PRÁCTICA 1 **Estructuras de datos y algoritmos**

Título: Programación de TAD genérico lineal en C++. **Curso 2022/23**

Objetivo:

- Diseñar una implementación lineal y en memoria dinámica en C++ del TAD genérico "*Estudiantes*", y usarlo para implementar un programa que gestione una colección de estudiantes en un proceso de matrícula, de acuerdo a lo que se describe en el enunciado de la práctica.

Fecha de entrega: **11 -11-2022 (GRUPO A) y 18 -11-2022 (GRUPO B)**

Descripción detallada

Se trata de implementar un TAD genérico, particularizarlo y utilizarlo para gestionar una colección de estudiantes, en la que inicialmente podrán inscribirse y borrarse estudiantes, y en la que una vez finalizada la inscripción, podrá realizarse la sucesiva selección por turnos de las asignaturas en las que desea matricularse cada estudiante, además de la posible actualización o eliminación del estudiantes que tenga el turno de matrícula en cada momento.

Tienes que realizar las siguientes **tareas**:

1) Desarrollar una implementación dinámica (utilizando un lista enlazada) en C++ del TAD genérico *Estudiantes*, al cual llamaremos "*Estudiantes*", y cuya especificación se incluye a continuación:

```
spec Estudiantes
   usa cadenas, booleanos, naturales
   parámetros formales
          géneros valor
          operaciones {se requiere que estén definidas las siguientes operaciones de comparación y
          de transformación a cadena:}

                generaCadena: valor v → cadena

   fpf

   género estudiantes {Los valores del TAD representan conjuntos de elementos que son pares
(clave,valor) y en los que no se permiten claves repetidas. Una "agrupación de estudiantes" estará
en todo momento en uno de sus dos estados posibles: Inscripción o Matrícula. Una "agrupación de
estudiantes" en estado de Inscripción podrá cambiar su estado a Matrícula, pero no a la inversa. El
TAD cuenta con operaciones que únicamente pueden utilizarse con una "agrupación de estudiantes" en
estado de Inscripción (añadir y quitar), operaciones que únicamente pueden utilizarse con una
"agrupación de estudiantes" en estado de Matrícula (obtenerValorEstudiante, obtenerClaveEstudiante,
actualizarEstudiante, eliminarEstudiante, pasarTurno), y el resto de las operaciones que pueden
utilizarse con una "agrupación de estudiantes" en cualquiera de sus dos estados posibles. El TAD
cuenta, entre otras, con las operaciones de un iterador que permite recorrer los datos de la
"agrupación de estudiantes" según el orden por clave, de menor a mayor, empezando con el elemento
de menor clave y finalizando con el de mayor clave.

Cuando una "agrupación de estudiantes" no vacía está en estado de Matrícula, existirá siempre un
estudiante destacado (o actual), que será aquel que tenga el turno de Matrícula. Las operaciones
que pueden utilizarse únicamente en la fase de Matrícula permiten consultar y manipular el elemento
que tenga el turno en cada momento, o pasar el turno al siguiente elemento. Al pasar a la fase de
Matrícula, el turno será el de menor clave en la agrupación de estudiantes. El turno pasará siempre
de un elemento al que le siga en el orden por clave de menor a mayor, es decir, a aquel con menor
clave mayor que la suya. Sin embargo, se considerará que tras el elemento de mayor clave, el siguiente
en tener el turno será otra vez el de menor clave.}

   operaciones

      crear: → estudiantes
      {Devuelve una "agrupación de estudiantes" de selección vacía, sin elementos (pares), y en estado
de Inscripción.}

      cardinal: estudiantes est → natural
      {Devuelve el nº de elementos (de pares) en la "agrupación de estudiantes" est}

      esVacía?: estudiantes est → booleano
      {Devuelve verdad si y sólo si est no tiene elementos}

      pertenece?: cadena c, estudiantes est → booleano
      {Devuelve verdad si y sólo si en est hay algún par (c,v)}

      parcial obtenerValor: cadena c, estudiantes est → valor
      {Devuelve el valor asociado a la clave c en un par en est.
      Parcial: la operación no está definida si c no está en est.}

      enMatrícula?: estudiantes est → booleano
      {Devuelve verdad si y sólo si est está en estado de Matrícula.}
```

parcial añadir: estudiantes est, cadena c, valor v → estudiantes
{Si en est no hay ningún par con clave c, devuelve una "agrupación de estudiantes" igual a la resultante de añadir el par (c,v) a est; si en est hay un par (c,v'), entonces devuelve una "agrupación de estudiantes" igual a la resultante de sustituir (c,v') por el par (c,v) en est.
***Parcial:** la operación no está definida si enMatrícula?(est).}*

parcial quitar: cadena c, estudiantes est → estudiantes
{Si c está en est, devuelve una "agrupación de estudiantes" igual a la resultante de borrar en est el par con clave c; si c no está en est, devuelve una "agrupación de estudiantes" igual a est.
***Parcial:** la operación no está definida si enMatrícula?(est).}*

