PRÁCTICA 0 **Estructuras de datos y algoritmos**

Título: Programación con TAD's genéricos en C++. **Curso 2021/22**

Objetivos:

- Completar la implementación de una estructura de datos sencilla en C++
- Implementar en C++ un *TAD genérico "colección de ítems temáticos fechados"*, y usarlo para gestionar una colección de *programas informáticos*.

Material (descargar de moodle)

- Transparencias sobre TAD genéricos en C++, y otras relacionadas con el primer ejercicio de este enunciado.
- Ficheros: *producto.h*, *producto.cc*, *agrupacion.h*, y ficheros de ejemplo de entrada/salida para el ejercicio 2.

Descripción detallada:

Ejercicio 1.

a) Implementa el TAD contacto definido en las transparencias de la práctica 0.
b) Completa la implementación del TAD agrupación añadiendo al fichero *agrupacion.h* las operaciones del iterador descrito en las transparencias de la práctica 0.
c) 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. dado el nombre de un producto, compruebe si está o no en la agrupación de productos.
d) Hacer lo mismo para una agrupación de contactos.

Ejercicio 2. El enunciado de este ejercicio se reutilizará parcialmente en las prácticas posteriores. Se trata de implementar un TAD genérico, particularizarlo y utilizarlo para gestionar una colección de programas informáticos. 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 colecciones de *ítems temáticos fechados* (ITF). A un ITF se le requiere tener definidas: una operación *identificador*, que devuelve una cadena que identifica a ese ítem; una operación *tema*, que devuelve una cadena igual al tema del ítem; y otra operación *fecha*, que devuelve la fecha asociada al ítem. La especificación del *TAD colecciónITF* que vamos a considerar es:

```
espec coleccionesITF
  usa booleanos, naturales, cadenas, fechas[1]
parámetro formal
  género ITF {Un ítem temático fechado}
    operación {Se requiere que el género ITF tenga definidas:}
      identificador: ITF i → cadena {operación que devuelve el identificador del ítem}
      tema: ITF i → cadena {operación que devuelve el tema del ítem}
      fecha: ITF i → fecha {operación que devuelve la fecha asociada al ítem}
fpf

género colecciónITF {Los valores del TAD representan colecciones de ITF con identificadores
                     no repetidos.}

operaciones
  crear: → colecciónITF
  {Crea una colección de ITF vacía, sin ítems.}
  añadir: colecciónITF c , ITF i → colecciónITF
  {Si en c no hay ningún ítem con identificador igual al de i, devuelve una colección igual a la
   resultante de añadir el ítem i a la colección c. En caso contrario, devuelve una colección
   igual a c.}
  borrar: cadena ident , colecciónITF c → colecciónITF
  {Si en c hay algún ítem con identificador igual a ident, devuelve una colección igual a la
   resultante de borrar ese ítem de la colección c. En caso contrario, devuelve una colección
   igual a c.}
  está?: cadena ident , colecciónITF c → booleano
  {Devuelve verdad si y sólo si en c hay algún ítem con identificador igual a ident.}
  parcial obtenerITF: cadena ident , colecciónITF c → ITF
  {Si en c hay algún ítem con identificador igual a ident, devuelve dicho ítem.
   Parcial: no definida si en c no hay ningún ítem con identificador igual a ident.}
```

