Práctica 3: Lista de tareas (parte 3)

Programación 2

Curso 2020-2021

Esta práctica consiste en implementar la *Práctica 2*, quitando las opciones de manejo de ficheros y gestión de argumentos, siguiendo el paradigma de programación orientada a objetos. Los conceptos necesarios para desarrollar esta práctica se trabajan en todos los temas de teoría, aunque especialmente en el *Tema 5*.

Condiciones de entrega

- La fecha límite de entrega para esta práctica es el **viernes 28 de mayo**, hasta las **23:59**
- La práctica consta de varios ficheros: `Task.cc`, `Task.h`, `List.cc`, `List.h`, `Project.cc`, `Project.h`, `ToDo.cc`, `ToDo.h`, `Util.cc` y `Util.h`. Todos ellos se deberán comprimir en un único fichero llamado `prog2p3.tgz` que se entregará a través del servidor de prácticas de la forma habitual. Para crear el fichero comprimido debes hacerlo de la siguiente manera:

Terminal

```
$ tar cvfz prog2p3.tgz Task.cc Task.h List.cc List.h Project.cc Project.h
  ToDo.cc ToDo.h Util.cc Util.h
```

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
- La corrección de la práctica se hará de forma automática (**no habrá corrección manual del profesor**), por lo que es imprescindible que respetes estrictamente los textos y los formatos de salida que se indican en este enunciado
- Al comienzo de todos los ficheros fuente entregados debes incluir un comentario con tu NIF (o equivalente) y tu nombre. Por ejemplo:

Task.h

```
// 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

En esta práctica se implementará de nuevo la *Práctica 2*, eliminando las funcionalidades relacionadas con ficheros y argumentos, y utilizando en esta ocasión el paradigma de programación orientado a objetos.

Gran parte del código que desarrollaste para la *Práctica 1* y para la *Práctica 2* puede ser reutilizado en esta práctica, por lo que no debes empezar el trabajo desde cero. Debes tratar de adaptar el código que ya tienes y añadir el que consideres necesario para implementar el diseño orientado a objetos.

2. Detalles de implementación

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

- `Util.h` y `Util.cc`. El fichero `Util.h` contiene la definición de la clase `Util` y el tipo enumerado `Error`. Además esta clase contiene el método `error` (implementados en `Util.cc`) que permiten mostrar por pantalla los errores. Se han eliminado los errores y mensajes relacionados con ficheros y argumentos, ya que no deben implementarse para esta práctica
- `prac3.cc`. Fichero que contiene el `main` de la práctica, junto con el código para mostrar el menú de opciones inicial del programa. Este fichero se encarga de crear los objetos de las clases implicadas en el problema, mostrar el menú principal e ir realizando las llamadas a los métodos correspondientes para replicar el funcionamiento de las prácticas anteriores utilizando en esta ocasión programación orientada a objetos. Este fichero no debe modificarse ni incluirse en la entrega

- `prac3`. 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
- `makefile`. Fichero que permite compilar de manera óptima todos los ficheros fuente de la práctica y generar un único ejecutable
- `autocorrector-prac3.tgz`. Contiene los ficheros del autocorrector para probar la práctica con algunas pruebas de entrada. Este autocorrector incluye todas las pruebas del corrector de la Práctica 2 que no usan ficheros ni argumentos. Además contiene varias pruebas unitarias para probar los métodos por separado. La corrección automática de la práctica se realizará con un corrector similar, con estas pruebas y otras más definidas por el profesorado de la asignatura

En esta práctica cada una de las clases se implementará en un módulo diferente, de manera que tendremos dos ficheros para cada una de ellas: `Task.h` y `Task.cc` para las tareas, `List.h` y `List.cc` para las listas, `Project.h` y `Project.cc` para los proyectos, `ToDo.h` y `ToDo.cc` para almacenar todos los proyectos, y `Util.h` y `Util.cc` para las funciones auxiliares. Estos ficheros se deben compilar conjuntamente para generar un único ejecutable. Una forma de hacer esto es de la siguiente manera:

Terminal
```
$ g++ -Wall Task.cc List.cc Project.cc List.cc Util.cc prac3.cc -o prac3
```

Esta solución no es óptima, ya que compila de nuevo todo el código fuente cuando puede que solo alguno de los ficheros haya sido modificado. Una forma más eficiente de realizar la compilación de código distribuido en múltiples ficheros fuente es mediante la herramienta `make`. Debes copiar el fichero `makefile` proporcionado en Moodle dentro del directorio donde estén los ficheros fuente e introducir la siguiente orden:

Terminal
```
$ make
```

- Puedes consultar las transparencias 60 en adelante del *Tema 5* si necesitas más información sobre el funcionamiento de `make`

Algunos de los métodos que vas a crear en esta práctica deben lanzar excepciones. Para ello deberás utilizar `throw` seguido del tipo de excepción (uno de los valores posibles del tipo enumerado `Error`). Esta excepción se deberá capturar mediante `try/catch` donde quiera que se invoque a un método que pueda lanzar una excepción. A continuación se muestra un ejemplo de cómo se lanzaría una excepción desde un método y cómo se capturaría en otro:

```
// Método donde se produce la excepción
List::List(string name){
  ...
  // Si "name" es una cadena vacía lanzamos la excepción con "throw"
  if(...){
    throw ERR_EMPTY;
  }
  ...
}
...
// Método donde se captura la excepción
void Project::addList(string name){
  ...
  try{
```

```
    ...
    List list(name); // Tratamos de crear una nueva lista
    ...
  }
  // Si se produce la excepción, mostramos el error correspondiente
  catch(Error e){
    Util::error(e);
  }
}
```

Para facilitar la posterior corrección por parte del profesorado de la práctica mediante pruebas unitarias (pruebas que evalúan el funcionamiento de cada una de las clases de manera aislada), **deberás declarar los atributos y métodos privados de las clases como** `protected` **en lugar de como** `private`. Un atributo o método `protected` es similar a uno privado. La diferencia es que los atributos o miembros `protected` son inaccesibles fuera de la clase (como los privados), pero pueden ser accedidos por una subclase (clase derivada) que herede de ella. Por ejemplo, la clase `Project` se declararía de esta manera:

```
class Project{
  protected: // Ponemos "protected" en lugar de "private"
    int id;
    ...
  public:
    Project(string name,string description="");
    ...
};
```

A lo largo de este enunciado, siempre que hablemos de métodos o atributos "privados" nos estaremos refiriendo a aquellos que irán en la parte `protected` de las clases que vas a definir.

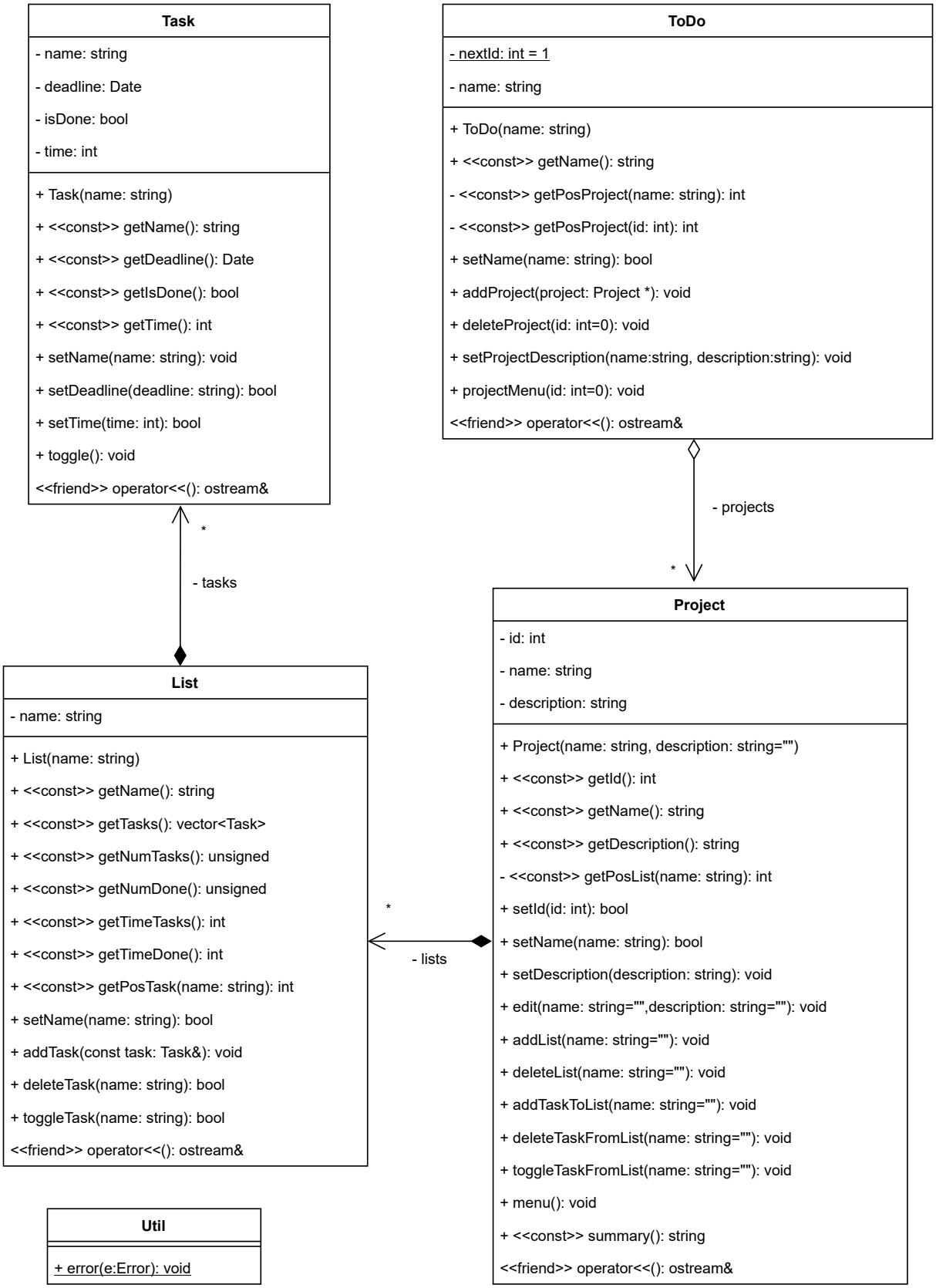
3. Clases y métodos

La figura que aparece en la página siguiente muestra un diagrama UML con las clases que hay que implementar, junto con los atributos, métodos y relaciones que tienen lugar en el escenario de esta práctica.

