PRÁCTICA 2 **Estructuras de datos y algoritmos**

Título: Programación de TAD genérico arborescente en C++. **Curso 2018/19**

Objetivo:

- Diseñar una implementación arborescente y en memoria dinámica en C++ del TAD genérico "*colecciónConMarca*" y usarlo para implementar un TAD "*repertorio*" que gestione un grupo de canciones.

Fecha límite de entrega: **19-12-2018 (incluido)**

Descripción detallada:

Se trata de implementar un TAD genérico, particularizarlo y utilizarlo para gestionar un grupo de canciones. Se realizarán las siguientes tareas:

1. Desarrollar una implementación dinámica en C++, utilizando un **árbol binario de búsqueda** (ABB), del TAD genérico **colecciónConMarca** cuya especificación se incluye a continuación:

```
especificación colecciónConMarca
   usa booleanos, naturales
   parámetros formales
    géneros clave, valor
    operaciones
    _<_: clave c1 , clave c2 -> booleano {Devuelve verdad si y solo si c1 es menor que c2}
   fpf
   género colecciónConMarca {Los valores del TAD representan conjuntos de tripletas (clave,valor,
booleano) en los que no se permiten claves repetidas, y que cuentan con las operaciones de un
iterador que permite recorrer los datos de la colecciónConMarca según el orden por clave, de menor
a mayor. El valor booleano de cada tripleta diremos que es la marca asociada a la clave.}
operaciones
   crear: → colecciónConMarca
   {Devuelve una colecciónConMarca vacía, sin elementos (tripletas)}
   añadir: colecciónConMarca d , clave c , valor v → colecciónConMarca
   {Si en d no hay ninguna tripleta con clave c, devuelve una colecciónConMarca igual a la
   resultante de añadir la tripleta (c,v,falso) a d; si en d hay una tripleta (c,v',b), entonces
   devuelve una colecciónConMarca igual a la resultante de sustituir (c,v',b) por (c,v,falso).}
   pertenece?: colecciónConMarca d, clave c → booleano
   {Devuelve verdad si y sólo si en d hay alguna tripleta (c,v,b).}
   parcial obtenerValor: colecciónConMarca d, clave c → valor
   {Devuelve el valor asociado a la clave c en d.
   Parcial: la operación no está definida si c no está en d.}
   parcial obtenerMarca: colecciónConMarca d, clave c → booleano
   {Devuelve la marca asociada a la clave c en d.
   Parcial: la operación no está definida si c no está en d.}
   quitar: colecciónConMarca d, clave c → colecciónConMarca
   {Si c está en d, devuelve una colecciónConMarca igual a la resultante de borrar en d la
   tripleta con clave c; si c no está en d, devuelve una colecciónConMarca igual a d.}
   cardinal: colecciónConMarca d → natural
   {Devuelve el nº de elementos (tripletas) en la colecciónConMarca d.}
   cardinalConMarca: colecciónConMarca d → natural
   {Devuelve el nº de elementos (tripletas) en la colecciónConMarca d cuya marca es igual a
   verdad.}
   cambiarMarca: colecciónConMarca d, clave c → colecciónConMarca
   {Si pertenece?(d,c), existe una tripleta (c,v,m) en la colecciónConMarca d, y devuelve la
   colecciónConMarca resultante de sustituir (c,v,m) por (c,v,(not m)). Si not pertenece?(d,c),
   devuelve una colecciónConMarca igual a d.}
   esVacío?: colecciónConMarca d → booleano
   {Devuelve verdad si y sólo si d no tiene elementos.}

   {Operaciones del iterador interno, para recorrer los datos de la colecciónConMarca en orden por
   clave de menor a mayor:}
   iniciarIterador: colecciónConMarca d → colecciónConMarca
   {Prepara el iterador y su cursor para que el siguiente elemento (tripleta) a visitar sea el
   primero de la colecciónConMarca d (situación de no haber visitado ningún elemento).}
   existeSiguiente?: colecciónConMarca d → booleano
   {Devuelve falso si ya se ha visitado el último elemento. Devuelve verdad en caso contrario.}
   parcial siguienteClave: colecciónConMarca d → clave
   {Devuelve la clave del siguiente elemento (tripleta) de d.
   Parcial: la operación no está definida si not existeSiguiente?(d).}
   parcial siguienteValor: colecciónConMarca d → valor
   {Devuelve el valor del siguiente elemento (tripleta) de d.
   Parcial: la operación no está definida si not existeSiguiente?(d).}
   parcial siguienteMarca: colecciónConMarca d → booleano
   {Devuelve la marca del siguiente elemento (tripleta) de d.
   Parcial: la operación no está definida si not existeSiguiente?(d).}
   parcial avanza: colecciónConMarca d → colecciónConMarca
   {Devuelve la colecciónConMarca resultante de avanzar el cursor en d.
   Parcial: la operación no está definida si not existeSiguiente?(d).}
fespec
```

