=linea
246       for (iterador=0; linea_aux ~ regexp; iterador++) {
247             sub(regexp,"",linea_aux)
248       }
249       return iterador
250 }
251
252 ##########################################################
    ##########
253 #               Nombre función: carga_informe_detector
          #
254 #     Descripción: carga el informe del detector en un array
          #
255 ##########################################################
    ##########
256 function carga_informe_detector()
257 {
258       getline error < informe_det
259       fichero_anterior=""
260       linea_anterior=""
261
262       while (getline error < informe_det) {
263             split(error,err_campos,";")
264             fichero=err_campos[1]
265             linea=err_campos[2]
266             subregla=err_campos[4]
267             contador_arreglos=0
268
269             if (fichero in array_errores && linea in
    array_errores[fichero]) {
270
      contador_arreglos=length(array_errores[fichero][linea]
    )
271             }
272             fichero_anterior=fichero
273             linea_anterior=linea
274
275
      array_errores[fichero][linea][contador_arreglos]=subre
    gla
276       }
277       close(informe_det)
278 }
279 ##########################################################
    ##########
280 #                  Nombre función: aplica_reglas
          #
281 #     Descripción: modifica la línea del fichero aplicando
```

```
la        #
282 #                        subregla correspondiente
          #
283 #     Argumentos: linea  --> línea a tratar
          #
284 #                 fichero  --> fichero al que pertenece la
    linea     #
285 #                 num_linea  --> número de línea de línea
          #
286 ##########################################################
    ##########
287 function aplica_reglas(linea, fichero, num_linea)
288 {
289       linea_fix[1]=linea
290       borra_ident=1
291       no_identar=0
292
293       if(fichero in array_errores && num_linea in
    array_errores[fichero]) {
294             for(num_subreglas in
    array_errores[fichero][num_linea]) {
295                   subregla =
    array_errores[fichero][num_linea][num_subreglas]
296                   if (subregla in arreglos){
297
298                         for(indice_subregla in
    arreglos[subregla]) {
299                               linea_fix[1] =
    gensub(arreglos[subregla][indice_subregla]["regexp"],
    arreglos[subregla][indice_subregla]["replac"], "g",
    linea_fix[1])
300                         }
301                         if (subregla ~ /0103/) {
302                               no_identar=1
303                         }
304                         borra_ident=0
305                   } else {
306                         log_awk("Subregla " subregla " no
    definida en corrector","TI")
307                   }
308                   n_correcciones++
309             }
310             split(linea_fix[1],linea_fix,"\n")
311       }
312
313       for (ind in linea_fix) {
314
315             # Si la regla de identación está activa
316             if (regla1) {
317                   if (no_identar) {
318                         calc_ident(linea_fix[ind],borra_ident)
319                   } else {
320
```

```
        linea_fix[ind]=calc_ident(linea_fix[ind],borra_ident)
321                 }
322             }
323
324             escribe_linea(linea_fix[ind], fichero)
325             delete linea_fix[ind]
326         }
327     }
328     ##############################################################
    ##########
329     #                   Nombre función: calc_ident
            #
330     #      Descripción: calcula la identación que le corresponde
    a línea  #
331     #      Argumentos: linea  --> línea a tratar
            #
332     #                   borra_ident --> si es 1 eliminamos la
    identación   #
333     #                       previa, si es 0, la mantenemos
            #
334     #      Return: línea identada correctamente
            #
335     ##############################################################
    ##########
336     function calc_ident(linea,borra_ident)
337     {
338         ident_actual=ident_sig
339         linea_sig=mi_getline(FILENAME,FNR+1)
340
341         if (FNR == 1) {
342             ident_actual=0
343             ident_sig=0
344             n_llaves_class=0
345             flag_class=0
346             ident_switch=0
347             ident_class=0
348             n_llaves_switch=0
349             n_ocurr=0
350             multilinea_redir=0
351             multilinea_operac=0
352             primera_linea=0
353         }
354
355         linea_mod=elimina_coment(linea)
356         n_abre_llave=cuenta_ocurrencias("{",linea_mod)
357         n_cierra_llave=cuenta_ocurrencias("}",linea_mod)
358
359         if (linea_mod ~ /^\s*switch\s*\(.*\)*\s*({|$)/) {
360             # ident_switch es la ident +1 para poder usar la
    variable también para control
361             ident_switch=1
362             n_llaves_switch+=n_abre_llave-n_cierra_llave
363         } else if (ident_switch) {
```

