PRÁCTICA 0 **Estructuras de datos y algoritmos**

Título: Programación con TAD's genéricos en C++. **Curso 2018/19**

Objetivo:

- Completar la implementación de una estructura de datos sencilla en C++
- Implementar en C++ el TAD genérico "*diccionario*" y usarlo para implementar un programa que gestione una colección de canciones, identificadas por un código.

Material (descargar de moodle**)**

- Transparencias sobre genéricos en C++.
- Ficheros: *producto.h*, *producto.cc*, *agrupacion.h*

Descripción detallada:

Ejercicio 1.

1. Implementa el TAD contacto definido en las transparencias de la práctica 0.
2. Completa la implementación del TAD agrupación añadiendo al fichero *agrupacion.h* las operaciones del iterador descrito en las transparencias de la práctica 0.
3. Crea un pequeño programa principal en el que:
   a. se introduzcan varios productos en una agrupación de productos;
   b. se muestren por pantalla los datos de todos los productos introducidos;
   c. se borre el último producto, y se vuelvan a mostrar todos los datos;
   d. hacer lo mismo para una agrupación de contactos.

Ejercicio 2.

El enunciado de esta práctica se reutilizará <u>parcialmente</u> en las prácticas posteriores. Se trata de implementar un TAD genérico, particularizarlo y utilizarlo para gestionar una colección de canciones, no repetidas e identificadas por un código. Tienes que realizar las siguientes tareas:

1. Desarrollar una implementación estática (es decir, utilizando un vector, con capacidad para no más de 100 elementos) en C++ de un TAD genérico para representar conjuntos de pares de la forma *(clave*, *dato)*, con claves no repetidas, a la cual llamaremos "*diccionario*". La especificación del TAD diccionario que vamos a considerar es:

```
espec diccionarios
  usa booleanos, naturales, cadenas
parámetros formales
  géneros clave, dato
  operaciones {se requiere que estén definidas las siguientes operaciones de comparación y de
              transformación a cadena:}
    iguales: clave c1, clave c2 -> booleano {Devuelve cierto si y sólo si c1 es igual que c2}
    generaCadena: clave o -> cadena
    generaCadena: dato v -> cadena

fpf

género diccionario {Los valores del TAD representan conjuntos de pares (clave,dato), en los que no
se permiten claves repetidas, y que cuentan con las operaciones de un iterador que permite
recorrer sus pares.}


operaciones
  crear: -> diccionario
   {Devuelve un diccionario vacío, sin elementos (pares).}

  existe?: diccionario g , clave k -> booleano
   {Devuelve verdad si y sólo si en g existe un par con clave k.}

  introducir: diccionario g , clave k , dato v -> diccionario
   {Si en g no existe ningún par con clave k (not existe?(g,k)), devuelve el diccionario
   resultante de añadir en g un par (k,v). Si en g ya existe un par (k,v1) entonces devuelve el
   diccionario resultante de sustituir dicho par en g, por un par (k,v).}

  parcial obtenerDato: diccionario g , clave k -> dato
   {Si en g existe un par (k,v), devuelve su dato v.
   Parcial: la operación no está definida si not existe?(g,k).}

  borrar: diccionario g , clave k -> diccionario
   {Si en g existe un par con clave k, entonces devuelve un diccionario igual al resultante de
   eliminar de g el par con clave k. Si not existe?(g,k), devuelve un diccionario igual a g.}
```

```
cardinal: diccionario g -> natural
{Devuelve el nº de pares en el diccionario g.}

listar: diccionario g -> cadena
 {Devuelve una cadena que contiene la representación, como cadenas de caracteres, de todos los
 pares de g y de tal forma que: los pares estén separados entre sí con el carácter de salto de
 línea; y para cada par su información se incluya con el siguiente formato:
 la cadena que represente la clave k (es decir, generaCadena(k)), a continuación una cadena
 ":::", seguida de la cadena que represente el valor v del par (es decir, generaCadena(v)).}

iniciarIterador: diccionario g -> diccionario
 {Inicializa el iterador para recorrer los pares del diccionario g, de forma que el siguiente
 par sea el primero a visitar (situación de no haber visitado ningún par).}

existeSiguiente?: diccionario g -> booleano
{Devuelve verdad si y sólo si queda algún par por visitar con el iterador del diccionario g.}

parcial siguienteClave: diccionario g -> clave
 {Devuelve la clave del siguiente par a visitar de g.
 Parcial: la operación no está definida si ya se ha visitado el último par.}

parcial siguienteDato: diccionario g -> dato
 {Devuelve el dato del siguiente par a visitar de g.
 Parcial: la operación no está definida si ya se ha visitado el último par.}

parcial avanza: diccionario g -> diccionario
 {Prepara el iterador para visitar el siguiente par del diccionario g.
 Parcial: la operación no está definida si ya se ha visitado el último par.}

fespec
```