Si necesitas incorporar más métodos y atributos a las clases descritas en el diagrama, puedes hacerlo, pero siempre incluyéndolos en la parte privada (`protected`) **de las clases**. Recuerda también que las relaciones de *agregación* y *composición* dan lugar a nuevos atributos cuando se traducen del diagrama UML a código. Consulta las transparencias 57 y 58 del *Tema 5* si tienes dudas sobre cómo traducir las relaciones de *agregación* y *composición* a código.

- **¡Ojo!** En esta práctica no está permitido añadir ningún atributo o método público (`public`) a las clases definidas en el diagrama. Sin embargo, como se ha indicado, sí que puedes añadir atributos y métodos privados (`protected`)
- Los atributos de las clases almacenan la misma información que los registros correspondientes de las prácticas anteriores

A continuación se describen los métodos de cada clase. Es posible que algunos de estos métodos no sea necesario utilizarlos para transformar las prácticas anteriores a su versión orientada a objetos, pero se utilizarán en las pruebas unitarias durante la corrección. Se recomienda implementar las clases en el orden en que aparecen en este enunciado.



3.1. Util

En el Moodle de la asignatura se proporcionará esta clase, que incluirá el tipo enumerado `Error` con todos los posibles errores que se pueden dar en la práctica, además del método `error` para emitir los correspondientes errores por pantalla. En principio no es necesario modificar esta clase, pero puedes añadir tus propios métodos (privados) a ella si lo consideras conveniente.

Por ejemplo, para mostrar el mensaje de error `ERR_ID`, deberás invocar al método `error` pasándole el correspondiente parámetro de la siguiente manera:

```
Util::error(ERR_ID);
```


- Ten en cuenta que `error` es un método de clase (`static`) y por eso debe invocarse utilizando esta sintaxis: nombre de la clase, seguido de `::` y finalmente el nombre del método. Consulta la transparencia 45 del *Tema 5* si tienes dudas a este respecto

3.2. Task

Esta clase permite almacenar los datos de las tareas. Los métodos de esta clase son los siguientes:

- `Task(string name)`. Constructor para crear una tarea a la que se le asignará el nombre pasado como parámetro. El campo `time` se inicializará a `1`. Los campos `day`, `month` y `year` de `deadline` se inicializarán a `1`, `1` y `2000` respectivamente. Al campo `isDone` se le dará el valor `false`
- `string getName() const`. Devuelve el nombre de la tarea
- `Date getDeadline() const`. Devuelve la fecha de finalización
- `bool getIsDone() const`. Devuelve el valor del atributo `isDone`
- `int getTime() const`. Devuelve el valor del tiempo
- `void setName(string name)`. Asigna el valor pasado como parámetro al campo `name` de la tarea
- `bool setDeadline(string deadline)`. Recibe una cadena con el formato "dd/mm/aaaa" y extrae cada uno de los campos para asignarlos a los campos `day`, `month` y `year` del atributo `deadline`. Deberá de hacer la comprobación de que la fecha es correcta y en caso de no serlo, no se asignará el valor, se mostrará el error `ERR_DATE` y devolverá `false`. Si la fecha es correcta, el método devolverá `true`
- `bool setTime(int time)`. Asigna el valor pasado como parámetro al campo `time` de la tarea. Si el valor es incorrecto (tal y como se estableció en las prácticas anteriores), no se asignará nada y se deberá mostrar el error `ERR_TIME`. El método devuelve `true` si el valor es correcto y `false` en caso contrario
- `void toggle()`. Cambia el valor del atributo `isDone`, de manera que si es verdadero pasa a valer falso y viceversa
- `ostream& operator<<(ostream &os,const Task &task)`. Operador de salida que muestra por pantalla los datos de la tarea con el mismo formato que la función `report` de la *Práctica 1*. Por ejemplo:

Terminal

```
  [ ] (30) 2021-10-1 : Write user manual
```


- Deberás declarar la estructura `Date` en el fichero `Task.h`, tal y como se definió en las anteriores prácticas, para poder usarla en esta clase

3.3. List

Esta clase permite almacenar los datos de las listas de tareas. Los métodos de esta clase son los siguientes:

- `List(string name)`. Constructor para crear una lista a partir de un nombre pasado como parámetro. Si el nombre pasado como parámetro fuera vacío se deberá **lanzar una excepción** de tipo `ERR_EMPTY` y no se creará el objeto `List`
- `string getName() const`. Devuelve el nombre de la lista
- `vector<Task>getTasks() const`. Devuelve el vector con todas las tareas de la lista
- `unsigned getNumTasks() const`. Devuelve el número de tareas que tiene la lista
- `unsigned getNumDone() const`. Devuelve el número de tareas completadas en la lista
- `int getTimeTasks() const`. Devuelve el número total de minutos que suman las tareas de la lista
- `int getTimeDone() const`. Devuelve el número total de minutos que suman las tareas completadas de la lista
- `int getPosTask(string name) const`. Devuelve la posición de la primera tarea con nombre `name` en el vector `tasks` de la lista, o `-1` si no la encuentra
- `bool setName(string name)`. Modifica el nombre de la lista. Si el nombre introducido es vacío no se asignará nada, se emitirá el error `ERR_EMPTY` y devolverá `false`. Si el nombre se ha podido cambiar, devolverá `true`
- `void addTask(Task task)`. Añade una nueva tarea, pasada como parámetro, a la lista. No hará falta hacer ninguna comprobación adicional sobre la tarea, ya que los métodos de `Task` se encargan de que todos los datos introducidos sean correctos
- `bool deleteTask(string name)`. Busca todas las tareas de la lista que tengan el nombre `name`, pasado como parámetro, y las borra. Si no encuentra ninguna tarea con ese nombre mostrará el error `ERR_TASK_NAME` y devolverá `false`. Si el borrado se ha llevado a cabo devolverá `true`
- `bool toggleTask(string name)`. Busca todas las tareas de la lista con el nombre pasado como parámetro y les cambia el estado de completado a no completado y viceversa. Si no encuentra ninguna tarea con ese nombre mostrará el error `ERR_TASK_NAME` y devolverá `false`. Si ha encontrado alguna taera devolverá `true`
- `ostream& operator<<(ostream &os,const List &list)`. Operador de salida que muestra por pantalla los datos de la lista con el mismo formato que en la función `report` de las prácticas anteriores, mostrando en primer lugar las tareas sin completar y luego las completadas. Éste sería un ejemplo de salida para una lista que tiene tres tareas (una de ellas completada):

Terminal