```
364             n_llaves_switch+=n_abre_llave-n_cierra_llave
365             if (n_llaves_switch == 0) {
366                   ident_switch=0
367                   flag_switch=0
368                   ident_actual-=4
369                   ident_sig-=4
370
371             } else {
372                   if (linea_mod ~ /^\s*case\s+.*:/ ||
    linea_mod ~ /^\s*default\s*:/) {
373                         if ( flag_switch == 0 ) {
374                               flag_switch=1
375                               ident_sig+=4
376                         } else {
377                               ident_actual-=4
378                         }
379                   }
380
381             }
382
383       }
384
385       if ( (linea_mod ~ /(std::)?(cout|cin)\s*(<<|>>|$)/) ||
    multilinea_redir) {
386             if (!multilinea_redir) {
387                   multilinea_redir=1
388                   ident_sig+=8
389             }
390             if (linea_mod ~ /;\s*$/) {
391                   ident_sig-=8
392                   multilinea_redir=0
393             }
394       } else if ((linea_mod ~ /[^=]=[^=]/ || linea_sig  ~
    /^\s*=[^=]/) && ( linea_mod ~ /(-|+|*[^\/]|=|\/[^\*\/])\s*$/
    || linea_sig ~ /^\s*(-|+|*[^\/]|=|\/[^\*\/])/ ) &&
    (linea_mod !~ /{\s*$/ && linea_mod !~ /;\s*$/) ||
    multilinea_operac) {
395             if (!multilinea_operac) {
396                   multilinea_operac=1
397                   ident_sig+=8
398             }
399             if (linea_mod ~ /;\s*$/) {
400                   ident_sig-=8
401                   multilinea_operac=0
402             }
403       } else {
404             if (linea_mod ~ /{([0-9]+,)+}/) {
405                   ident_multilinea(linea_mod,"{","}")
406             } else {
407                   if (linea_mod ~ /{/ && linea_mod !~ /{.*}/)
    {
408                         ident_sig+=4*n_abre_llave
409                   }
```

```
410
411                 if (linea_mod ~ /}/ && linea_mod !~ /{.*}/)
    {
412                       ident_actual-=4*n_cierra_llave
413                       ident_sig-=4*n_cierra_llave
414                 }
415                 ident_multilinea(linea_mod,"\\(","\\)")
416           }
417     }
418
419
420
421     if (linea_mod ~ /^\s*class\s+/ && linea_mod !~
    /;\s*$/) {
422           # ident_class es la ident +1 para poder usar la
    variable también para control
423           ident_class=1
424           n_llaves_class+=n_abre_llave-n_cierra_llave
425     } else if (ident_class) {
426           n_llaves_class+=n_abre_llave-n_cierra_llave
427           if (n_llaves_class == 0) {
428                 ident_actual-=4
429                 ident_sig-=4
430                 ident_class=0
431                 flag_class=0
432           } else {
433                 if (linea_mod ~
    /^\s*(public|private|protected)\s*:/) {
434                       if ( flag_class == 0 ) {
435                             flag_class=1
436                             ident_sig+=4
437                       } else {
438                             ident_actual-=4
439                       }
440                 }
441
442           }
443
444     }
445
446     espacios=""
447     if (ident_actual<0) {
448           log_awk(" 021 Error con la identación del fichero
    "FILENAME" en el corrector","E")
449
450     }
451     for (i=0; i<ident_actual; i++) {
452           espacios=(espacios " ")
453     }
454
455     if (borra_ident == 0) {
456           sub(/^/,espacios,linea)
457     } else {
```

```
458             sub(/^\s*/,espacios,linea)
459       }
460
461       linea_ant=linea
462
463       return linea
464 }
465 ##############################################################
    ##########
466 #                   Nombre función: elimina_coment
              #
467 #      Descripción: elimina los comentarios de línea
              #
468 #      Argumentos: linea  --> línea a tratar
              #
469 #      Return: línea modificada, ahora sin comentarios
              #
470 ##############################################################
    ##########
471 function elimina_coment(linea)
472 {
473       if (linea ~ /\".+\"/) {
474             gsub(/\"[^\"]+\"/,"",linea)
475       }
476
477       if (linea ~ /\/\//) {
478             sub(/\s*\/\/.*/,"",linea)
479       }
480
481
482       if (linea ~ /\/\*/ || coment_rang) {
483             if (linea ~ /\/\*/ && linea ~ /\*\//) {
484                   sub(/\/\*.*\*\//,"",linea)
485                   coment_rang=0
486             } else if (linea ~ /\/\*/) {
487                   sub(/\/\*.*/,"",linea)
488                   coment_rang=1
489             } else if (linea ~ /\*\//) {
490                   sub(/.*\*\//,"",linea)
491                   coment_rang=0
492             } else {
493                   sub(/^.*$/,"",linea)
494             }
495       }
496
497       return linea
498 }
499 ##############################################################
    ##########
500 #                   Nombre función: ident_multilinea
              #
501 #      Descripción: calcula la identación de una sentencia en
     varias  #
```