La implementación de la operación "listar" deberá hacerse obligatoriamente utilizando las operaciones del iterador.

2. Utilizar la implementación del TAD genérico anterior para implementar un programa C++ que permita probar el TAD gestionando una colección de canciones, que se identificarán por un código. Para ello, el tipo genérico clave se particularizará en una *cadena* de caracteres, y el tipo genérico dato se particularizará en un tipo *canción*. El tipo *canción* deberá poder utilizarse para representar toda la información que corresponda a un concepto de canción que incluya: el nombre de la canción, el nombre del autor (sendos datos de tipo cadena), el año de su composición (de tipo *entero*), y la duración en segundos de la canción (de tipo *entero*). El tipo *canción* se deberá implementar como TAD, de acuerdo a las indicaciones dadas en la asignatura, y ser diseñado con las propiedades y operaciones que se consideren adecuadas. Entre ellas deberá haber una operación que devuelva una cadena conteniendo toda la información de una canción con el formato: *nombre --- autor --- año (duración)* que será necesario para cumplir con los formatos de la salida tal como se describen más adelante.

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

```
<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, LC, EC, LT, que representarán las operaciones de: *"Añadir una Canción a la colección"*, *"Listar todos los datos de una Canción"*, *"Eliminar una Canción*", y *"Listar Todas las canciones"*, 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 la colección; 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 *LC* o *EC*, en la siguiente línea del fichero aparecerá la clave de la canción de la colección que se deberá listar o eliminar, respectivamente.

- Si la instrucción es *LT*, 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 la colección una canción, utilizando la clave y los datos dados para la instrucción. Cuando se procese una instrucción *EC*, el programa deberá eliminar de la colección la canción con la clave dada para la instrucción. Cuando se procese una instrucción *LC*, el programa deberá listar en su salida la información que contiene la colección sobre la canción con la clave dada para la instrucción. Cuando se procese una instrucción *LT*, el programa deberá listar en su salida la información sobre todas las canciones que contiene la colección.

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:

a) Por cada instrucción de tipo *AC*: **si es posible introducir los datos de una nueva canción en la colección**, 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 la colección pero sí que se actualiza una canción en la colección**, se escribirá una línea en el fichero que empiece con la cadena "ACTUALIZACION: "; **si no es posible introducir o actualizar los datos en la colección**, se escribirá una línea en el fichero de salida que empiece con la cadena "insercion CANCELADA: ". En cualquiera de los tres casos, se seguirá escribiendo en el fichero la concatenación de:

   1) la clave de la canción, seguida de la cadena ":::",

   2) seguida de la información de la canción con el siguiente formato:

      - la cadena "<* ", seguida del nombre de la canción, seguido de la cadena " --- ",
      - seguida del nombre del autor, seguido de la cadena " --- ",
      - seguida del año de composición de la canción, seguido de la cadena " (",
      - seguida del número de segundos de duración que tiene la canción, y
      - seguido de la cadena ")*>" , y finalmente un salto de línea.

   Es decir, una canción con el formato: *Clave:::<* nombre --- autor --- año (duración)*>*

b) Por cada instrucción de tipo *LC* **si en la colección se encuentra una canción con la clave dada**, se escribirá una línea en el fichero de salida que empiece con la cadena "ENCONTRADA: ", y seguida de la misma concatenación descrita en los puntos a.1–a.2, con la información de la clave, y datos de la canción en la colección. **Si en la colección 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.

c) Por cada instrucción de tipo *EC*: **si en la colección 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 clave utilizada, seguida de un salto de línea. **Si en la colección no se encuentra una canción con la clave dada**, se escribirá una línea en el fichero de salida que empiece con la cadena "eliminacion INNECESARIA: ", seguida de la clave utilizada para la instrucción.

d) Por cada instrucción de tipo *LT*, se escribirá en el fichero de salida una primera línea con la cadena "TOTAL: ", seguida del número total de canciones de la colección, seguido de un salto de línea, y a continuación la información de todas las canciones de la colección, utilizando para cada una de ellas nuevas líneas de texto de acuerdo al formato descrito en los puntos a.1–a.2,

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

Observaciones:

- No se impondrá ninguna ruta concreta para los ficheros que leerá/escribirá el programa. De esta forma, el fichero de entrada será siempre "entrada.txt" (no "entrada2.txt", "datos/entrada.txt" o similares), y el fichero de salida se llamará "salida.txt", no "salida2.txt", "misalida.txt", etc.
- 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.

- **Tener en cuenta que en las prácticas que presentaréis en breve (esta práctica 0 no se presenta), todas las indicaciones de este tipo serán de obligado cumplimiento.**

entrada.txt

