Práctica 1: Lista de tareas (parte 1)

Programación 2

Curso 2020-2021

Esta práctica consiste en el desarrollo de una lista de tareas pendientes. Los conceptos necesarios para desarrollar esta práctica se trabajan en el *Tema 1* y *Tema 2* de teoría.

Condiciones de entrega

- La fecha límite de entrega para esta práctica es el **viernes 5 de marzo**, hasta las **23:59**
- Debes entregar un único fichero llamado `prac1.cc` con el código de todas las funciones

Código de honor

Si se detecta copia (total o parcial) en tu práctica, tendrás un **0** en la entrega y se informará a la dirección de la Escuela Politécnica Superior para que adopte medidas disciplinarias

Está bien discutir con tus compañeros posibles soluciones a las prácticas
Está bien apuntarte a una academia si sirve para obligarte a estudiar y hacer las prácticas

Está mal copiar código de otros compañeros para resolver tus problemas
Está mal apuntarte a una academia para que te hagan las prácticas

Si necesitas ayuda acude a tu profesor/a
No copies

Normas generales

- Debes entregar la práctica exclusivamente a través del servidor de prácticas del Departamento de Lenguajes y Sistemas Informáticos (DLSI). Se puede acceder a él de dos maneras:
  - Página principal del DLSI (`https://www.dlsi.ua.es`), opción "ENTREGA DE PRÁCTICAS"
  - Directamente en la dirección `https://pracdlsi.dlsi.ua.es`
- Cuestiones que debes tener en cuenta al hacer la entrega:
  - El usuario y la contraseña para entregar prácticas son los mismos que utilizas en UACloud
  - Puedes entregar la práctica varias veces, pero sólo se corregirá la última entrega
  - No se admitirán entregas por otros medios, como el correo electrónico o UACloud
  - No se admitirán entregas fuera de plazo
- Tu práctica debe poder ser compilada sin errores con el compilador de C++ existente en la distribución de Linux de los laboratorios de prácticas

- Si tu práctica no se puede compilar su calificación será 0
- El 70 % de la nota de la práctica dependerá de la corrección automática, por lo que es imprescindible que respetes estrictamente los textos y los formatos de salida que se indican en este enunciado. El otro 30 % dependerá de la revisión manual del código que haga tu profesor de prácticas, por lo que debes también ajustarte a la guía de estilo de la asignatura. Además se tendrá en cuenta que hayas actualizado tu código en *GitHub* todas las semanas
- Al comienzo de todos los ficheros fuente entregados debes incluir un comentario con tu NIF (o equivalente) y tu nombre. Por ejemplo:

prac1.cc

```
// DNI 12345678X GARCIA GARCIA, JUAN MANUEL
...
```

- El cálculo de la nota de la práctica y su relevancia en la nota final de la asignatura se detallan en las transparencias de presentación de la asignatura (*Tema 0*)

1. Descripción de la práctica

Una de las formas de organizar las tareas pendientes de realizar, tanto en un proyecto como a nivel personal, es utilizar un programa para gestionar listas de tareas. Hay muchos programas de este tipo disponibles en el mercado, como Todoist, Any.do, Todo.txt y Trello.

En esta práctica tendrás que desarrollar tu propio programa para gestionar listas de tareas pendientes (aunque más sencillo que los programas mencionados anteriormente). Sus funcionalidades se irán incrementando en las siguientes prácticas.

2. Detalles de implementación

En el Moodle de la asignatura se publicarán varios ficheros que necesitarás para la correcta realización de la práctica:

- `prac1.cc`. Este fichero contiene un esqueleto de programa sobre el que realizar tu práctica. Descárgalo y añade tu código en él. Este fichero contiene la siguiente información:
  - Los registros (`struct`) necesarios para hacer la práctica
  - Un tipo enumerado `Error` que contiene todas las posibles clases de error que se pueden dar en esta práctica (por ejemplo, `ERR_OPTION`)
  - Una función `error` que se encarga de mostrar el correspondiente mensaje de error por pantalla en función del parámetro que se le pase. Por ejemplo, cuando la función recibe el parámetro `ERR_OPTION`, mostrará por pantalla el mensaje "`ERROR: wrong menu option`"
  - Una función `showMainMenu` que muestra por pantalla el menú principal
  - Los prototipos de las funciones que se deben implementar obligatoriamente para gestionar todas las opciones del menú, tal y como se explica en la Sección 6. Además de éstas, deberás incluir las funciones que consideres oportunas para que tu código sea más sencillo y fácil de leer e interpretar por un humano (más concretamente, por el profesor que tiene que corregir tu práctica)
  - Una función `main` que implementa la gestión del menú principal y llama a las funciones correspondientes dependiendo de la opción elegida por el usuario
- `autocorrector-prac1.tgz`. Contiene los ficheros del autocorrector para probar la práctica con algunas pruebas de entrada. La corrección automática de la práctica se realizará con un programa similar, con estas pruebas y otras más definidas por el profesorado de la asignatura