[1] El TAD fecha representa fechas válidas del calendario, similar al visto en clase, y con las operaciones que se considere necesarias.

```
  obsoletos: fecha f , cadena separador , colecciónITF c → cadena
  {Devuelve la cadena resultante de concatenar los identificadores de todos los ítems que hay en c
   con fecha estrictamente anterior a f, en cualquier orden y separados entre sí por la cadena
   separador.}
  purgar: fecha f , colecciónITF c → colecciónITF
  {Devuelve una colección igual a la resultante de eliminar de c todos los ítems con fecha
   estrictamente anterior a f.}
  delTema: cadena t , cadena separador , colecciónITF c → cadena
  {Devuelve la cadena resultante de concatenar los identificadores de todos los ítems que hay en c
   con tema igual a t, en cualquier orden y separados entre sí por la cadena separador.}
  esVacía?: colecciónITF c → booleano
  {Devuelve verdad si y sólo si c no contiene ningún ítem.}
  tamaño: colecciónITF c → natural
  {Devuelve el número de ítems que hay en la colección c.}

  {Las siguientes operaciones constituyen un iterador:}
  iniciarIterador: colecciónITF c → colecciónITF
  {Inicializa el iterador para recorrer los ítems de la colección c, de forma que el siguiente
   ítem a visitar sea el que tiene el (lexicográficamente) menor identificador (situación de no
   haber visitado ningún ítem).}
  haySiguiente?: colecciónITF c → booleano
  {Devuelve verdad si queda algún ítem por visitar con el iterador de la colección c, devuelve
  falso si ya se ha visitado el último ítem.}
  parcial siguiente: colecciónITF c → ITF
  {Devuelve el siguiente ítem a visitar con el iterador de la colección c.
   Parcial: la operación no está definida si no quedan ítems por visitar (no haySiguiente(c).}
  parcial avanza: colecciónITF c → colecciónITF
  {Prepara el iterador de la colección c para que se pueda visitar el siguiente ítem, entendiendo
   por siguiente aquel ítem con (lexicográficamente) menor identificador de los todavía no
   visitados.
   Parcial: la operación no está definida si no quedan ítems por visitar (no haySiguiente(c).}
fespec
```

2. Desarrollar una implementación en C++ del TAD que se especifica a continuación:

```
espec programas
  usa cadenas, fechas

género programa  {Los valores del TAD representan la información de que disponemos sobre un
                  programa informático.}

operaciones
  crear: cadena nom , fecha cre , cadena leng , cadena desc → programa
  {Crea un programa de nombre nom, con fecha de creación cre, codificado en lenguaje leng y con
   descripción desc.}
  suNombre: programa p → cadena
  {Devuelve el nombre del programa p.}
  suFecha: programa p → fecha
  {Devuelve la fecha del programa p.}
  suLenguaje: programa p → cadena
  {Devuelve el lenguaje de codificación del programa p.}
  suDescripción: programa p → cadena
  {Devuelve la descripción del programa p.}
fespec
```

3. Utilizar la implementación del TAD implementado en la tarea 2 y la del TAD genérico implementado en la tarea 1, para implementar un programa de prueba que gestione una colección de programas y nos permita probar la implementación del TAD colecciónITF. Los programas de la colección se identificarán por su nombre. El lenguaje de codificación de un programa será considerado como su tema en la colección. La fecha de un programa en la colección será la fecha del propio programa.

   El programa de prueba deberá crear una colección de programas vacía y, a continuación, leer las instrucciones de las operaciones a realizar con la colección desde un fichero de texto denominado "entrada.txt". El fichero "entrada.txt" tendrá la siguiente estructura o formato:

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

   Donde `<instrucciónX>` podrá ser alguna de las siguientes cadenas: "*A*", "*O*", "*B*", "*P*", "*NO*", "*NL*", "*L*", que representarán respectivamente las ordenes de: *"**A**ñadir un nuevo programa a la colección", "**O**btener el programa de la colección con un nombre dado", "**B**orrar el programa de la colección con un nombre dado", "**P**urgar la colección, es decir, borrar todos los programas con fecha estrictamente anterior a una dada", "listar los **N**ombres de todos los programas **O**bsoletos de la colección, es decir, aquellos con fecha estrictamente anterior a una dada",*

*"listar los **N**ombres de todos los programas de la colección codificados en un **L**enguaje dado" y "**L**istar todos los programas de la colección por orden lexicográfico de su nombre"*. 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, es decir:

- Si la instrucción es "*A",* las siguientes 4 líneas del fichero contendrán los valores de: nombre, fecha, lenguaje y descripción de un programa, respectivamente. La descripción no superará una línea. El formato de la fecha será: *aaaa-mm-dd*.

- Si la instrucción es *"O"* o *"B",* la siguiente línea del fichero contendrá el valor del nombre de un programa.

- Si la instrucción es *"P"* o *"NO"*, la siguiente línea del fichero contendrá el valor de una fecha (formato: *aaaa-mm-dd*).

- Si la instrucción es *"NL"*, la siguiente línea del fichero contendrá el nombre de un lenguaje de programación.

- Si la instrucción es *"L",* ésta no requiere que le sigan otras líneas con más datos para su ejecución, así que la siguiente línea sería la de la siguiente instrucción.