2. Utilizar la implementación del TAD *colecciónConMarca* del punto anterior, y la implementación del TAD *canción* desarrollado en la práctica anterior, para implementar en C++ el TAD *repertorio* que se especifica a continuación. El objetivo del TAD *repertorio* es gestionar un grupo de canciones. El número de canciones que se pueden almacenar en un repertorio no estará limitado.
   La especificación del TAD *repertorio* que vamos a considerar es:

espec repertorio
usa booleanos, naturales, cadenas, canciones, colecciónConMarca
género repertorio *{Los valores del TAD representan un repertorio de canciones con las siguientes características: una cadena de caracteres para representar el título del repertorio, y un conjunto de canciones, cada una de ellas identificada por una clave (cadena) y marcada con un valor booleano para indicar si la canción ha sido escuchada o no.}*
operaciones
crear: cadena tit → repertorio
{Devuelve un repertorio vacío (sin canciones), y título tit}
tituloRepertorio: repertorio r → cadena
{Devuelve el título del repertorio r}
existeCanción?: repertorio r, cadena k → booleano
{Devuelve cierto si y solo si en el repertorio r existe una canción identificada por clave k}
añadir: repertorio r , cadena k , canción c → repertorio
{Si not existeCanción?(r,k), devuelve el repertorio resultante de añadir en r una canción no escuchada c e identificada por la clave k. Si existeCanción?(r,k), en r existe una canción c1 identificada por la clave k, entonces devuelve el repertorio resultante de sustituir en r la canción c1 por la canción c, conservando la misma clave, pero marcada como no escuchada.}
parcial obtenerCanción: repertorio r, cadena k → canción
{Si existeCanción?(r,k), devuelve la canción identificada por la clave k en r.
Parcial: la operación no está definida si not existeCanción?(r,k).}
parcial escuchada?: repertorio r, cadena k → booleano
{Si existeCanción?(r,k), devuelve verdad si y solo si la canción identificada por la clave k en r ha sido escuchada.
Parcial: la operación no está definida si not existeCanción?(r,k).}
modificarEscuchada: repertorio r, cadena k, booleano m → repertorio
{Si existeCanción?(r,k), devuelve el repertorio resultante de marcar la canción identificada por la clave k con m. *Si not existeCanción?(r,k), devuelve un repertorio igual a r.}*
quitarCanción: repertorio r, cadena k → repertorio
{Si existeCanción?(r,k), devuelve el repertorio resultante de eliminar de r la canción identificada por k. En caso contrario, devuelve un repertorio igual a r.}
totalCanciones: repertorio r → natural
{Devuelve el número total de canciones en el repertorio r.}
totalCancionesEscuchadas: repertorio r → natural
{Devuelve el número total de canciones escuchadas en el repertorio r.}

listarRepertorio: repertorio r → cadena
{Devuelve una cadena que contiene la representación como cadena de caracteres de toda la información sobre todas las canciones del repertorio. La cadena se formará como la concatenación de:
- *la cadena "CANCIONES NO ESCUCHADAS: ", seguida del total de canciones no escuchadas, seguido de un salto de línea;*
- *la información de todas las canciones no escuchadas del repertorio por orden de clave en sentido ascendente, y de tal forma que toda la información sobre una canción esté separada de la siguiente canción con el carácter de salto de línea. La información de cada canción tendrá el siguiente formato: la cadena que identifica a la canción en r, seguida de la cadena ":::", seguida de la cadena "<* ", seguida de una cadena con la información de la canción[1], y finalmente, seguida de la cadena "*>".*
- *la cadena "CANCIONES ESCUCHADAS: ", seguida del total de canciones escuchadas, seguido de un salto de línea;*
- *la información de todas las canciones escuchadas del repertorio con el mismo formato que el indicado para las canciones no escuchadas.*

