# Gestión de revistas:
# justificación de las operaciones

6 de diciembre de 2013

## ● La operación eliminar_revista (clase Biblioteca)

Esta operación tiene como consecuencia que toda información relativa a la revista cuyo título nos llega por parámetro explícito se elimina de la biblioteca.

## ● Implementación

```
void Biblioteca::eliminar_revista(string nombre) {

// Pre: Existe una revista en la biblioteca con título = nombre
// Post: La revista con dicho título desaparece de la biblioteca

    int pos, cal;

    bool trobat;

    busca_biblio(nombre, cal, trobat, pos);

    list<Revista>::iterator it = biblio[cal-1].begin();

/* Inv1: La suma de la posición a la que apunta el iterador más pos es igual a la posición de la
         revista de biblio que queremos borrar (0 <=  pos <= biblio[cal-1].size() - 1) */

    while (pos > 0) {

        ++it;

        --pos;

    }

    biblio[cal-1].erase(it);

    pos = busca_biblio2(nombre, cal);

    list< pair<string, string> >::iterator it2 = biblio2[cal-1].begin();

/* Inv2: La suma de la posición a la que apunta el iterador más pos es igual a la posición de la
         revista de biblio2 que queremos borrar (0 <=  pos <= biblio2[cal-1].size() - 1) */
```

```
    while (pos > 0) {
        ++it2;
        --pos;
    }
    biblio2[cal-1].erase(it2);
}
```

- Justificación

Primero de todo, hemos de garantizar que las llamadas a busca_biblio y busca_biblio2 son correctas (es decir, que cumplen la precondición de ambas funciones) para luego suponer que hacen lo que esperamos de ellas y así poder justificar de manera precisa eliminar_revista. Una vez justificada esta función, procederemos a demostrar la corrección de las otras dos por separado. La primera de ellas, busca_biblio, tiene como precondición cierto, mientras que busca_biblio2 tiene como precondición que exista una revista en la biblioteca con título = *nombre* y cuyo índice de calidad sea igual a *cal*. La primera parte de la precondición de busca_biblio2 se cumple por el mero hecho de coincidir con la precondición de eliminar_revista y la segunda parte se satisface por la postcondición de busca_biblio (página 3 del pdf). Supondremos que ésta última se cumple y pasamos a justificar el primer bucle:

✓ *Inicializaciones*. Antes de entrar en el while, se cumple el invariante porque el iterador apunta a la primera revista de la lista (posición 0) y, por lo tanto, la suma de la posición a la que apunta el iterador más *pos* es igual a *pos*, que por la postcondición de busca_biblio sabemos que contiene la posición de la revista que queremos borrar.

✓ *Condición de salida*. Saldremos del while cuando *pos* llegue a valer cero. En ese instante, el iterador apuntará a la revista que deseamos borrar porque la suma de la posición a la que apunta el iterador más *pos* será igual a la posición a la que apunta el iterador, y el invariante nos dice que en esa posición está la revista que queremos eliminar de la biblioteca.

✓ *Cuerpo del bucle*. En cada iteración del bucle, incrementamos y decrementamos en una unidad el iterador y el valor de *pos*, respectivamente. Por lo tanto, la suma de la posición a la que apunta el iterador más *pos* será un valor constante que, según el invariante, será la posición de la revista que queremos borrar.

✓ *Finalización*. En cada iteración, la distancia entre *pos* y 0 se va haciendo más pequeña a causa del *--pos* del final del bucle.

Justificación de las instrucciones posteriores al primer bucle
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

✓ Para que la revista cuyo título nos llega por parámetro explícito desaparezca de la biblioteca (postcondición), hemos de eliminarla tanto de biblio como de biblio2. Como hemos visto antes, justo después de salir del bucle, el iterador apuntará a la revista de biblio que deseamos borrar, por

lo que tan solo tendremos que usar la operación de la clase lista *erase* con dicho iterador como parámetro para eliminarla de biblio. La llamada a *erase* será correcta, dado que su precondición es que el iterador apunte a un elemento existente en el parámetro implícito y nos hemos asegurado de que eso se cumpla.

