PRÁCTICA 1 **Estructuras de datos y algoritmos**

Título: Programación de TAD genérico lineal en C++. **Curso 2017/18**

Objetivo:

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

Fecha límite de entrega: **8 -11-2017 (incluido)**

Descripción detallada (el enunciado de esta práctica se reutilizará en la siguiente práctica):

Se trata de implementar un TAD genérico, particularizarlo y utilizarlo para gestionar una colección de participantes, en la que inicialmente podrán inscribirse participantes, y en la que una vez finalizada la inscripción, podrá realizarse la sucesiva selección por turnos de los participantes en la ronda, además de la posible actualización o eliminación del participante que tenga el turno en cada momento.

Tienes que realizar las siguientes **tareas**:

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

```
espec rondaSelección
   usa cadenas, booleanos, naturales
   parámetros formales
          géneros clave, valor
          operaciones {se requiere que estén definidas las siguientes operaciones de comparación y
          de transformación a cadena:}
                 generaCadena: clave c → cadena
                 generaCadena: valor v → cadena
                  _=_: clave c1 , clave c2 → booleano
                  _<_: clave c1 , clave c2 → booleano {devuelve verdad sii c1 es menor que c2}
   fpf

   género ronda {Los valores del TAD representan conjuntos de elementos que son pares
(clave,valor) y en los que no se permiten claves repetidas. Una ronda de selección estará en todo
momento en uno de sus dos estados posibles: Inscripción o Selección. Una ronda en estado de
Inscripción podrá cambiar su estado a Selección, pero no a la inversa. El TAD cuenta con
operaciones que únicamente pueden utilizarse con una ronda en estado de Inscripción (añadir y
quitar), operaciones que únicamente pueden utilizarse con una ronda en estado de Selección
(obtenerCandidatoSuValor,  obtenerCandidatoSuClave,  actualizarCandidato,  eliminarCandidato,
pasarTurno), y el resto de las operaciones que pueden utilizarse con una ronda 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 ronda 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 ronda no vacía está en estado de Selección, existirá siempre un elemento destacado ( o
actual), que será aquel que tenga el turno. Las operaciones que pueden utilizarse únicamente en la
fase de Selección 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 Selección, el primer elemento en tener
el turno será el de menor clave en la ronda. 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: → ronda
   {Devuelve una ronda de selección vacía, sin elementos (pares), y en estado de Inscripción.}

   cardinal: ronda r → natural
   {Devuelve el nº de elementos (de pares) en la ronda r}

   esVacía?: ronda r → booleano
   {Devuelve verdad si y sólo si r no tiene elementos}

   pertenece?: clave c , ronda r → booleano
   {Devuelve verdad si y sólo si en r hay algún par (c,v)}

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

   enSelección?: ronda r → booleano
   {Devuelve verdad si y sólo si r está en estado de Selección.}
```

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

parcial quitar: clave c , ronda r → ronda
{Si c está en r, devuelve una ronda igual a la resultante de borrar en r el par con clave c; si c no está en r, devuelve una ronda igual a r.
***Parcial:** la operación no está definida si enSelección?(r).}*

cerrarInscripción: ronda r → ronda
{Si enSelección?(r), devuelve una ronda igual a r; si no enSelección?(r), devuelve una ronda igual a la resultante de poner r en estado de Selección y además, si no esVacía?(r), fijar el turno en el elemento de menor clave en la ronda r.}

parcial pasarTurno: ronda r → ronda
{Si enSelección?(r), devuelve una ronda igual a la resultante de pasar el turno del elemento que actualmente tenga el turno al que le siga en el orden por clave, y teniendo en cuenta que si el que tenía el turno era el de mayor clave, el turno deberá pasar al elemento de menor clave en r.
***Parcial:** la operación no está definida si no es verdad enSelección?(r) o si es verdad esVacía?(r).}*

parcial obtenerCandidatoSuClave: ronda r → clave
{Si enSelección?(r), devuelve la clave c del par (c,v) que tiene el turno en la ronda r.
***Parcial:** la operación no está definida si no es verdad enSelección?(r) o si esVacía?(r).}*

parcial obtenerCandidatoSuValor: ronda r → valor
{Si enSelección?(r), devuelve el valor v del par (c,v) que tiene el turno en la ronda r.
***Parcial:** la operación no está definida si no es verdad enSelección?(r) o si esVacía?(r).}*