```
LC
C1
AC
CM_GM_Sinf3
Symphony N.3
Gustav Mahler
1896
6300
AC
BD_ult_cd2-13
Hurricane
Bob Dylan
1975
512
AC
LoTR_FR-cd3-1
Khazad-dum
Howard Shore
2001
480
LT
LC
BD_ult_cd2-13
EC
C1
AC
BD_ult_cd2-13
Hurricane
Bob Dylan y J. Levy
1975
512
LT
EC
CM_GM_Sinf3
LC
CM_GM_Sinf3
LT
EC
LoTR_FR-cd3-1
LT
EC
BD_ult_cd2-13
LT
EC
LoTR_FR-cd3-1
```

salida.txt

```
cancion DESCONOCIDA: C1
INSERCION: CM_GM_Sinf3:::<* Symphony N.3 --- Gustav Mahler --- 1896 ( 6300 )*>
INSERCION: BD_ult_cd2-13:::<* Hurricane --- Bob Dylan --- 1975 ( 512 )*>
INSERCION: LoTR_FR-cd3-1:::<* Khazad-dum --- Howard Shore --- 2001 ( 480 )*>
TOTAL: 3
CM_GM_Sinf3:::<* Symphony N.3 --- Gustav Mahler --- 1896 ( 6300 )*>
BD_ult_cd2-13:::<* Hurricane --- Bob Dylan --- 1975 ( 512 )*>
LoTR_FR-cd3-1:::<* Khazad-dum --- Howard Shore --- 2001 ( 480 )*>
ENCONTRADA: BD_ult_cd2-13:::<* Hurricane --- Bob Dylan --- 1975 ( 512 )*>
eliminacion INNECESARIA: C1
ACTUALIZACION: BD_ult_cd2-13:::<* Hurricane --- Bob Dylan y J. Levy --- 1975 ( 512 )*>
TOTAL: 3
CM_GM_Sinf3:::<* Symphony N.3 --- Gustav Mahler --- 1896 ( 6300 )*>
BD_ult_cd2-13:::<* Hurricane --- Bob Dylan y J. Levy --- 1975 ( 512 )*>
LoTR_FR-cd3-1:::<* Khazad-dum --- Howard Shore --- 2001 ( 480 )*>
cancion ELIMINADA: CM_GM_Sinf3
cancion DESCONOCIDA: CM_GM_Sinf3
TOTAL: 2
BD_ult_cd2-13:::<* Hurricane --- Bob Dylan y J. Levy --- 1975 ( 512 )*>
LoTR_FR-cd3-1:::<* Khazad-dum --- Howard Shore --- 2001 ( 480 )*>
cancion ELIMINADA: LoTR_FR-cd3-1
TOTAL: 1
BD_ult_cd2-13:::<* Hurricane --- Bob Dylan y J. Levy --- 1975 ( 512 )*>
cancion ELIMINADA: BD_ult_cd2-13
TOTAL: 0
eliminacion INNECESARIA: LoTR_FR-cd3-1
```