***La implementación de esta operación deberá hacerse obligatoriamente utilizando las operaciones del iterador de la colecciónConMarca del repertorio**.}*

3. Utilizar la implementación del TAD del apartado 2), para implementar un programa de prueba que nos permita probar la implementación del TAD *repertorio* y del TAD *colecciónConMarca.*

   El código fuente del programa de prueba (*main*) deberá encontrarse en un fichero llamado *"practica2.cpp"* y cumplir escrupulosamente con el funcionamiento y formatos que se describen a continuación, o la práctica no será evaluada.

   El programa de prueba deberá crear un repertorio de canciones vacío con título *"Mi repertorio de canciones"*, y a continuación leer las instrucciones de las operaciones a realizar con el repertorio de canciones desde un fichero de texto denominado "entrada.txt". **Para resolver cada instrucción, el programa deberá utilizar las operaciones ofrecidas por el TAD *repertorio*.** El fichero "entrada.txt" tendrá la siguiente estructura o formato:

[1] De acuerdo al formato descrito en el apartado 2 de la práctica 1.

```
<instrucción1>
<datos para la instrucción1>
...
<datos para la instrucción1>
<instrucción2>
<datos para la instrucción2>
...
<datos para la instrucción2>
...
<instrucciónÚltima>
<datos para la instrucciónÚltima>
...
<datos para la instrucciónÚltima>
```

Donde <instrucciónX> podrá ser alguna de las siguientes cadenas de caracteres: *AC*, *OC*, *EC*, *MES*, *LR*, que representarán las operaciones de: *"Añadir una Canción al repertorio", "Obtener los datos de una Canción", "Eliminar una canción*", "*Modificar EScuchada"* y *"Listar todas las canciones del Repertorio",* respectivamente. El programa finalizará cuando haya procesado todas las instrucciones del fichero. Supondremos que el fichero "entrada.txt" tendrá siempre una estructura como la descrita, sin errores:

- Si la instrucción es *AC*, las 5 líneas siguientes contendrán los valores de: en la primera línea, la clave con la que se deberá introducir la canción en el repertorio; en la segunda línea, el nombre de la canción; en la tercera línea, el autor de la canción; en la cuarta línea, el año en el que fue compuesta la canción; y en la quinta línea, el número de segundos que dura la canción.

- Si la instrucción es *OC* o *EC* en la siguiente línea del fichero aparecerá una clave de la canción del repertorio.

- Si la instrucción es *MES* las dos siguientes líneas contendrán, respectivamente, una clave y un número entero (0/1).

- Si la instrucción es *LR,* la operación no necesita más datos, así que la siguiente línea en el fichero será la del inicio de la siguiente instrucción (o fin de fichero).

Cuando se procese una instrucción *AC*, el programa intentará introducir o modificar en el repertorio una canción, utilizando la clave y los datos dados para la instrucción. Cuando se procese una instrucción *OC*, el programa deberá listar en su salida la información que contiene el repertorio sobre la canción con la clave dada para la instrucción. Cuando se procese una instrucción *EC*, el programa intentará eliminar la canción con la clave dada para la instrucción. Cuando se procese una instrucción *MES*, si el valor entero dado para la instrucción es igual a 0, se intentará marcar como no escuchada la canción con la clave dada, y si el valor entero dado es igual a 1, se intentará marcar como escuchada. Cuando se procese una instrucción *LR*, el programa deberá listar en su salida toda la información del repertorio de acuerdo al formato que se describe más adelante.

Como resultado de su ejecución, el programa creará un fichero de texto "salida.txt" en el que irá escribiendo el resultado de cada orden ejecutada, y que constará de las siguientes líneas:

- Por cada instrucción de tipo *AC*: **si es posible introducir los datos de una nueva canción en el repertorio**, se escribirá una línea en el fichero de salida que empiece con la cadena "INSERCION: "; **si no es posible introducir los datos de una nueva canción en el repertorio pero sí que se actualiza una canción en el repertorio**, se escribirá una línea en el fichero que empiece con la cadena "ACTUALIZACION: ". En cualquiera de los dos casos, se seguirá escribiendo en el fichero la concatenación de:

a.1. la clave de la canción, seguida de la cadena ":::",

a.2. seguida de la información de la canción con el siguiente formato:

  - la cadena "<* ", seguida de la información de la canción[2], seguida de la cadena "*>";
  - seguida de la cadena " escuchada" o " NO escuchada", dependiendo si la canción ha sido escuchada o no, y finalmente un salto de línea.