✓ A continuación se produce la llamada a busca_biblio2 que hemos comentado anteriormente. Ésta devolverá un entero que guardaremos en pos y que nos indicará la posición de la revista de biblio2 que queremos borrar. Finalmente, volvemos a realizar el primer bucle pero esta vez con un iterador correspondiente a una lista de pair de strings (la justificación de este segundo bucle es análoga a la del primero) y ya solo nos queda hacer un *erase* con dicho iterador como parámetro para que no quede ningún rastro de la revista en la biblioteca y de esta forma cumpliremos la postcondición.

## ● La operación auxiliar busca_biblio (clase Biblioteca)

La operación busca_biblio busca una revista en biblio cuyo título sea igual al string que le pasamos por parámetro explícito.

### ● Implementación

```
void Biblioteca::busca_biblio(string nombre, int &cal, bool &trobat, int &pos) {

// Pre: Cierto
// Post: trobat nos dice si existe la revista en biblio y, en caso afirmativo, pos indica la posición
        donde está la revista en la lista de revistas y cal representa su índice de calidad

   pos = 0; // Para cumplir el invariante antes de entrar en el bucle (no sería necesario)

   trobat = false;

   int i = 0;

/* Inv1: 0 <= i <= biblio.size(); trobat = "la revista con título = nombre es un elemento de
         biblio[0 .. i – 1]"; cal solo tendrá un valor relevante si trobat = true, ya que para que
         indique el nivel de calidad de la revista que buscamos primero habrá que encontrarla
         (si trobat = true, 1 <= cal <= biblio.size()); pos indica la posición donde nos encontramos
         en la lista (0 <= pos <= biblio[i].size()) */

   while (i < biblio.size() and not trobat) {

      list<Revista>::iterator it = biblio[i].begin();

      pos = 0;

/* Inv2: 0 <= i <= biblio.size(); trobat = … ; cal solo tendrá... ; pos indica... ; biblio[i].begin() <= it
         <= biblio[i].end() */
```

```
      while (it != biblio[i].end() and not trobat) {
         if ((*it).consultar_nombre() == nombre) {
            trobat = true;
            cal = i+1;
         }
         else {
            ++it;
            ++pos;
         }
      }
      ++i;
   }
}
```

- Justificación

Justificación del segundo bucle
- - - - - - - - - - - - - - - - - - - - - - -

✓ *Inicializaciones*. Si estamos en disposición de entrar en el segundo bucle, quiere decir que hemos entrado en el primero, por lo que *i* será menor que biblio.size() y *trobat* será false, es decir, que la revista que buscamos no está en biblio[0 .. *i* – 1]. Si *trobat* es igual a false, el valor de *cal* es irrelevante. Por otro lado, inicializamos *pos* a 0 porque en dicha posición está la primera revista de la lista que tenemos que comprobar si es la que buscamos. Asimismo, inicializamos el iterador para que apunte al primer elemento de la lista, cumpliendo así todo el invariante.

✓ *Condición de salida*. Saldremos del bucle si:

- *it* alcanza el valor de biblio[i].end(). Si esto llega a suceder, querrá decir que hemos llegado al final de la lista sin encontrar la revista. En ese caso, habrá que incrementar la *i* para volver a entrar en el primer bucle y verificar si la revista está en la siguiente lista. Si no entramos en el primer bucle porque al incrementar la *i* vemos que ésta llega a ser igual a biblio.size(), eso significará que hemos recorrido biblio y no hemos encontrado la revista, por lo que habremos cumplido la postcondición, ya que *trobat* será igual a false.

- *trobat* = true. Por el invariante, esto significará que hemos encontrado la revista en biblio[0 .. *i* – 1]. Antes de salir del bucle, después de ejecutar *cal = i+1*, *cal* indicará el nivel

de calidad de la revista encontrada (cada posición de biblio representa una calidad: en la posición 0 tenemos las revistas de calidad 1, en la posición 1 tenemos las de calidad 2, etc.) y *pos* no se incrementará, por lo que indicará la posición de dicha revista en la lista . Incrementaremos la *i*, subiremos al primer bucle, no entraremos en él porque *trobat* será igual a true y de esta manera cumpliremos la postcondición.