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

parcial eliminarCandidato: ronda r → ronda
{Si enSelección?(r), devuelve una ronda igual a la resultante de eliminar de r el elemento que actualmente tenía el turno en r. Además, en el caso de que queden más elementos en la ronda, en la ronda resultante deberá tener el turno el elemento que seguía en r a aquel que tenía el turno (y que ha sido eliminado), teniendo en cuenta que si el que tenía el turno era el de mayor clave, el turno deberá pasar al de menor clave de los que queden.
***Parcial:** la operación no está definida si no es verdad enSelección?(r) o si es verdad esVacía?(r).}*

listar: ronda r → cadena
{Devuelve una cadena que contiene la representación, como cadenas de caracteres, de todos los elementos de la ronda r en orden 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 formato [k::v], es decir: un carácter '[', seguido de la cadena que represente la clave k, a continuación una cadena ":::", seguido a continuación de la cadena que represente el valor v, seguido de un carácter ']', y seguido a continuación del carácter de salto de línea.}

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

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

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

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

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

fespec

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

2) Diseñar e implementar un TAD *Participante* que defina un nuevo tipo *participante* y que deberá poder utilizarse para representar toda la información que corresponda a un concepto de participante que incluya: una sola cadena con sus datos de contacto, y sendos contadores naturales con: su número de aciertos, número de fallos, y número de veces que ha optado por pasar. El tipo *participante* 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 participante devuelva una cadena que contenga la información del participante con el siguiente formato: una cadena "OK: ", seguida del número de aciertos del participante, seguida de la cadena "||F: " seguida del número de fallos del participante, seguida de la cadena "||P: " seguida del número de veces que ha optado por pasar, seguida de la cadena "||D: ", y seguida de la cadena con los datos de contacto del participante.

3) Utilizar los TADs implementados en las tareas anteriores para implementar un programa de prueba que nos permita probar la implementación del TAD *rondaSelección* de la tarea 1, de acuerdo a lo que se describe a continuación. El tipo genérico *clave* se particularizará en una cadena de caracteres que denominaremos *alias* de un participante, y el tipo genérico valor se particularizará utilizando el tipo *participante* de la tarea 2.

   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 describen a continuación, o la práctica no será evaluada.

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

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

   Donde `<instrucciónX>` podrá ser alguna de las siguientes cadenas de caracteres: *ipa*, *bpa*, *mpa*, *ci*, *oc*, *dc*, *ac*, *pt*, *lr*, que representarán las operaciones de: *"Inscribir participante"*, *"Borrar participante"*, *"Mostrar participante"*, *"Cerrar inscripción"*, *"Obtener candidato"*, *"Descartar candidato"*, *"Actualizar candidato"*, *"Pasar turno"*, y *"Listar ronda"*, respectivamente. El programa finalizará cuando haya procesado todas las instrucciones del fichero. Supondremos que el fichero "rondaentrada.txt" tendrá siempre una estructura como la descrita, sin errores.

   Si la instrucción es *ipa*, las 2 líneas siguientes contendrán sendas cadenas de caracteres: en la primera línea, la cadena de caracteres con el *alias* para el participante que se desea inscribir en la ronda de selección; y en la siguiente línea una cadena de caracteres con los datos de contacto para el participante.

   Si la instrucción es *mpa* o *bpa*, la siguiente línea del fichero contendrá la cadena de caracteres con el *alias* que identifique al participante de la ronda con el que se quiere utilizar la instrucción.

   Si la instrucción es *ac*, la siguiente línea del fichero contendrá un carácter 'F', 'A' o 'P'. La operación deberá actualizar la información del participante que tenga el turno, sumándole un fallo, un acierto o un pase, según el carácter leído haya sido 'F', 'A' o 'P' respectivamente.

   Si la instrucción es *ci*, *oc*, *dc*, *pt*, o *lr*, la operación no necesitará más datos, así que la siguiente línea en el fichero será la del inicio de la siguiente instrucción (o fin de fichero).

   Como resultado de su ejecución, el programa deberá ejecutar las instrucciones leídas del fichero "*rondaentrada.txt*", sobre la ronda de participantes creada inicialmente, y además creará un fichero de texto "*rondasalida.txt*" que constará de las siguientes líneas:

   - Por cada instrucción de tipo *ipa*, **si es posible añadir** el participante en la ronda de selección, se escribirá una línea en el fichero de salida que empiece con la cadena: "participante INSCRITO: ", si en la ronda no existía previamente un participante inscrito con el *alias* dado, o con la cadena "participante ACTUALIZADO: ", si en la ronda ya existía previamente un participante inscrito con el *alias* dado (y que deberá haberse actualizado con los datos de contacto dados), en ambos casos la cadena irá seguida de la concatenación de:

a.1 el *alias* utilizado como clave del participante,

a.2 seguido por el carácter ';'

a.3 seguido de una cadena que contenga la información del participante con el formato que se ha descrito en la tarea 2 de este enunciado.

Si no se puede realizar la inscripción del participante, debido a que la operación no está permitida para el estado actual de la ronda, se escribirá una línea en el fichero de salida que empiece con la cadena "inscripcion CERRADA: " seguida de la misma concatenación descrita en los puntos a.1-a.3.

