PRÁCTICA 5 — Estructuras de Datos y Algoritmos

Título: Programación con TAD genéricos en C++ — Curso: 2025/26

Objetivos:

- Implementación de un programa principal que utiliza la "*colección de elementos interdependientes",* implementada en prácticas anteriores, para desarrollar un programa que permita la planificación y gestión del tiempo empleado por un usuario, de acuerdo a lo que se describe a continuación.

Descripción detallada:

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

```
especificación tareas
usa cadenas, reales
   género tarea  {DESCRIPCION del TAD: Los valores del TAD tarea representarán tuplas formadas como
   (nombre, descripción, unidadTiempo, tiempoEstimado, tiempoEmpleado), siendo: el nombre una
   cadena, la descripción una cadena, la unidadTiempo una cadena que representará el qué se
   considerará como unidad de tiempo para los tiempos de dicha tarea, tiempoEstimado un número real
   que representará el número total de unidades de tiempo (unidadTiempo) que se estima que será
   necesario invertir en la realización de la tarea, y tiempoEmpleado un número real que representará
   el tiempo total ya empleado en la realización de la tarea (medido en unidades unidadTiempo).
   Inicialmente toda tarea tendrá 0,0 unidades de tiempo empleado. Los tiempos, tiempoEstimado o
   tiempoEmpleado, no podrán ser negativos.

   Para una tarea se podrá utilizar como unidadTiempo cualquier cadena. El TAD no limitará de ninguna
   forma las diferentes unidadTiempo que se puedan llegar a utilizar, ni ofrecerá ninguna operación
   para convertir los tiempos entre diferentes unidades de tiempo, ni ninguna verificación sobre
   correspondencias entre la unidadTiempo usada en la tarea y sus tiempoEstimado o tiempoEmpleado.}
operaciones
  parcial crearTarea: cadena nom, cadena desc, cadena uni, real esti → tarea
  {Dadas tres cadenas y un real, devuelve una tarea compuesta con el nombre nom, la descripción
   desc, con uni como su unidad de tiempo, esti como el tiempo estimado para su realización, y sin
   tiempo empleado en la tarea.
  Parcial: La operación no está definida si esti <0,0}
  nombre: tarea t → cadena
  {Dada una tarea t, devuelve el nombre de la tarea.}
  descripción: tarea t → cadena
  {Dada una tarea t, devuelve la descripción de la tarea.}
  unidad: tarea t → cadena
  {Dada una tarea t, devuelve la unidad de tiempo de la tarea.}
  estimación: tarea t → real
  {Dada una tarea t, devuelve el tiempo estimado para la realización de la tarea.}
  invertido: tarea t → real
  {Dada una tarea t, devuelve el tiempo ya empleado en la realización de la tarea.}
  cambiarNombre: tarea t, cadena n2 → tarea
  {Devuelve una tarea igual a la resultante de sustituir en t, su nombre por la cadena n2.}
  cambiarDescripción: tarea t, cadena d2 → tarea
  {Devuelve una tarea igual a la resultante de sustituir en t, su descripción por la cadena d2.}
  cambiarUnidad: tarea t, cadena u2 → tarea
  {Devuelve una tarea igual a la resultante de sustituir en t, su unidadTiempo por la cadena u2.}
  parcial cambiarTiempoEstimado: tarea t, real estiNuevo → tarea
  {Devuelve una tarea igual a la resultante de sustituir en t, su tiempoEstimado por estiNuevo.
  Parcial: La operación no está definida si estiNuevo <0,0}
  parcial cambiarTiempoEmpleado: tarea t, real inverNuevo → tarea
  {Devuelve una tarea igual a la resultante de sustituir en t, su tiempoEmpleado por inverNuevo.
  Parcial: La operación no está definida si inverNuevo <0,0}
fespec
```

2. Desarrollar un programa principal *planificadorTiempo* que, utilizando el TAD genérico *coleccióInterdep* (implementado en la práctica 3 o en la práctica 4), concretizado con identificadores de tipo *string* (cadena) y con valores del tipo *tarea* (implementado en el apartado anterior), cree una colección de tareas vacía, y a continuación interactúe con el usuario de forma que le permita gestionar una colección de tareas interdependientes, y el tiempo que dedica a ellas, de acuerdo a las reglas que se describen a continuación.