✓ *Cuerpo del bucle*. Lo que hacemos en cada iteración del bucle es comprobar si la revista a la que apunta el iterador tiene como título la variable *nombre*. En caso afirmativo, ponemos *trobat* a true y le asignamos a *cal* un valor igual a *i* + 1 por lo motivos que hemos descrito anteriormente y se cumplirá el invariante. En caso negativo, debemos incrementar el iterador y el valor de *pos* para indicar que vamos a mirar la siguiente revista de la lista. En este caso, se cumplirá el invariante porque la condición de entrada al bucle es que *it != biblio[i].end()* y eso implica que al incrementar el iterador como mucho llegará a valer biblio[i].end(). Teniendo en cuenta que *it* y *pos* se incrementan a la vez, que nunca sobrepasaremos biblio[i].end() y que la posición de éste equivaldría a la de biblio[i].size(), deducimos que *pos* no alcanzará nunca un valor superior a biblio[i].size().

✓ *Finalización*. En cada iteración, disminuye la distancia entre *it* y biblio[i].end() a causa del *++it*.

Justificación del primer bucle
- - - - - - - - - - - - - - - - - - - - -

✓ *Inicializaciones*. Antes de entrar en el bucle, no hemos analizado ningún elemento de biblio, por lo que hemos de inicializar la *i* a 0. Para satisfacer el resto del invariante, hemos de inicializar *trobat* a false (no puede existir ningún elemento en un subvector vacío biblio[0 .. -1]), *pos* a 0 (aunque realmente no haría falta porque nada más entrar en el segundo bucle lo ponemos a cero) y a *cal* no podemos darle un valor inicial porque antes de entrar en el while no hemos ni empezado la búsqueda de la revista en cuestión, por lo que no hemos podido encontrarla.

✓ *Condición de salida*. Saldremos del bucle si:

- *i* alcanza el valor de biblio.size(). Si esto ocurre, querrá decir por el invariante que hemos recorrido por completo biblio y que no hemos encontrado la revista con el título que buscábamos. Por lo tanto, *trobat* valdrá false y se cumplirá la postcondición porque dicho booleano nos indicará correctamente que no existe la revista solicitada en biblio.

- *trobat* = true. En ese caso, y de nuevo por el invariante, querrá decir que la revista está en biblio[0 .. *i* – 1] y, por lo tanto, *trobat* valdrá true y se cumplirá la postcondición porque dicho booleano nos indicará correctamente que existe la revista solicitada en biblio y porque los valores de *pos* y *cal* serán correctos por lo que hemos comentado en la justificación del segundo bucle.

✓ *Cuerpo del bucle*. Si entramos en el bucle, querrá decir que *trobat* = false, lo cual significa por el invariante que la revista no está en biblio[0 .. *i* – 1]. Por lo tanto, lo que habrá que comprobar dentro del bucle es si la revista se encuentra en biblio[0 .. *i*]. En caso afirmativo, habrá que poner *trobat* a true, asignarle el valor que corresponda a *cal* y dejar *pos* como estaba, tal y como hemos visto en la justificación del segundo bucle. De esta manera, cuando incrementemos la *i*, se cumplirá el invariante del *trobat* (la revista estará en biblio[0 .. *i* – 1]), no entraremos en el primer bucle y se cumplirá la postcondición. En caso de que la revista no esté en biblio[0 .. *i*], incrementaremos la *i*

para buscar en la siguiente lista de revistas e igualmente se cumplirá el invariante porque *trobat* indicará que la revista no existe en biblio[0 .. *i* – 1].

✓ *Finalización*. En cada iteración, disminuye la distancia entre *i* y biblio.size() a causa del ++*i*.

## ● La operación auxiliar busca_biblio2 (clase Biblioteca)

La operación busca_biblio2 busca en biblio2 una revista cuyos título y calidad sean iguales al string y al entero que le pasamos a la función por parámetro explícito.