```
#                    líneas                                    #
#     Argumentos: linea  --> línea a tratar
          #
#                  exp_apert  --> exp regular del carácter
apertura   #
#                  exp_cierre --> exp regular del carácter
cierre     #
##############################################################
##########
function ident_multilinea(linea,exp_apert,exp_cierre)
{
      dif_ocurr=cuenta_ocurrencias(exp_apert,linea)-
cuenta_ocurrencias(exp_cierre,linea)
      n_ocurr=n_ocurr+dif_ocurr

      if (linea ~ /exp_apert/ || linea ~ /exp_cierre/ ||
n_ocurr ||
primera_linea ) {
            if (n_ocurr) {
                  if (primera_linea == 0 ) {
                        primera_linea++
                        ident_sig+=8
                  }
            } else if (dif_ocurr) {
                  ident_sig-=8
                  primera_linea=0
            }
            if (linea ~ /[{;]\s*$/) {
                  if (primera_linea==0) {
                        primera_linea++
                        ident_sig+=8
                  }
                  if (linea ~ /[{;]/ && n_ocurr == 0) {
                        if (linea ~/^\s*{\s*$/) {
                              ident_actual-=8
                        }
                        ident_sig-=8
                        primera_linea=0
                  }
            }
      }
}
##############################################################
##########
#             Nombre función: escribe_linea                  #
#     Descripción: escribe la línea en el fichero
correspondiente    #
#     Argumentos: linea  --> línea a escribir
          #
#                  fichero  --> fichero en el que escribir
          #
#                  identacion --> número de espacios de
identación    #
```

```
543  ##############################################################
     ##########
544  function escribe_linea(linea, fichero)
545  {
546       sub(ruta_ficheros"\\/","",fichero)
547       fichero=ruta_corregidos"/"fichero
548
549       if (linea !~ /^\s*_NO-IMPRIMIR_$/) {
550            if (linea ~ /_ELIM-RETORNO_/) {
551                 sub(/\s*_ELIM-RETORNO_/," ",linea)
552                 printf "%s" , linea >> fichero
553            } else {
554                 print linea >> fichero
555            }
556       }
557  }
558
559  BEGIN {
560       ident_actual=0
561       ident_sig=0
562       n_llaves_class=0
563       flag_class=0
564       ident_switch=0
565       ident_class=0
566       n_llaves_switch=0
567       n_ocurr=0
568       multilinea_redir=0
569       multilinea_operac=0
570       primera_linea=0
571       n_correcciones=0
572
573       carga_informe_detector()
574       genera_array_arreglos()
575
576       or_06=arreglos["0601"][2]["replac"]
577  }{
578       n_matches=0
579       linea_match=$0
580       while (match(linea_match,tipos_var) != 0) {
581            if (n_matches == 0) {
582                 tipo=""
583            }
584            tipo=tipo""gensub(tipos_var".*",
     "\\1",1,linea_match)
585            sub(tipos_var,"",linea_match)
586            n_matches++
587       }
588
589       arreglos["0601"][2]["replac"]=or_06 tipo
590       arreglos["0602"][2]["replac"]=or_06 tipo
591
592       aplica_reglas($0, FILENAME, FNR)
593  }
```

```
594
595   END {
596         log_awk("Se han realizado un total de " n_correcciones
      " correcciones","TI")
597   }' $WORKSHOP
598
599
600   ret_corrector=$?
601   if [[ $ret_corrector == 0 ]]
602   then
603         log "CORRECTOR FINALIZÓ SIN INCIDENCIAS" "TI"
604         if [[ $SOBRESCRIBIR == "SI" ]]
605         then
606               rm -rf $RUTA_FICHEROS
607               log "${RUTA_FICHEROS} eliminado" "I"
608               mv $RUTA_CORREGIDOS $RUTA_FICHEROS
609               log "${RUTA_CORREGIDOS} es ahora ${RUTA_FICHEROS}"
      "I"
610               RUTA_CORREGIDOS=$RUTA_FICHEROS
611         fi
612   else
613         log "018 CORRECTOR NO FINALIZÓ CORRECTAMENTE" "E"
614   fi
```