- `prac1`. Fichero ejecutable de la práctica (compilado para máquinas Linux de 64 bits) desarrollado por el profesorado de la asignatura, para que puedas probarlo con las entradas que quieras y ver la salida correcta esperada

- En la corrección de la práctica se introducirán siempre datos del tipo correcto, aunque con valores que pueden ser incorrectos. Por ejemplo, si se pide el tiempo de duración de una tarea, se probará siempre con un valor entero, que podría ser `-1237` o `0`, pero nunca se introducirá un valor de otro tipo (carácter, `string`, número real, ...)
- El resto de decisiones en la implementación de la práctica quedan a tu criterio, pero ten en cuenta que el código fuente será revisado por tu profesor de prácticas siguiendo la guía de estilo publicada en el Moodle de la asignatura. Parte de la nota de la práctica depende de dicha revisión

3. Funcionamiento del programa

El programa que vas a desarrollar te permitirá gestionar las tareas que forman parte de un proyecto. Para organizarlas mejor, estas tareas estarán agrupadas en listas. Por ejemplo, imagina que quieres gestionar las tareas que hay que llevar a cabo en las prácticas de esta asignatura. El nombre del proyecto podría ser *Programación 2*. Las tareas a llevar a cabo se podrían agrupar en una serie de listas para tenerlas más ordenadas. Podríamos crear tres listas que se llamen *Práctica 1*, *Práctica 2* y *Práctica 3*. Finalmente, en cada lista definiríamos las tareas que correspondan. En la lista *Práctica 1* podríamos tener, entre otras, una tarea llamada *Crear función para comprobar fechas* y otra llamada *Probar autocorrector*.

En esta primera práctica se trabajará con un único proyecto. En la siguiente práctica se ampliará la funcionalidad del programa para poder manejar múltiples proyectos.

4. Componentes

El programa a desarrollar debe poder gestionar tres elementos fundamentales: las *tareas*, que son procesos indivisibles, las *listas*, que agrupan tareas relacionadas entre sí y, por último, los *proyectos*, que agrupan varias listas de tareas. Los siguientes apartados describen la estructura de cada uno de estos componentes en detalle.

4.1. Task

Las estructuras de tipo `Task` almacenan la información relativa a una tarea indivisible dentro de una lista asociada a un proyecto. Cada tarea contendrá un nombre que la describa (`name`), un campo que guarde la fecha límite de finalización (`deadline`), un valor booleano que indique si la tarea se ha completado o no (`isDone`), y uno más que indique el tiempo previsto de duración para la tarea (`time`). Esta información se almacenará en un registro con el siguiente formato:

```
struct Task{
  string name;
  Date deadline;
  bool isDone;
  int time;
};
```

El tipo de dato `Date` es a su vez un registro que contiene el día (`day`), mes (`month`) y año (`year`) de la fecha límite establecida por el usuario para finalizar la tarea. Este registro se define de la siguiente manera:

```
struct Date{
  int day;
```

```
  int month;
  int year;
};
```

Por ejemplo, podríamos tener una tarea que se llame *Crear función para borrar listas* (campo `name`), que inicialmente esté sin hacer (campo `isDone`), cuya fecha de finalización sea el 5 (campo `day` de `deadline`) de marzo (campo `month` de `deadline`) de 2021 (campo `year` de `deadline`) y con una duración estimada de 9999 minutos (campo `time`).

- El campo `time` expresa la duración esperada de la tarea en minutos. Por ejemplo, un valor de 120 indicaría 2 horas de duración.

4.2. List

Una lista (`List`) contiene un conjunto de tareas. Tendrá un nombre que la describa (`name`) y un vector que almacene las tareas asociadas a la lista (`tasks`). Toda esta información se almacenará en un registro del siguiente tipo:

```
struct List{
  string name;
  vector<Task> tasks;
};
```

Por ejemplo, podemos tener una lista llamada *Práctica 1* (`name`) con tres tareas almacenadas (`tasks`), cada una con la información descrita en el apartado anterior.