### ● Implementación

```
int Biblioteca::busca_biblio2(string nombre, int cal) {

// Pre: En biblio2 existe una revista (los pairs de strings simbolizan revistas, de las
        cuales solo almacenamos su título y su área según el criterio 2) con el título
        y la calidad solicitados
// Post: El resultado es la posición donde se encuentra la revista en la lista de revistas
         de calidad cal

   int pos = 0;

   bool trobat = false;

   list< pair<string, string> >::iterator it = biblio2[cal-1].begin();

/* Inv: biblio2[cal-1].begin() <= it <= biblio2[cal-1].end() - 1; trobat = "la revista con título =
        nombre está entre biblio2[cal-1].begin() y la posición anterior a la que apunta it"; pos indica
        la posición donde nos encontramos en la lista (0 <= pos <= biblio2[cal-1].size() - 1) */

   while (it != biblio2[cal-1].end() and not trobat) {

      if ((*it).second == nombre) trobat = true;

      else {

         ++it;

         ++pos;

      }

   }

   return pos;

}
```

- Justificación

✓ *Inicializaciones*. Antes de entrar en el bucle, hemos de inicializar el iterador para que apunte a la primera revista de la lista, *pos* ha de valer 0 porque en dicha posición está la primera revista que queremos comprobar si es la que buscamos y *trobat* ha de ser false porque no puede existir ninguna revista entre biblio2[cal-1].begin() y la posición anterior a biblio2[cal-1].begin().

✓ *Condición de salida*. Saldremos del bucle si:

- *it* alcanza el valor de biblio2[cal-1].end(). Esto querría decir que hemos recorrido toda la lista y que no hemos encontrado la revista, pero esa situación no puede darse por la precondición, ya que se nos garantiza que la revista está en biblio2.

- *trobat* = true. En ese caso, habremos encontrado la revista y *pos* indicará la posición donde se encuentra la revista en la lista, tal y como se pretende en la postcondición.

✓ *Cuerpo del bucle*. Lo que hacemos en cada iteración del bucle es comprobar si la revista a la que apunta el iterador tiene como título la variable *nombre*. En caso afirmativo, ponemos *trobat* a true, salimos del bucle y *pos* indica la última posición que hemos mirado de la lista, que coincide con la posición que ocupa la revista en la lista. Por lo tanto, se cumplen tanto el invariante como la postcondición. En caso negativo, incrementamos *it* y *pos* para indicar que vamos a mirar la siguiente revista de la lista. También se cumplirá el invariante porque la condición de entrada al bucle es que *it != biblio2[cal-1].end()* y eso implica que al incrementar el iterador como mucho llegará a valer biblio2[cal-1].end(), pero *it* nunca sobrepasará el valor de biblio2[cal-1].end() - 1 porque sabemos por la precondición que la lista contiene la revista que buscamos. Teniendo en cuenta que *it* y *pos* se incrementan a la vez, que nunca llegaremos a biblio2[cal-1].end() y que la posición de éste equivaldría a la de biblio2[cal-1].size(), deducimos que *pos* no alcanzará nunca un valor superior a biblio2[cal-1].size() - 1.

✓ *Finalización*. En cada iteración, disminuye la distancia entre *it* y biblio2[cal-1].end() a causa del *++it*.

## - La operación calcular_area_c2 (clase Jerarquiarea)

Esta operación obtiene el área temática asociada a una revista según el criterio 2.