cerrarInscripción: estudiantes est → estudiantes
{Si enMatrícula?(est), devuelve una "agrupación de estudiantes" igual a est; si no enMatrícula?(est), devuelve una "agrupación de estudiantes" igual a la resultante de poner est en estado de Matrícula y además, si no esVacía?(est), fijar el turno en el elemento inscrito en primer lugar en la "agrupación de estudiantes" est.}

parcial pasarTurno: estudiantes est → estudiantes
{Si enMatrícula?(est), devuelve una "agrupación de estudiantes" igual a la resultante de pasar el turno del elemento que actualmente tenga el turno al que le siga en el orden de inscripción, y teniendo en cuenta que si el que tenía el turno era el último inscrito, el turno deberá pasar al primer inscrito.
***Parcial:** la operación no está definida si no es verdad enMatrícula?(est) o si es verdad esVacía?(est).}*

parcial obtenerCandidatoSuClave: estudiantes est → cadena
{Si enMatrícula?(est), devuelve la clave c del par (c,v) que tiene el turno en la "agrupación de estudiantes" est.
***Parcial:** la operación no está definida si no es verdad enMatrícula?(est) o si esVacía?(est).}*

parcial obtenerCandidatoSuValor: estudiantes est → valor
{Si enMatrícula?(est), devuelve el valor v del par (c,v) que tiene el turno en la "agrupación de estudiantes" est.
***Parcial:** la operación no está definida si no es verdad enMatrícula?(est) o si esVacía?(est).}*

parcial actualizarCandidato: estudiantes est, valor v → estudiantes
{Si enMatrícula?(est), devuelve una "agrupación de estudiantes" igual a la resultante de actualizar en est el valor del elemento que tiene el turno, actualizándolo a v.
***Parcial:** la operación no está definida si no es verdad enMatrícula?(est) o si esVacía?(est).}*

parcial eliminarCandidato: estudiantes est → estudiantes
{Si enMatrícula?(est), devuelve una "agrupación de estudiantes" igual a la resultante de eliminar de est el elemento que actualmente tenía el turno en est. Además, en el caso de que queden más elementos en la "agrupación de estudiantes", en la "agrupación de estudiantes" resultante deberá tener el turno el elemento que seguía en est a aquel que tenía el turno (y que ha sido eliminado), teniendo en cuenta que si el que tenía el turno era el último inscrito, el turno deberá pasar al primer inscrito.
***Parcial:** la operación no está definida si no es verdad enMatricula?(est) o si es verdad esVacía?(est).}*

listar: estudiantes est → cadena
{Devuelve una cadena que contiene la representación, como cadenas de caracteres, de todos los elementos de la "agrupación de estudiantes" est ordenados por clave de menor a mayor, separados unos de otros por un salto de línea, y de tal forma que para cada elemento su información se incluya con el siguiente formato:
"clave: xxx"
"valor": [los elementos de valor, pueden ser varias lineas]
"---------------"}

{Operaciones de iterador interno, para recorrer los elementos de la "agrupación de estudiantes" de selección, en orden por clave de menor a mayor:}
iniciarIterador: estudiantes est → estudiantes
{Prepara el iterador y su cursor para que el siguiente elemento (par) a visitar sea el primero de la "agrupación de estudiantes" est (situación de no haber visitado ningún elemento)}

existeSiguiente?: estudiantes est → booleano
{Devuelve falso si ya se ha visitado el último elemento de est, devuelve verdad en caso contrario}

parcial siguienteClave: estudiantes est → cadena
{Devuelve la clave del siguiente elemento (par) de est.
***Parcial:** la operación no está definida si no existeSiguiente?(est)}*

parcial siguienteValor: estudiantes est → valor
{Devuelve el valor del siguiente elemento (par) de est.
***Parcial**: la operación no está definida si no existeSiguiente?(est)}*

parcial avanza: estudiantes est → estudiantes
{Devuelve la "agrupación de estudiantes" resultante de avanzar el cursor del iterador en est.
***Parcial:** la operación no está definida si no existeSiguiente?(est)}*

fespec

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

2) Especificar e implementar un TAD "*unizar"* que defina un nuevo tipo "*unizar"* y que deberá poder utilizarse para representar toda la información que corresponda a un concepto de estudiante de la Universidad de Zaragoza que incluya: una cadena con su nombre, una cadena con la titulación en la que se ha matriculado y un expediente con capacidad máxima para 40 asignaturas identificadas por un código natural de 4 cifras y una calificación representada por un código natural de 2 cifras, donde se pueda añadir una asignatura al expediente o calificar una asignatura referenciada por su código. El tipo *unizar* se deberá implementar como un TAD, de acuerdo a las indicaciones dadas en la asignatura, y ser diseñado con las operaciones y propiedades que se consideren adecuadas. El TAD deberá contar con una operación que dado un estudiante devuelva una cadena que contenga la información del estudiante con el siguiente formato:

```
"nombre: xxxxxxxxxxxxx"
"titulación: xxxxxxxxxxxxxxxx"
"asignaturas: 1-xxxx-yy; 2-xxxx-yy; …; 40-xxxx-yy
```