- Por cada instrucción de tipo *bpa*, **si es posible borrar** el participante de la ronda de selección, se escribirá una línea en el fichero de salida que empiece con la cadena: "participante BORRADO: ", seguida de la cadena con el *alias* del participante borrado.

  Si no se puede realizar el borrado del participante, debido a que el participante no se encuentra inscrito en la ronda, se escribirá una línea en el fichero de salida que empiece con la cadena "participante NO ENCONTRADO: ", seguida de la cadena con el *alias* dado. **Si no se puede realizar el borrado** del participante, debido a que la operación no está permitida para el estado actual de la ronda, se escribirá una línea en el fichero de salida que empiece con la cadena "BORRADO participante DESCARTADO: ", seguida de la cadena con el *alias* dado.

- Por cada instrucción de tipo *mpa*, **si es posible encontrar** el participante de la ronda de selección, se escribirá una línea en el fichero de salida que empiece con la cadena: "participante ENCONTRADO: ", seguida de la misma concatenación descrita en los puntos a.1-a.3.

  Si no se puede encontrar el participante, debido a que el participante no se encuentra inscrito en la ronda, se escribirá una línea en el fichero de salida que empiece con la cadena "participante DESCONOCIDO: ", seguida de la cadena con el *alias* dado.

- Por cada instrucción de tipo *ci*, **si es posible cambiar la ronda a estado de Selección**, se escribirá una línea en el fichero de salida que empiece con la cadena: "inscripcion CERRADA con participantes TOTALES: ", seguida del número total de participantes inscritos en la ronda de selección.

  Si el estado de la ronda no puede cambiarse, se escribirá una línea en el fichero de salida que empiece con la cadena "CIERRE de inscripcion DESCARTADO".

- Por cada instrucción de tipo *oc*, **si un participante tiene el turno**, se escribirá una línea en el fichero de salida que empiece con la cadena: "CANDIDATO a evaluar: ", seguida de la información de dicho participante con el mismo formato que la concatenación descrita en los puntos a.1-a.3.

  Si no hay un participante que tenga el turno, se escribirá una línea en el fichero de salida que empiece con la cadena "ronda VACIA".

  Si no se puede realizar la operación, debido a que la operación no está permitida para el estado actual de la ronda, se escribirá una línea en el fichero de salida que empiece con la cadena "CONSULTA candidato DESCARTADA".

- Por cada instrucción de tipo *dc*, **si un participante tiene el turno**, se procederá a eliminarlo de la ronda y se escribirá una línea en el fichero de salida que empiece con la cadena: "candidato ELIMINADO: ", seguida de la información de dicho participante con el mismo formato que la concatenación descrita en los puntos a.1-a.3.

  Si no hay un participante que tenga el turno, se escribirá una línea en el fichero de salida que empiece con la cadena "ronda VACIA".

  Si no se puede realizar la operación, debido a que la operación no está permitida para el estado actual de la ronda, se escribirá una línea en el fichero de salida que empiece con la cadena "ELIMINACION candidato DESCARTADA".

- Por cada instrucción de tipo a*c*, **si un participante tiene el turno**, se procederá a aplicarle el cambio indicado con la instrucción, y se escribirá una línea en el fichero de salida que empiece con la cadena: "candidato ACTUALIZADO: ", seguida de la información con la que queda dicho participante en la ronda, utilizando el mismo formato que la concatenación descrita en los puntos a.1-a.3.

  Si no hay un participante que tenga el turno, se escribirá una línea en el fichero de salida que empiece con la cadena "ronda VACIA".

  Si no se puede realizar la operación, debido a que la operación no está permitida para el estado actual de la ronda, se escribirá una línea en el fichero de salida que empiece con la cadena "ACTUALIZACION candidato DESCARTADA".

- Por cada instrucción de tipo *pt*, **si un participante tiene el turno**, se procederá a pasar el turno a otro participante de la ronda, y se escribirá una línea en el fichero de salida que empiece con la cadena: "TURNO

en candidato: ", seguida de la información del participante que recibe el turno con el mismo formato que la concatenación descrita en los puntos a.1-a.3.

Si no hay un participante que tenga el turno, se escribirá una línea en el fichero de salida que empiece con la cadena "ronda VACIA".

Si no se puede realizar la operación, debido a que la operación no está permitida para el estado actual de la ronda, se escribirá una línea en el fichero de salida que empiece con la cadena "CAMBIO de turno DESCARTADO".