Para resolver cada instrucción, el programa deberá utilizar las operaciones ofrecidas por los TADs implementados en las tareas anteriores. Cuando se procese una instrucción *"A"*, el programa intentará introducir en la colección un nuevo programa, utilizando los datos dados en el fichero para la instrucción. Cuando se procese la instrucción "*O*", el programa intentará obtener un programa de la colección que tenga el nombre dado. Cuando se procese la instrucción *"B"*, el programa deberá eliminar de la colección el programa que tenga el nombre dado. Cuando se procese la instrucción *"P"*, el programa deberá eliminar de la colección todos los programas con fecha estrictamente anterior a la dada. Cuando se procese la instrucción *"NO"*, el programa deberá listar en su salida los nombres de los programas de la colección con fecha estrictamente anterior a la dada, separados por la cadena ":::". Cuando se procese una instrucción *"NL"*, el programa deberá listar en su salida los nombres de todos los programas de la colección codificados en el lenguaje dado, separados por la cadena ":::". Cuando se procese una instrucción *"L"*, el programa deberá listar en su salida la información sobre todos los programas que contiene la colección, por orden lexicográfico de su nombre.

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 *"A"*, **si es posible introducir los datos de un nuevo programa en la colección**, se escribirá una línea en el fichero de salida que empiece con la cadena "INTRODUCIDO: "; **si no es posible introducir el programa en la colección**, se escribirá una línea en el fichero de salida que empiece con la cadena "NO INTRODUCIDO: ". En cualquiera de los dos casos, se seguirá escribiendo a continuación, y en la misma línea del fichero, la concatenación de los datos de: nombre, fecha, lenguaje y descripción (separados entre sí por la cadena ":::") del programa considerado (formato de la fecha: *aaaa-mm-dd*).

- Por cada instrucción de tipo *"O",* **si existía** algún programa en la colección con el nombre dado, se escribirá una línea que empiece con la cadena "LOCALIZADO: ", seguida de la concatenación de los datos de: nombre, fecha, lenguaje y descripción (separados entre sí por la cadena ":::") del programa localizado en la colección. **Si no existía** ningún programa en la colección con el nombre dado, se escribirá una línea que empiece con la cadena "NO LOCALIZADO: ", seguida del nombre utilizado en la operación.

- Por cada instrucción de tipo *"B"*, **si existía** algún programa en la colección con el nombre dado, se escribirá una línea que empiece con la cadena "BORRADO: ", **si no existía** ningún programa en la colección con el nombre dado, se escribirá una línea que empiece con la cadena "NO BORRADO: ". En ambos casos, la cadena escrita irá seguida a continuación, y en la misma línea del fichero, por el nombre utilizado en la operación.

- Por cada instrucción de tipo *"P",* **si existían** programas en la colección con fecha estrictamente anterior a la dada, se escribirá una línea que empiece con la cadena "PURGADOS: ", **si no existía** ningún programa en la colección con fecha estrictamente anterior a la dada, se escribirá una línea que empiece con la cadena "NO PURGADOS: ". En ambos casos, la cadena escrita irá seguida a continuación, y en la misma línea del fichero, por la fecha utilizada en la operación (formato: *aaaa-mm-dd*).

- Por cada instrucción de tipo *"NO",* se escribirá una línea que empiece con la cadena "ANTERIORES A: " seguida de la fecha utilizada en la operación (formato: *aaaa-mm-dd*), seguida de la cadena ":::", seguida de los nombres de los programas de la colección con fecha estrictamente anterior a la dada, separados entre sí por la cadena ":::" (si no hay ningún programa con fecha estrictamente anterior a la dada, no se escribirá nada tras la cadena ":::").

- Por cada instrucción de tipo *"NL",* se escribirá una línea que empiece con la cadena "LENGUAJE: " seguida del lenguaje utilizado en la operación, seguida de la cadena ":::", seguida de los nombres de los programas de la

colección con lenguaje de codificación igual al dado, separados entre sí por la cadena ":::" (si no hay ningún programa con ese lenguaje, no se escribirá nada tras el ":::").