- Implementación

```
string Jerarquiarea::calcular_area_c2(Revista &r) {

// Pre: La revista r ha sido inicializada (a ésta se le ha asignado un nombre, unas palabras clave
        y una calidad) y la jerarquía también
// Post: El resultado es el área temática que corresponde a la revista r según el criterio 2 (debe ser el
        área más profunda cuyas palabras cubiertas incluyan a todas las apariciones de todas las
        palabras clave de la revista)
```

```
    vector<string > v (r.consultar_num_pal_clave());
    int i = 0;
/* Inv: 0 <= i <= v.size()
    while (i < v.size()) {
        v[i] = r.consultar_pal_clave(i);
        ++i;
    }
    pair<bool, string> p = inmersion_c2(a, v);
    return p.second;
}
```

- Justificación

Justificación del bucle
- - - - - - - - - - - - - - - -

✓ *Inicializaciones*. El objetivo del bucle es almacenar las palabras clave de la revista en un vector para después pasárselo a la función de inmersión. Para este cometido, tendremos que recorrer todas las posiciones del vector y en cada una de ellas guardar la palabra clave que nos devuelve la llamada a *consultar_pal_clave*. Por lo tanto, hemos de inicializar la *i* a 0.

✓ *Condición de salida*. Saldremos del bucle si *i* alcanza el valor de v.size(). Eso significará que hemos recorrido todo el vector y que éste contiene las palabras clave de la revista.

✓ *Cuerpo del bucle*. Si entramos en el bucle, querrá decir que *i* < v.size(), por lo que al incrementar la *i*, ésta nunca sobrepasará el valor de v.size() y no haremos un acceso fuera de rango. Una vez asegurado esto, solo nos queda hacer la llamada pertinente a *consultar_pal_clave*, la precondición de la cual siempre se respetará, puesto que 0 <= *i* <= v.size() = número de palabras clave de r.

✓ *Finalización*. En cada iteración, disminuye la distancia entre *i* y v.size() a causa del *++i*.

Justificación de las instrucciones posteriores al bucle
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

✓ La llamada a inmersion_c2 es correcta porque se satisface la precondición de ésta (página 9 del pdf). Supondremos, pues, que se cumplirá la postcondición de inmersion_c2 (página 9 del pdf). De ser así, p.first será siempre igual a true (se nos garantiza que toda revista tiene al menos una palabra clave y que toda palabra clave de una revista debe aparecer en el árbol) y p.second será el área que tendremos que devolver para cumplir la postcondición.

- La operación inmersion_c2 (clase Jerarquiarea)

Esta operación tiene el mismo objetivo que calcular_area_c2.

- Implementación

```
pair <bool, string> Jerarquiarea::inmersion_c2(Arbre<string> &a, vector<string> &v) {

// Pre: a = A; v no es vacío (una revista tiene asociada una palabra clave como mínimo);
// Post: El pair que devuelve la función indica si en A tenemos una palabra clave y, en ese caso,
        el área asociada a la revista para el árbol A según el criterio 2

   pair<bool, string> p (false, " ");

   if (not a.es_buit()) { // caso base

      string raiz = a.arrel();

      Arbre<string> a1, a2;

      a.fills(a1, a2);  // a1 = hi(A); a2 = hd(A); a  = buit

      if (a1.es_buit() and a2.es_buit()) { // si no tiene hijos y la raíz es palabra
                                           // clave, la raíz es el área que buscamos

         int i = 0;

/* Inv: 0 <= i <= v.size(); p.first = "la raíz es un elemento de v[0..i-1]" */

         while (i < v.size() and not p.first) {

            if (v[i] == raiz) {

               p.first = true;

               p.second = raiz;

            }

            else ++i;

         }

      }

      else {
```

```
            pair<bool, string> p1 = inmersion_c2(a1, v);

//  HI1: p1.first = “en hi(A) tenemos como mínimo una palabra clave”; p1.second = el área asociada
//  a la revista para el árbol hi(A) (si p1.first es false, su valor será " ", lo cual es irrelevante)

            pair<bool, string> p2 = inmersion_c2(a2, v);

//  HI2: p2.first = “en hd(A) tenemos como mínimo una palabra clave”; p2.second = el área asociada
//  a la revista para el árbol hd(A) (si p2.first es false, su valor será " ", lo cual es irrelevante)

            p.first = (p1.first or p2.first);

            if (p.first) {

               if (p1.first and p2.first) p.second = raiz;

               else if (p1.first and not p2.first) p.second = p1.second;

               else if (not p1.first and p2.first) p.second = p2.second;

            }

         }

         a.plantar(raiz, a1, a2);

         return p;

      }

}
```