3) Utilizar los TADs implementados en las tareas anteriores para implementar un programa de prueba que nos permita probar la implementación del TAD "*Estudiantes"* de la tarea 1, de acuerdo a lo que se describe a continuación. La clave será el NIP del estudiante, y el tipo genérico valor se particularizará utilizando el tipo "*unizar"* de la tarea 2.

El programa deberá ofrecer un menú de opciones donde existan las siguientes posibilidades:

- "*Inscribir estudiante":* pedir NIP y titulación del estudiante e incluirlo en la lista. Verificar si la inscripción no está cerrada, si el usuario ya existe y mostrar el resultado de la inscripción.
    - Mensajes: *INSCRIPCION CERRADA. ESTUDIANTE ACTUALIZADO datos. ESTUDIANTE INSCRITO datos.*
- "*Borrar estudiante":* pedir NIP del estudiante y borrarlo. Verificar si la inscripción no está cerrada y si el estudiante existe".
    - Mensajes: *INSCRIPCION CERRADA. ESTUDIANTE BORRADO nip. ESTUDIANTE NO ENCONTRADO.*
- "*Mostrar estudiante":* pedir NIP del estudiante y mostrar sus datos. Verificar si el estudiante existe.
    - Mensajes: *ESTUDIANTE ENCONTRADO nip + datos estudiante. ESTUDIANTE NO ENCONTRADO.*
- "*Cerrar inscripción":* cierra el periodo de inscripción.
    - Mensajes*: INSCRIPCION CERRADA con nº de estudiantes totales x. NO SE ADMITEN MAS INSCRIPCIONES.*
- "*Obtener estudiante en matrícula":* muestra los datos del estudiante en estado de matrícula. Verificar si el estudiante existe y si se ha comenzado el proceso de matrícula.
    - Mensajes*: ESTUDIANTE EN MATRICULA nip + nombre. TODAVÍA NO HA COMENZADO EL PROCESO DE MATRICULA". NO EXISTEN ESTUDIANTES.*
- "*Descartar estudiante en matrícula":* elimina el estudiante que está en proceso de matrícula en ese momento. Verificar si el estudiante existe y si se ha comenzado el proceso de matrícula.
    - Mensajes*: ESTUDIANTE ELIMINADO nip. TODAVÍA NO HA COMENZADO EL PROCESO DE MATRICULA". NO EXISTEN ESTUDIANTES*
- *"Actualizar matrícula estudiante":* actualiza la información del estudiante que tenga el turno con un código numérico de 4 cifras solicitado por pantalla relativo a la asignatura en la que se ha matriculado y su nota correspondiente . Verificar si el estudiante existe, si se ha comenzado el proceso de matrñicula y si el el nº de asignaturas, el código de asignatura o la nota son erróneos
    - Mensajes: *ESTUDIANTE ACTUALIZADO nip + datos estudiante. TODAVÍA NO HA COMENZADO EL PROCESO DE MATRICULA. NO EXISTEN ESTUDIANTES. ERROR AL INTRODUCIR ASIGNATURA.*

- *"Pasar turno"*: pasar el turno de matrícula al siguiente estudiante. Verificar si el estudiante existe y si se ha comenzado el proceso de matrícula

    - Mensajes: *TURNO EN ESTUDIANTE nip + nombre. TODAVÍA NO HA COMENZADO EL PROCESO DE MATRICULA. NO EXISTEN ESTUDIANTES.*

- *"Listar todos los estudiantes"*: muestra información en pantalla de todos los estudiantes y sus asignaturas matriculadas. Muestra el estudiante en turno de matrícula y verifica si todavía no ha comenzado el proceso de matrícula o si no hay estudiantes.

    LISTADO DE ESTUDIANTES

    TOTAL ESTUDIANTES: XX

    nip : xxx

    nombre: xxx

    titulación: xxx

    asignaturas: 1: xxx; 2: …..;

    ---

    nip : xxx

    TURNO DE MATRICULA EN ESTUDIANTE nip

    NO HA COMENZADO EL PROCESO DE MATRICULA

    NO HAY ESTUDIANTES

Observaciones.

- **El código fuente entregado será compilado y probado en CodeBlocks.**
- Para poder trabajar con punteros hay que activar en los "settings" del compilador la opción "-std=C++11"
- 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.
- El código fuente del programa de prueba (*main*) deberá encontrarse en un fichero llamado *"practica1.cpp"*, y cumplir escrupulosamente con el funcionamiento y formatos que se han descrito en el enunciado.