   Por simplificar, el *planificador* no permitirá gestionar colecciones de tareas interrelacionadas de formas excesivamente complejas, sino que limitará la interrelación entre (los identificadores de) sus tareas de forma que:

   - Una tarea nunca podrá ser considerada como prerrequisito directo para ella misma.
   - En cada momento, una tarea podrá ser prerrequisito directo de 1 o ninguna otra tarea.
   - En cada momento, una tarea podrá tener 0 o N tareas como prerrequisitos directos para ella.
   - Las relaciones "prerrequisito directo de/para" entre tareas, siempre deberán darse entre tareas existentes dentro de la colección de tareas gestionada por el planificador.
   - El planificador permitirá registrar tiempo trabajado (o actualizar su tiempo estimado) en cualquiera de las tareas de su colección.
   - Nunca podrá eliminarse una tarea que aún tenga como prerrequisito directo a alguna otra tarea.

- El planificador permitirá cambiar las relaciones "prerrequisito directo de/para" entre tareas de su colección, siempre y cuando se respeten estas reglas.

El programa *planificadorTiempo* deberá permitir la interacción entre el usuario y el programa, mediante un menú de opciones que permita al usuario, al menos:

1. Crear una nueva tarea con la información que indique el usuario, y añadirla (si es posible) a la colección de tareas del planificador. Según lo que indique el usuario, la nueva tarea se añadirá como una tarea que no es prerrequisito directo de ninguna otra, o se añadirá ya como prerrequisito directo de otra tarea de la colección del planificador.
2. Dada la información necesaria para identificar dos tareas, t1 y t2, permita al usuario añadir o cambiar (si es posible) en su colección de tareas, la información de que t1 pasa a ser prerrequisito directo de t2.
3. Dada la información necesaria para identificar una tarea, permita al usuario añadir o cambiar (si es posible) en su colección de tareas, la información de que dicha tarea no es prerrequisito directo de ninguna otra tarea.
4. Dada la información necesaria para identificar una tarea, permita al usuario actualizar (si es posible) la información de dicha tarea en la colección. Se deberá permitir al menos actualizar el tiempo estimado para la tarea, y/o el tiempo empleado en la tarea. Se puede optar por permitir actualizar también el resto de los datos de la tarea (nombre, descripción y unidad de tiempo).
5. Dada la información necesaria para identificar una tarea, consulte si en la colección existe dicha tarea, y en tal caso le muestre en pantalla al usuario toda la información detallada sobre dicha tarea, incluyendo la información de: si dicha tarea es prerrequisito directo de otra tarea o no, y en su caso cuál es la identificación de la tarea que la tiene como prerrequisito directo; y, para cada tarea que sea prerrequisito directo para la tarea consultada, se deberá mostrar al menos la identificación de la tarea prerrequisito, su nombre, unidad de tiempo, tiempo estimado, y tiempo ya empleado en ella.
6. Dada la información necesaria para identificar una tarea, borre (si es posible) dicha tarea.
7. Listar por pantalla toda la información de todas las tareas en la colección del planificador, mostrando para cada una de ellas: todo el detalle de la tarea, cuántas tareas tiene como prerrequisito directo, si dicha tarea es prerrequisito directo para otra tarea o no, y en su caso la identificación de la tarea de la que es prerrequisito directo.
8. Salir del programa.

El programa *planificadorTiempo* deberá mantener informado al usuario en todo momento, mostrando por pantalla si la operación seleccionada ha tenido efecto o no sobre la colección.

Instrucciones.

- **Está práctica no será evaluada, pero es recomendable realizarla.**
- **El código fuente deberá ser compilado y probado en *lab000*, que es donde deberá funcionar correctamente.**
- **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...
    ```
- Todos los ficheros con código fuente deberán estar **correctamente documentados**.
- En el **comentario inicial de cada fichero** de código fuente se añadirán los **nombres completos y NIP de los autores.**
- Los TAD deberán implementarse siguiendo las instrucciones dadas en las clases y prácticas de la asignatura, no se permite utilizar Programación Orientada a Objetos, y tampoco el uso o lanzamiento de excepciones.
- No se permite usar las clases o componentes de la *Standard Template Library (STL)*, ni otras bibliotecas similares.