- Justificación

✓ *Caso directo.* Si *a* no es vacío, tiene hijos vacíos y la raíz es una palabra clave de la revista, el área que buscamos es la raíz, puesto que es el área más alejada de *a* que incluye a todas las apariciones de todas las palabras clave de la revista en *a*. Las llamadas a *es_buit* son correctas porque *es_buit* tiene cierto como precondición. Lo mismo sucede con las llamadas a *arrel* y *fills*, ya que *a* no es vacío por la rama del if en la que nos encontramos y *a1* y *a2* son vacíos al declararlos. El proceso para saber si la raíz es una palabra clave consiste en recorrer *v* y si la raíz coincide con algún elemento de éste, significa que lo es y, por lo tanto, es el área que buscamos. Procedemos a justificar el bucle que realiza este cometido, que es muy similar a los justificados anteriormente.

- *Inicializaciones*. Para recorrer el vector desde el primer elemento, hemos de inicializar la *i* a 0. Para satisfacer el resto del invariante, hemos de inicializar *p.first* a false, ya que no puede existir ningún elemento en un subvector vacío v[0 .. -1]. Esto lo hacemos nada más entrar a la función, al declarar el pair.

- *Condición de salida*. Si salimos del bucle porque *i* alcanza el valor de v.size(), querrá decir que la raíz no es palabra clave y no hemos de asignarle ningún valor a *p.first*. Esta situación

se puede dar mientras actúe la recursividad (cuando lleguemos a una hoja y ésta no sea palabra clave), pero no si entramos por primera vez a la función con un árbol no vacío sin hijos, porque sabemos que todas las palabras clave de la revista deben aparecer en el árbol. Si salimos del bucle porque *p.first* = true, querrá decir que la raíz es palabra clave.

- *Cuerpo del bucle*. Si entramos en el bucle, querrá decir que *i* < v.size(), por lo que al incrementar la *i*, ésta nunca sobrepasará el valor de v.size() y no haremos un acceso fuera de rango. En cuanto a *p.first*, tan solo actualizaremos su valor si *raiz* coincide con el elemento del vector que estamos mirando. En ese caso, hemos de asignarle a *p.second* el valor de *raiz* para cumplir la postcondición.

- *Finalización*. En cada iteración, disminuye la distancia entre *i* y v.size() a causa del *++i*.

✓ *Caso recursivo.* Si el árbol no es vacío y al menos uno de sus dos hijos no es vacío, tenemos que mirar si en alguno de ellos hay una palabra clave. En caso afirmativo, como A incluye a las palabras clave cubiertas por hi(A) y hd(A) y por hipótesis de inducción *p1.first* y *p2.first* indican si en los hijos hay alguna palabra clave, *p.first = (p1.first or p2.first)* = true. A continuación hay que evaluar los tres casos posibles. El primero de ellos es que hayamos encontrado palabras clave en ambos hijos. En ese caso, hemos de devolver la raíz de A en el *p.second* para cubrir todas las apariciones de todas las palabras clave de la revista y de esta forma cumpliremos la postcondición. Los otros dos casos posibles son que solo hayamos encontrado palabras clave en el hijo izquierdo o bien que solo las hayamos encontrado en el derecho. En cualquiera de estos dos casos, hemos de devolver mediante *p.second* el área que nos indique *p1.second* o *p2.second*, dependiendo de en cual de los dos hemos encontrado palabras clave, para cumplir la postcondición. Si *p.first = (p1.first or p2.first)* = false, eso quiere decir que *p1.first = p2.first* = false, que por hipótesis de inducción significa que en los hijos no hay palabras clave y, por lo tanto, cumpliremos la postcondición al devolver *p.first* = false y *p.second* = " ". Por último, antes de retornar el pair, hemos de llamar a *plantar* para reconstruir el árbol por si necesitamos obtener el área de alguna otra revista (la llamada es correcta porque *a* es un árbol vacío desde la llamada a *fills*).

✓ *Finalización.* En cada llamada recursiva, el árbol es cada vez más pequeño.