4.3. Project

Un proyecto (`Project`) es una agrupación de listas de tareas. Contendrá un identificador (`id`), un nombre (`name`), una descripción (`description`) y un vector de listas de tareas (`lists`) que seguirán el formato descrito en el apartado anterior. Los proyectos se almacenarán en registros con el siguiente formato:

```
struct Project{
  int id;
  string name;
  string description;
  vector<List> lists;
};
```

Siguiendo el ejemplo inicial, podemos tener un proyecto con el identificador *1* (`id`), cuyo nombre sea *Programación 2* (`name`), cuya descripción sea *Tareas a realizar en la mejor asignatura del Grado en Ingeniería Informática de la UA* (`description`) y que contenga tres listas de tareas (`lists`), una para cada una de las prácticas a realizar durante el curso.

- El campo `id` lo rellenará de manera automática el programa. En esta primera práctica, como solo vamos a manejar un proyecto, tendrá siempre el valor `1`

5. Menú

Al ejecutar la práctica, se mostrará de inicio por pantalla el menú principal del programa, quedando a la espera de que el usuario elija una opción:

Terminal

```
1- Edit project
2- Add list
3- Delete list
4- Add task
5- Delete task
6- Toggle task
7- Report
q- Quit
Option:
```

Las opciones válidas que puede introducir el usuario son los números del `1` al `7` y la letra `q`. Si la opción elegida no es ninguna de éstas (por ejemplo un `9`, una `x` o un `;`), se emitirá el error `ERR_OPTION` llamando a la función `error` con dicho parámetro. Cuando el usuario elige una opción correcta, se debe ejecutar el código asociado a dicha opción. Al finalizarla, volverá a mostrar el menú principal y a pedir otra opción, hasta que el usuario decida salir del programa utilizando la opción `q`.

6. Opciones

En los siguientes apartados se describe el funcionamiento que deberá tener cada una de las siete opciones que se muestran en el menú principal del programa.

6.1. Edit project

Esta opción permite al usuario introducir el nombre (`name`) y la descripción (`description`) del proyecto actual (el único que tendremos en esta práctica), haciendo uso de la función `editProject`. Si el proyecto ya tuviera un nombre y/o una descripción, se sobrescribirá con los nuevos datos introducidos. Esta función se activa cuando el usuario elige la opción `1` del menú principal. El programa pedirá en primer lugar el nombre del proyecto con el siguiente mensaje:

Terminal

```
Enter project name:
```

Si se introduce una cadena vacía, se emitirá el error `ERR_EMPTY` y se volverá a pedir al usuario que introduzca el nombre mostrando el mismo mensaje. Si es correcto, se asignará el nuevo nombre al proyecto y se pedirá a continuación que introduzca la descripción mostrando el siguiente mensaje:

Terminal

```
Enter project description:
```

A diferencia del nombre, la descripción puede quedarse vacía. Tras almacenar la nueva descripción del proyecto el programa volverá al menú principal.

- Si el usuario introduce una descripción vacía, igualmente se guardará como nueva descripción para el proyecto (sobrescribiendo lo que hubiera antes)

6.2. Add list

Esta opción permite la creación de una nueva lista (`List`), haciendo uso para ello de la función `addList`. Ésta se activa cuando el usuario elige la opción `2` del menú principal. En primer lugar se le pedirá al usuario que introduzca el nombre de la lista mediante el siguiente mensaje:

```
Terminal
Enter list name:
```

Si el nombre introducido es una cadena vacía, se emitirá el error `ERR_EMPTY` y se volverá a pedir el nombre con el mismo mensaje. Si ya existe una lista con ese nombre, se mostrará el error `ERR_LIST_NAME` y se volverá al menú principal. Si todo es correcto, se creará un nuevo registro `List` con el nombre (`name`) introducido (de momento con el vector `tasks` vacío) y se añadirá al final del vector `lists` del proyecto.

6.3. Delete list

Esta opción permite eliminar una lista existente, haciendo uso para ello de la función `deleteList`. Ésta se activa cuando el usuario elige la opción `3` del menú. En primer lugar se pedirá al usuario que introduzca el nombre de la lista que se desea eliminar con el siguiente mensaje:

```
Terminal
Enter list name:
```

Si el usuario introduce una cadena vacía se emitirá el error `ERR_EMPTY` y se volverá a pedir el nombre con el mismo mensaje. Si existe una lista con ese nombre, se eliminará de la lista `lists` del proyecto junto con todas sus tareas. En caso contrario, se emitirá el error `ERR_LIST_NAME` y se terminará la función volviendo al menú principal.