- Por cada instrucción de tipo *OC*: **si en el repertorio se encuentra una canción con la clave dada**, se escribirá una línea en el fichero de salida que empiece con la cadena "ENCONTRADA: ", seguida de la misma concatenación descrita en los puntos a.1–a.2, con la información sobre la canción en el repertorio. **Si en el repertorio no se encuentra una canción con la clave dada**, se escribirá una línea en el fichero de salida que empiece con la cadena "cancion DESCONOCIDA: ", seguida de la clave utilizada para la instrucción.

- Por cada instrucción de tipo *EC*: **si en el repertorio se encuentra una canción con la clave dada**, se escribirá una línea en el fichero de salida que empiece con la cadena "cancion ELIMINADA: " seguida de la información de la canción eliminada de acuerdo al formato descrito en los puntos a.1–a.2. **Si en el repertorio no se encuentra**

[2] De acuerdo al formato descrito en el apartado 2 de la práctica 1.

una canción con la clave dada, se escribirá una línea en el fichero de salida que empiece con la cadena "eliminacion de cancion INNECESARIA: ", seguida de la clave utilizada para la instrucción.

- Por cada instrucción de tipo *MES*: **si la canción con la clave dada cambia de estar no escuchada a escuchada, o viceversa**, se escribirá una línea con la cadena "cambio ESCUCHADA: ", seguida de la información de la canción de acuerdo al formato descrito en los puntos a.1-a.2. **Si la canción existe en el repertorio, pero no cambia de escuchada a no escuchada, o viceversa**, se escribirá una línea en el fichero de salida con la cadena "cambio ESCUCHADA INNECESARIO: ", seguida de la información de la canción de acuerdo al formato descrito en los puntos a.1-a.2. **Si la canción no existe en el repertorio**, se escribirá una línea en el fichero de salida con la cadena "cambio IMPOSIBLE: ", seguida de la clave utilizada para la instrucción.

  • Por cada instrucción de tipo *LR*: se escribirá en el fichero de salida una primera línea con la cadena "TITULO: ", seguida del título del repertorio, seguido de un salto de línea, seguido de la cadena "NUMERO de canciones: ", seguida del número total de canciones en el repertorio, seguido de un salto de línea, y seguido de la cadena descrita en la operación *listarRepertorio* del TAD repertorio.

Al final se muestra un ejemplo de fichero "entrada.txt", y su correspondiente fichero "salida.txt".

Consideraciones sobre la implementación en C++

En la implementación de la práctica 2, os podéis encontrar que el compilador señala un error indicando algo similar a:

```
note:   template argument deduction/substitution failed
note:   couldn't deduce template parameter 'K' ...
  ...   = nombre_funcion(...sus parámetros);
```

En este caso, el compilador nos está diciendo que al invocar la función genérica "nombre_funcion" (no al declararla, sino al usarla) no puede deducir de qué tipo es el parámetro formal K: K ¿es un string?, ¿un entero? Para solucionar este tipo de errores se debe invocar a la función de la siguiente forma:

```
.... = nombre_funcion<K,V>(...sus parámetros)
```

Esto es, poniendo entre el nombre de la función y la lista de parámetros un "<K,V>", que ayude al compilador a recordar que los parámetros formales son K y V.

Observaciones:

- **El código fuente entregado será compilado y probado en hendrix, que es donde deberá funcionar correctamente.**
- **El código fuente entregado deberá compilar correctamente con la opción –std=c++11 activada.**
  - Esto significa que si se trabaja con la línea de comandos, deberá compilarse con:
    `g++ -std=c++11 ficheros_compilar...`
  - Si se trabaja con CodeBlocks, el proyecto de la práctica deberá estar configurado con la opción `"Have g++ follow the C++11 ISO C++ language standard [-std=c++11]"` activada (localizable en los menús de CodeBlocks, seleccionando sucesivamente: `"Settings"->"Compiler"->->"Global compiler settings"-> pestaña "Compiler Flags"`)
- Todos los ficheros con código fuente que se presenten como solución de esta práctica deberán estar correctamente documentados.
- En el comentario inicial de cada fichero de código fuente se añadirán los nombres y NIAs de los autores de la práctica.
- **Los TADs deberán implementarse siguiendo las instrucciones dadas en las clases y prácticas de la asignatura, y no se permite utilizar Programación Orientada a Objetos.**
- No se permite usar las clases o componentes de la *Standard Template Library (STL)*, ni similares.
- **Todas las indicaciones que se dan en los enunciados de las prácticas respecto a nombres de ficheros, programas, opciones del programa, formatos de los ficheros de entrada o de los ficheros de salida que deban generarse, etc., deben cumplirse escrupulosamente para que la práctica sea evaluada.**
- El código fuente del programa de prueba (*main*) deberá encontrarse en un fichero llamado *"practica2.cpp"*, y cumplir escrupulosamente con el funcionamiento y formatos que se han descrito en el enunciado.
- La ruta del fichero de entrada deberá ser "entrada.txt" (no "entrada1.txt", "mientrada.txt", "datos/entrada.txt", ni similares). Ídem para el fichero "salida.txt".
- La salida debe seguir las especificaciones del enunciado. Por ejemplo, cuando escribimos "cancion DESCONOCIDA: " en el fichero de salida, está escrito sin tilde, la primera palabra en minúsculas, la segunda palabra en mayúsculas, y con un espacio en blanco entre las dos palabras, y otro espacio en blanco tras el carácter ':'. De forma similar, será obligatorio cumplir todos los formatos descritos en este enunciado.

Material a entregar. Instrucciones.

- La práctica solo deberá someterla **uno** de los miembros del equipo de prácticas desde su cuenta de hendrix, y preferiblemente siempre el mismo para todas las prácticas.
- Conectarse a hendrix-ssh.cps.unizar.es según se explica en el documento "Realización y entrega de prácticas en los laboratorios del DIIS" disponible en moodle.
- Crear un directorio, llamado J_p2, si tu profesor tutor es Jose lloret, o Y_p2, si tu profesora tutora es Yolanda Villate, donde se guardará un directorio *practica2* que contenga todos los ficheros desarrollados para resolver la práctica (este directorio, *practica2,* deberá contener todos los ficheros con código fuente C++ necesarios para resolver la práctica y dos ficheros de texto, ***entrada.txt*** y ***salida.txt,*** con los formatos explicados pero que sean significativamente diferentes a los proporcionados como ejemplo, y con los que habréis probado la implementación realizada en vuestra práctica). A la hora de evaluar la práctica se utilizará tanto el fichero de prueba que se entregue, como ficheros de prueba entregados por otros compañeros, o ficheros propios de los profesores.
- Crear el fichero X_p2.tar, con X igual a J o Y, dependiendo de quién sea tu profesor tutor, con el contenido del directorio X_p2 ejecutando el comando

```
tar -cvf X_p2.tar X_p2
```

- Enviar el fichero X _p2.tar, con X igual a J o Y, dependiendo de quién sea tu profesor tutor, mediante la orden

```
someter -v eda_18 X_p2.tar
```

ADVERTENCIA: la orden `someter` no permite someter un fichero si el mismo usuario ha sometido antes otro fichero con el mismo nombre y para la misma asignatura, por lo tanto, **antes de someter vuestra práctica, aseguraos de que se trata de la versión definitiva que queréis presentar para su evaluación.**

entrada.txt

```
OC
C1
AC
LoTR_FR-cd3-1
Khazad-dum
H. Shore
2001
480
AC
BD_ult_cd3-14
Dark Eyes
B. Dylan
1985
309
AC
CM_Sinf3
Symp.N.3
G. Mahler
1896
6300
AC
BD_ult_cd2-13
Hurricane
B. Dylan
1975
52
LR
EC
C1
MES
C1
1
MES
CM_Sinf3
0
OC
BD_ult_cd2-13
MES
BD_ult_cd2-13
1
LR
AC
BD_ult_cd2-13
Hurricane
B. Dylan y J. Levy
1975
512
AC
ACBD_ult_cd1-7
Boots
B. Dylan
1964
278
LR
EC
CM_Sinf3
LR
```