```
Design
[ ] (30) 2021-4-12 : Sprites
[ ] (45) 2021-6-10 : Backgrounds
[X] (12) 2021-3-1 : Logo
```

- El único método que lanza excepciones en esta clase es el constructor `List`
- En el operador de salida `operator<<` de `List` tendrás que hacer uso del operador de salida de `Task` para mostrar la información de cada una de las tareas que contiene

3.4. Project

Esta clase permite almacenar los datos de los proyectos. Los métodos de esta clase son los siguientes:

- `Project(string name, string description="")`. Constructor para crear un proyecto a partir de un nombre y una descripción pasados como parámetros. Si el nombre pasado como parámetro fuera vacío se deberá lanzar una excepción de tipo `ERR_EMPTY` y no se creará el objeto `Project`. El parámetro `description` tiene como valor por defecto la cadena vacía, de manera que si no se pasa este parámetro la descripción quedará vacía para el nuevo proyecto creado. Al identificador `id` del proyecto se le asignará valor `0`. Cuando se añada el proyecto al programa, como veremos más abajo, es cuando se le asignará un identificador válido
- `int getId() const`. Devuelve el identificador del proyecto
- `string getName() const`. Devuelve el nombre del proyecto
- `string getDescription() const`. Devuelve la descripción del proyecto
- `int getPosList(string name) const`. Devuelve la posición de la lista con nombre `name` en el vector `lists`, o `-1` si no la encuentra
- `bool setId(int id)`. Modifica el identificador del proyecto, asignándole el valor que se le pasa como parámetro. Debe comprobar que el valor sea mayor o igual que `0`. En caso contrario, no se asignará nada, mostrará el error `ERR_ID` y devolverá `false`. Si se ha podido cambiar correctamente el valor, devolverá `true`
- `bool setName(string name)`. Modifica el nombre del proyecto. Si se introduce una cadena vacía, debe dar el error `ERR_EMPTY` y no modificar nada. Si se ha modificado el nombre la función devolverá `true`. Si no, devolverá `false`
- `void setDescription(string description)`. Modifica la descripción del proyecto (puede ser una cadena vacía)
- `void edit(string name="",string description="")`. Permite cambiar el nombre y la descripción del proyecto. Los parámetros pueden omitirse (o ser la cadena vacía). En ese caso, se mostrará el mensaje pidiendo al usuario que introduzca el correspondiente valor. Su comportamiento es el mismo que la opción `Edit Project` de la *Práctica 2*, es decir, hasta que el usuario no introduzca un nombre que no sea la cadena vacía (o se haya pasado el parámetro `name` con un valor distinto de cadena vacía) se seguirá mostrando el mensaje pidiendo el nombre del proyecto. Ten en cuenta que si el valor de `description` es distinto de la cadena vacía, no hará falta mostrar el mensaje `Enter project description:`
- `void addList(string name="")`. Añade una nueva lista con el nombre pasado como parámetro al proyecto. El parámetro puede omitirse (o ser la cadena vacía), en cuyo caso se pedirá al usuario que introduzca el nombre de la lista con el mensaje habitual (`Enter list name:`). El comportamiento de este método, en cuanto a mensajes, errores y comprobaciones, es igual a la opción `Add list` de la *Práctica 2*