6.4. Add task

Esta opción permite crear y añadir una nueva tarea a alguna de las listas del proyecto, haciendo uso para ello de la función `addTask`. Ésta se activa cuando el usuario elige la opción `4` del menú. En primer lugar se le pedirá al usuario que introduzca el nombre de la lista a la que se desea añadir la tarea con el siguiente mensaje:

```
Terminal
Enter list name:
```

Como en las opciones anteriores, si se introduce una cadena vacía se emitirá el error `ERR_EMPTY` y se volverá a solicitar el nombre con el mismo mensaje. Si no existe una lista con ese nombre se emitirá el error `ERR_LIST_NAME` y se volverá al menú principal.

Si el nombre de la lista es correcto, se pedirá el nombre de la nueva tarea con el siguiente mensaje:

```
Terminal
Enter task name:
```

A continuación se pedirá la fecha de finalización al usuario con el siguiente mensaje:

```
Terminal
Enter deadline:
```

La fecha se introducirá con el siguiente formato *día/mes/año* (por ejemplo, `12/5/2021` o `3/10/2022`). Puedes asumir que el formato de entrada siempre será correcto y no hace falta comprobarlo. Lo que sí deberás mirar es si la fecha proporcionada es correcta. Para ello deberás comprobar que el año está comprendido entre 2000 y 2100, además de que el día que se introduzca sea adecuado con respecto al mes (por ejemplo, el 31 de noviembre y el 30 de febrero son fechas erróneas). Si la fecha fuera errónea, se mostrará el mensaje `ERR_DATE` y se volverá al menú principal sin crear la tarea.

Finalmente se pedirá al usuario el tiempo estimado de duración de la tarea en minutos mostrando el mensaje:

```
Terminal
Enter expected time:
```

El tiempo debe estar comprendido entre 1 y 180, ambos incluidos. En caso contrario se emitirá el error `ERR_TIME` y se volverá al menú principal sin crear la tarea. Si toda la información es correcta, se creará un nuevo registro `Task` con la información introducida y se almacenará al final del vector `tasks` de la lista.

- Se permite crear una tarea con el nombre vacío
- Se permite crear una tarea con el mismo nombre que otra, en la misma lista o en otra (puede ser un lío para el usuario, pero está permitido)
- Al crear la tarea, el campo `isDone` debe ponerse a `false`
- A la hora de comprobar las fechas, tendrás que ver si el año introducido es bisiesto, en cuyo caso sería correcto introducir el 29 de febrero como fecha. Un año es bisiesto si es divisible por 4, con una excepción: si además es divisible por 100, debe serlo también por 400 o de lo contrario no será bisiesto. Por ejemplo, 1900 no es bisiesto, pero 2000 sí

6.5. Delete task

Esta opción permite eliminar una tarea de una lista del proyecto, haciendo uso para ello de la función `deleteTask`. Se activa cuando el usuario elige la opción `5` del menú. En primer lugar se pedirá al usuario el nombre de la lista donde está la tarea que queremos eliminar, usando para ello el siguiente mensaje:

```
Terminal
Enter list name:
```

Como en las opciones anteriores, si se introduce una cadena vacía se emitirá el error `ERR_EMPTY` y se volverá a solicitar el nombre con el mismo mensaje. Si no existe una lista con ese nombre se emitirá el error `ERR_LIST_NAME` y se volverá al menú principal.

Si el nombre de la lista es correcto, se pedirá el nombre de la tarea con el siguiente mensaje:

```
Terminal
Enter task name:
```

Si después de recorrer todas las tareas de la lista no hubiera ninguna con ese nombre, se mostrará el mensaje de error `ERR_TASK_NAME` y se volverá al menú principal. Si todo es correcto, se eliminará la tarea de la lista correspondiente.

- En el caso de que la lista dada no tenga ninguna tarea añadida no se actuará de manera especial. Es decir, se leerá el nombre de la tarea y al tratar de localizarla se mostrará el error `ERR_TASK_NAME`, ya que no existe ninguna tarea con ese nombre (ni con ningún otro) en la lista
- Si en la lista hubieran dos o más tareas con el mismo nombre, se eliminarán todas ellas

6.6. Toggle task

Esta opción permite cambiar el estado de una tarea a completada o no, haciendo uso para ello de la función `toggleTask`. Se activa cuando el usuario elige la opción `6` del menú. En este caso se pedirá al usuario el nombre de la lista donde está la tarea que queremos cambiar, usando para ello el siguiente mensaje:

Terminal

```
Enter list name:
```

Como en las opciones anteriores, si se introduce una cadena vacía se emitirá el error `ERR_EMPTY` y se volverá a solicitar el nombre con el mismo mensaje. Si no existe una lista con ese nombre se emitirá el error `ERR_LIST_NAME` y se volverá al menú principal.