- Por cada instrucción de tipo *"L",* se escribirá en el fichero "*salida.txt*" una primera línea con la cadena "----- LISTADO: ", seguida del número total de programas en la colección y a continuación, por cada programa almacenado en la colección y siguiendo un orden lexicográfico por su nombre, se escribirá otra línea con la concatenación de los datos del programa: nombre, fecha (formato: *aaaa-mm-dd*), lenguaje y descripción (separados entre sí por la cadena ":::"). El listado finalizará con una línea con la cadena "-----". **La implementación de esta instrucción deberá utilizar obligatoriamente las operaciones del iterador de la implementación del *TAD colecciónITF*.**

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

Observaciones.

- El código fuente de las futuras prácticas que entregarás (la 1 y la 2) será **compilado y probado en Hendrix**, que es donde deberá funcionar correctamente. Es recomendable que, aunque esta práctica no se entregue, también te asegures de que se podrá compilar y ejecutar correctamente en Hendrix.
- El código 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 algún entorno de programación (*Visual Studio Code*, *Code::Blocks*, *CodeLite*, etc) y no se utiliza la línea de comandos para compilar, el proyecto de la práctica deberá estar configurado para compilar con la opción –std=c++11.
- 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 "NO INTRODUCIDO: " en el fichero de salida, está escrito en mayúsculas, con un espacio en blanco tras ':'; lo mismo para el resto de instrucciones. **Será obligatorio cumplir todos los formatos descritos en este enunciado**.
- **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.
- **Tener en cuenta que en las prácticas que presentaréis más adelante (esta práctica 0 no se presenta), todas las indicaciones de este tipo serán de obligado cumplimiento.**

entrada.txt

```
A
mergesort
2021-07-01
C++
ordenacion por mezcla
A
radixsort
2021-07-07
C++
ordenacion radix
A
cocktailsort
2021-07-02
Python
ordenacion burbuja bidireccional
B
heapsort
O
heapsort
O
mergesort
A
quicksort
2021-07-03
C++
ordenacion de Hoare
A
quicksort
2021-07-03
C++
metodo rapido
A
burbuja
2021-07-05
Ada
ordenacion intercambio directo
L
B
cocktailsort
NL
C++
NL
Fortran
NO
2021-06-01
P
2021-06-01
NO
2021-07-04
P
2021-07-04
L
```

salida.txt

```
INTRODUCIDO: mergesort:::2021-07-01:::C++:::ordenacion por mezcla
INTRODUCIDO: radixsort:::2021-07-07:::C++:::ordenacion radix
INTRODUCIDO: cocktailsort:::2021-07-02:::Python:::ordenacion burbuja bidireccional
NO BORRADO: heapsort
NO LOCALIZADO: heapsort
LOCALIZADO: mergesort:::2021-07-01:::C++:::ordenacion por mezcla
INTRODUCIDO: quicksort:::2021-07-03:::C++:::ordenacion de Hoare
NO INTRODUCIDO: quicksort:::2021-07-03:::C++:::metodo rapido
INTRODUCIDO: burbuja:::2021-07-05:::Ada:::ordenacion intercambio directo
-----LISTADO: 5
burbuja:::2021-07-05:::Ada:::ordenacion intercambio directo
cocktailsort:::2021-07-02:::Python:::ordenacion burbuja bidireccional
mergesort:::2021-07-01:::C++:::ordenacion por mezcla
quicksort:::2021-07-03:::C++:::ordenacion de Hoare
radixsort:::2021-07-07:::C++:::ordenacion radix
-----
BORRADO: cocktailsort
LENGUAJE: C++:::mergesort:::quicksort:::radixsort
LENGUAJE: Fortran:::
ANTERIORES A: 2021-06-01:::
NO PURGADOS: 2021-06-01
ANTERIORES A: 2021-07-04:::mergesort:::quicksort
PURGADOS: 2021-07-04
-----LISTADO: 2
burbuja:::2021-07-05:::Ada:::ordenacion intercambio directo
radixsort:::2021-07-07:::C++:::ordenacion radix
-----
```

Detalles sobre la implementación con C++:

- Ejemplo de un esquema básico que puede servir para leer el fichero de instrucciones:

```
ifstream f;
f.open("entrada.txt");
string instruccion;
string salto;
while (f >> instruccion) {
    getline(f,salto);
    if (instruccion == "A") {
        string nom;
        //... sigue el programa

    } else if (instruccion == "O") {
        //... sigue el programa

}
//... sigue el programa
```