- `void deleteList(string name="")`. Borra la lista con nombre `name` del proyecto, es decir, la elimina del vector `lists`. Si el parámetro pasado se omite o contiene la cadena vacía, se pedirá al usuario que introduzca el nombre de la lista con el mensaje habitual. El comportamiento del método, en cuanto a mensajes, errores y comprobaciones, es igual a la opción `Delete list` de la *Práctica 2*
- `void addTaskToList(string name="")`. Añade una nueva tarea a una lista con nombre `name` del proyecto. El parámetro puede omitirse (o ser la cadena vacía), en cuyo caso se pedirá al usuario que introduzca el nombre de la lista con el mensaje habitual. Si el nombre es correcto se pedirá al usuario el nombre de la tarea, la fecha de finalización y el tiempo estimado, se creará la tarea y se añadirá a la lista. Se usarán los mismos mensajes y se emitirán los mismos errores que en la opción `Add task` de la *Práctica 2*

- `void deleteTaskFromList(string name="")`. Borra una o más tareas de la lista con el nombre pasado como parámetro. El parámetro puede omitirse (o ser la cadena vacía), en cuyo caso se pedirá al usuario que introduzca el nombre de la lista con el mensaje habitual. El comportamiento de este método, en cuanto a mensajes, errores y comprobaciones, es igual a la opción `Delete task` de la *Práctica 2*

- `void toggleTaskFromList(string name="")`. Cambia el estado de una o más tareas de la lista de nombre `name`. El parámetro puede omitirse (o ser la cadena vacía), en cuyo caso se pedirá al usuario que introduzca el nombre de la lista con el mensaje habitual. El comportamiento de este método, en cuanto a mensajes, errores y comprobaciones, es igual a la opción `Toggle task` de la *Práctica 2*

- `void menu()`. Permite editar los datos del proyecto, como en la opción `Project menu` de la práctica anterior, mostrando el menú y gestionando las opciones. La única diferencia es que no deberá de mostrarse el mensaje solicitando el `id` del proyecto, sino directamente el menú. Las opciones deberán manejarse empleando los métodos descritos en este enunciado (por ejemplo, el método `deleteTaskFromList` descrito más arriba te va a permitir manejar la opción `Delete task` del menú). En el caso de la funcionalidad `Report` del menú, ésta es equivalente a la información que muestra el operador de salida `operator<<` de la clase `Project`, tal y como se muestra más abajo. Para invocar dentro de `menu` al operador de salida sobre el objeto `Project`, tendrás que hacer `cout << *this << endl;`, donde `*this` hace referencia al propio objeto sobre el que se ha invocado el método `menu` (tiene más información al respecto en la transparencia 46 del *Tema 5*)

- `string summary() const`. Devuelve una cadena con la misma información que la opción `Summary` de la *Práctica 2*. La cadena devuelta contendrá el identificador del proyecto entre paréntesis, un espacio en blanco, el nombre del proyecto, otro espacio en blanco y entre corchetes el número de tareas completadas junto con el número total de tareas (no hay que añadir salto de línea al final). Ejemplo:

Terminal