Si el nombre de la lista es correcto, se pedirá el nombre de la tarea con el siguiente mensaje:

Terminal

```
Enter task name:
```

Si después de recorrer todas las tareas de la lista no hubiera ninguna con ese nombre, se mostrará el mensaje de error `ERR_TASK_NAME` y se volverá al menú principal. Si todo es correcto, se cambiará el valor del campo `isDone`: si estaba a `false` pasará a valer `true` y viceversa.

- Si la lista estuviera vacía se actuaría igual a como se indicó en `deleteTaks`, es decir, se leerá el nombre de la tarea y al tratar de localizarla se mostrará el error `ERR_TASK_NAME`, ya que no existe ninguna tarea con ese nombre en la lista
- Si en la lista hubieran dos o más tareas con el mismo nombre, se cambiará el estado de todas ellas

6.7. Report

Esta opción mostrará por pantalla un informe con las tareas definidas en el proyecto. Para mostrar esta información se usará la función `report`. Se activa cuando el usuario elige la opción `7` del menú principal. El informe mostrado contendrá la siguiente información:

- Nombre del proyecto (`Name` en el ejemplo)
- Descripción del proyecto (`Description` en el ejemplo)
- Información de cada una de las listas: nombre y conjunto de tareas que contiene
- Para cada tarea hay que mostrar si está completada o no (entre corchetes), su tiempo estimado (entre paréntesis), su fecha de finalización (con el formato `año-mes-día`), el carácter dos puntos (`:`) y el nombre
- Número total de tareas pendientes (`Total left` en el ejemplo) y tiempo total de éstas entre paréntesis
- Número total de tareas realizadas (`Total done` en el ejemplo) y tiempo total de éstas entre paréntesis
- Tarea que tiene la fecha de finalización más antigua y esté sin completar (`Highest priority` en el ejemplo), mostrando únicamente su nombre y la fecha de finalización entre paréntesis (con el formato `año-mes-día`)

A continuación se muestra un ejemplo de salida por pantalla para un proyecto con nombre `My first videogame`, que contiene tres listas (`Design`, `Code` y `Documentation, demo and more`), cada una con una serie de tareas. Aquellas que están completadas aparecen con el símbolo `X` entre corchetes `[X]`, mientras que las pendientes aparecen como un par de corchetes con un espacio en blanco en medio `[ ]`.

Terminal

```
Name: My first videogame
Description: My own project for developing an awesome videogame in C/C++
Design
[ ] (30) 2021-4-12 : Sprites
[ ] (45) 2021-6-10 : Backgrounds
[X] (12) 2021-3-1 : Logo
Code
[ ] (120) 2021-8-1 : Write code
[ ] (60) 2021-8-29 : Unit tests
[ ] (50) 2021-9-2 :
[X] (50) 2021-2-28 : Design classes/UML
Documentation, demo and more
[ ] (30) 2021-10-1 : Write user manual
[ ] (45) 2021-10-15 : Prepare demo inputs
[ ] (35) 2021-11-15 : Record demo
[ ] (40) 2021-12-22 : Publish game (maybe in GitHub)
Total left: 9 (455 minutes)
Total done: 2 (62 minutes)
Highest priority: Sprites (2021-4-12)
```

- Para cada lista, se mostrarán primero las tareas que están pendientes de resolver en el orden en el que fueron introducidas. A continuación se mostrarán las tareas completadas, también en el orden en que fueron introducidas
- Si el proyecto no tiene descripción, no se mostrará la línea con el mensaje `Description`
- Si el nombre de una tarea está vacío, se mostrará toda su información a excepción del nombre. En el ejemplo anterior, la tercera tarea de la lista `Code` tiene el nombre vacío
- A la hora de mostrar la tarea sin completar que tenga la fecha de finalización más antigua, en caso de empate entre dos tareas se mostrará la que primero encuentre. Asumimos que se recorren las listas empezando por la que se creó primero. Asumimos también que dentro de cada lista se recorren las tareas empezando por la que se creó primero
- Si no hay ninguna tarea pendiente en el proyecto, el mensaje de la última línea (`Highest priority:`) no debe aparecer por pantalla
- Aunque el número de minutos en el resumen de tareas pendientes y hechas sea `1`, deberá ponerse `minutes` y no `minute`. Por ejemplo, `Total done: 1 (1 minutes)`