salida.txt

```
cancion DESCONOCIDA: C1
INSERCION: LoTR_FR-cd3-1:::<* Khazad-dum --- H. Shore --- 2001 ( 480 )*> NO escuchada
INSERCION: BD_ult_cd3-14:::<* Dark Eyes --- B. Dylan --- 1985 ( 309 )*> NO escuchada
INSERCION: CM_Sinf3:::<* Symp.N.3 --- G. Mahler --- 1896 ( 6300 )*> NO escuchada
INSERCION: BD_ult_cd2-13:::<* Hurricane --- B. Dylan --- 1975 ( 52 )*> NO escuchada
TITULO: Mi repertorio de canciones
NUMERO de canciones: 4
CANCIONES NO ESCUCHADAS: 4
BD_ult_cd2-13:::<* Hurricane --- B. Dylan --- 1975 ( 52 )*>
BD_ult_cd3-14:::<* Dark Eyes --- B. Dylan --- 1985 ( 309 )*>
CM_Sinf3:::<* Symp.N.3 --- G. Mahler --- 1896 ( 6300 )*>
LoTR_FR-cd3-1:::<* Khazad-dum --- H. Shore --- 2001 ( 480 )*>
CANCIONES ESCUCHADAS: 0
eliminacion de cancion INNECESARIA: C1
cambio IMPOSIBLE: C1
cambio ESCUCHADA INNECESARIO: CM_Sinf3:::<* Symp.N.3 --- G. Mahler --- 1896 ( 6300 )*> NO escuchada
ENCONTRADA: BD_ult_cd2-13:::<* Hurricane --- B. Dylan --- 1975 ( 52 )*> NO escuchada
cambio ESCUCHADA: BD_ult_cd2-13:::<* Hurricane --- B. Dylan --- 1975 ( 52 )*> escuchada
TITULO: Mi repertorio de canciones
NUMERO de canciones: 4
CANCIONES NO ESCUCHADAS: 3
BD_ult_cd3-14:::<* Dark Eyes --- B. Dylan --- 1985 ( 309 )*>
CM_Sinf3:::<* Symp.N.3 --- G. Mahler --- 1896 ( 6300 )*>
LoTR_FR-cd3-1:::<* Khazad-dum --- H. Shore --- 2001 ( 480 )*>
CANCIONES ESCUCHADAS: 1
BD_ult_cd2-13:::<* Hurricane --- B. Dylan --- 1975 ( 52 )*>
ACTUALIZACION: BD_ult_cd2-13:::<* Hurricane --- B. Dylan y J. Levy --- 1975 ( 512 )*> NO escuchada
INSERCION: ACBD_ult_cd1-7:::<* Boots  --- B. Dylan --- 1964 ( 278 )*> NO escuchada
TITULO: Mi repertorio de canciones
NUMERO de canciones: 5
CANCIONES NO ESCUCHADAS: 5
ACBD_ult_cd1-7:::<* Boots  --- B. Dylan --- 1964 ( 278 )*>
BD_ult_cd2-13:::<* Hurricane --- B. Dylan y J. Levy --- 1975 ( 512 )*>
BD_ult_cd3-14:::<* Dark Eyes --- B. Dylan --- 1985 ( 309 )*>
CM_Sinf3:::<* Symp.N.3 --- G. Mahler --- 1896 ( 6300 )*>
LoTR_FR-cd3-1:::<* Khazad-dum --- H. Shore --- 2001 ( 480 )*>
CANCIONES ESCUCHADAS: 0
cancion ELIMINADA: CM_Sinf3:::<* Symp.N.3 --- G. Mahler --- 1896 ( 6300 )*> NO escuchada
TITULO: Mi repertorio de canciones
NUMERO de canciones: 4
CANCIONES NO ESCUCHADAS: 4
ACBD_ult_cd1-7:::<* Boots  --- B. Dylan --- 1964 ( 278 )*>
BD_ult_cd2-13:::<* Hurricane --- B. Dylan y J. Levy --- 1975 ( 512 )*>
BD_ult_cd3-14:::<* Dark Eyes --- B. Dylan --- 1985 ( 309 )*>
LoTR_FR-cd3-1:::<* Khazad-dum --- H. Shore --- 2001 ( 480 )*>
CANCIONES ESCUCHADAS: 0
```