```
(4) My first videogame [2/11]
```

  En este ejemplo, el proyecto tiene el identificador `4`, su nombre es `My first videogame` y tiene un total de `11` tareas, de las cuales se han completado `2`

- `ostream& operator<<(ostream &os,const Project &project)`. Operador de salida que muestra por pantalla los datos del proyecto con el mismo formato que en las prácticas anteriores. Éste sería un ejemplo de salida para un proyecto:

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
Total left: 5 (305 minutes)
Total done: 2 (62 minutes)
Highest priority: Sprites (2021-4-12)
```

- El único método que lanza excepciones en esta clase es el constructor `Project`
- En el operador de salida `operator<<` tendrás que hacer uso del operador de salida de `List` para mostrar la información de cada lista
- Si necesitas convertir un número a `string`, consulta la transparencia 28 del *Tema 2*

3.5. ToDo

Esta clase permite almacenar los datos de todos los proyectos del programa. Los métodos de esta clase son los siguientes:

- `ToDo(string name)`. Constructor para crear un objeto `ToDo` a partir de un nombre pasado como parámetro. Si el nombre pasado como parámetro fuera vacío se deberá lanzar una excepción de tipo `ERR_EMPTY` y no crear el objeto `ToDo`
- `string getName() const`. Devuelve el nombre almacenado en el objeto `ToDo`
- `int getPosProject(string name) const`. Devuelve la posición del proyecto con nombre `name` en el vector `projects`, o `-1` si no lo encuentra
- `int getPosProject(int id) const`. Devuelve la posición del proyecto con identificador `id` en el vector `projects`, o `-1` si no lo encuentra
- `bool setName(string name)`. Permite asignar un nuevo nombre a `ToDo`. Si se le pasa la cadena vacía, se debe mostrar el error `ERR_EMPTY` y devolverá `false`. Si el nombre se ha cambiado correctamente devolverá `true`
- `void addProject(Project *project)`. Añade un puntero a proyecto a la lista `projects`. Deberá asignarle al proyecto el identificador que haya en `nextId` e incrementar éste, al igual que se hacía en la función `addProject` de las prácticas anteriores. Si ya hay un proyecto con ese nombre en la lista, lanzará la excepción `ERR_PROJECT_NAME` y no lo añadirá
- `void deleteProject(int id=0)`. Busca un proyecto con el identificador pasado como parámetro y lo borra del vector `projects`. Si el parámetro se omite o vale `0`, se le pedirá al usuario que introduzca el identificador del proyecto con el mensaje "`Enter project id:`", igual que se hacía en la opción `Delete project` de la *Práctica 2*. Si no hay ningún proyecto con ese `id` deberá mostrar el error `ERR_ID` y no hacer nada
- `void setProjectDescription(name,description)`. Busca un proyecto con el nombre `name` y le asigna la descripción `description`. Si no existe un proyecto con ese nombre en el vector `projects` se mostrará el error `ERR_PROJECT_NAME` y no se hará nada
- `void projectMenu(int id=0)`. Busca un proyecto con el identificador pasado como parámetro y muestra el menú de edición usando el método `menu` de `Project`. Si el parámetro se omite, o vale `0`, se le pedirá al usuario que introduzca el identificador del proyecto con el mensaje "`Enter project id:`", igual que se hacía en la opción `Project menu` de la *Práctica 2*. Si no hay ningún proyecto con ese `id` mostrará el error `ERR_ID` y no se hará nada más
- `ostream& operator<<(ostream &os,const ToDo &toDo)`. Operador de salida que muestra por pantalla un resumen de los proyectos del programa con el mismo formato que la opción `Summary` de la práctica anterior. Éste sería un ejemplo de salida para un programa que tiene dos proyectos:

Terminal

```
(1) Programacion 2 [1/8]
(2) Juego StarWars [0/4]
```

- Los únicos métodos que lanzan excepciones en esta clase son el constructor `ToDo` y `addProject`
- Es posible tener dos métodos que se llaman igual si los parámetros son diferentes. Es lo que se llama la *sobrecarga* de funciones. Es el caso de `getPosProject`
- En `addProject`, al existir una relación de agregación entre `ToDo` y `Project`, los proyectos se crean en otra parte del programa (en nuestro caso en el programa `prac3.cc`) y lo que se almacena en el vector `projects` son los punteros a esos proyectos

4. Programa principal

El programa principal estará en el fichero `prac3.cc` que se publicará en el Moodle de la asignatura. Este fichero contendrá la función que muestra el menú principal y la función `main` que se encarga de gestionar dicho menú. Este fichero utiliza los principales métodos que ofrecen las clases implementadas, aunque no todos. Te puede servir para entender mejor cómo se utilizan los métodos que has creado para cada una de las clases. Este fichero no debe entregarse con la práctica.