- Por cada instrucción de tipo *lr*, se escribirá en el fichero de salida la concatenación de:

  b.1 una línea de texto con la cadena "***** Ronda en fase: ", seguida de la cadena "INSCRIPCION" o de la cadena "SELECCION" según en qué estado se encuentre la ronda, y seguida de un salto de línea,

  b.2 seguida de una línea de texto con la cadena "TOTAL: ", seguida del número total de participantes en la ronda, y seguido de un salto de línea,

  b.3 seguida del resultado de la operación *listar* para la ronda,

  b.4 en el caso de que la ronda esté en estado de Selección, seguirá una línea de texto con la cadena "TURNO en candidato: ", seguida del *alias* del participante que tenga el turno en la ronda o de la cadena "---" si ningún participante tiene el turno, y seguido de un salto de línea,

  b.5 finalizando con una línea de texto con la cadena "*****" seguida de un salto de línea.

A continuación se incluye a modo de ejemplo un fichero "*rondaentrada.txt*" y su correspondiente fichero "*rondasalida.txt*":

rondaentrada.txt

```
ipa
Bran
Invernalia
ipa
Jon
El Muro
ipa
Aria
Desembarco del Rey
oc
pt
ipa
Eddard
Desembarco del Rey
lr
ipa
Aria
En el camino
mpa
Aria
bpa
Eddard
ci
lr
oc
ac
P
pt
ac
A
lr
ac
A
ac
F
pt
lr
bpa
Aria
dc
lr
dc
dc
pt
lr
```

rondasalida.txt

```
participante INSCRITO: Bran;OK: 0||F: 0||P: 0||D: Invernalia
participante INSCRITO: Jon;OK: 0||F: 0||P: 0||D: El Muro
participante INSCRITO: Aria;OK: 0||F: 0||P: 0||D: Desembarco del Rey
CONSULTA candidato DESCARTADA
CAMBIO de turno DESCARTADO
participante INSCRITO: Eddard;OK: 0||F: 0||P: 0||D: Desembarco del Rey
***** Ronda en fase: INSCRIPCION
TOTAL: 4
[Aria:::OK: 0||F: 0||P: 0||D: Desembarco del Rey]
[Bran:::OK: 0||F: 0||P: 0||D: Invernalia]
[Eddard:::OK: 0||F: 0||P: 0||D: Desembarco del Rey]
[Jon:::OK: 0||F: 0||P: 0||D: El Muro]
*****
participante ACTUALIZADO: Aria;OK: 0||F: 0||P: 0||D: En el camino
participante ENCONTRADO: Aria;OK: 0||F: 0||P: 0||D: En el camino
participante BORRADO: Eddard
inscripcion CERRADA con participantes TOTALES: 3
***** Ronda en fase: SELECCION
TOTAL: 3
[Aria:::OK: 0||F: 0||P: 0||D: En el camino]
[Bran:::OK: 0||F: 0||P: 0||D: Invernalia]
[Jon:::OK: 0||F: 0||P: 0||D: El Muro]
TURNO en candidato: Aria
*****
CANDIDATO a evaluar: Aria;OK: 0||F: 0||P: 0||D: En el camino
candidato ACTUALIZADO: Aria;OK: 0||F: 0||P: 1||D: En el camino
TURNO en candidato: Bran;OK: 0||F: 0||P: 0||D: Invernalia
candidato ACTUALIZADO: Bran;OK: 1||F: 0||P: 0||D: Invernalia
***** Ronda en fase: SELECCION
TOTAL: 3
[Aria:::OK: 0||F: 0||P: 1||D: En el camino]
[Bran:::OK: 1||F: 0||P: 0||D: Invernalia]
[Jon:::OK: 0||F: 0||P: 0||D: El Muro]
TURNO en candidato: Bran
*****
candidato ACTUALIZADO: Bran;OK: 2||F: 0||P: 0||D: Invernalia
candidato ACTUALIZADO: Bran;OK: 2||F: 1||P: 0||D: Invernalia
TURNO en candidato: Jon;OK: 0||F: 0||P: 0||D: El Muro
***** Ronda en fase: SELECCION
TOTAL: 3
[Aria:::OK: 0||F: 0||P: 1||D: En el camino]
[Bran:::OK: 2||F: 1||P: 0||D: Invernalia]
[Jon:::OK: 0||F: 0||P: 0||D: El Muro]
TURNO en candidato: Jon
*****
BORRADO participante DESCARTADO: Aria
candidato ELIMINADO: Jon;OK: 0||F: 0||P: 0||D: El Muro
***** Ronda en fase: SELECCION
TOTAL: 2
[Aria:::OK: 0||F: 0||P: 1||D: En el camino]
[Bran:::OK: 2||F: 1||P: 0||D: Invernalia]
TURNO en candidato: Aria
*****
candidato ELIMINADO: Aria;OK: 0||F: 0||P: 1||D: En el camino
candidato ELIMINADO: Bran;OK: 2||F: 1||P: 0||D: Invernalia
ronda VACIA
***** Ronda en fase: SELECCION
TOTAL: 0
TURNO en candidato: ---
*****
```

Observaciones.

- **El código fuente entregado será compilado y probado en hendrix, que es donde deberá funcionar correctamente.**
- **El código fuente entregado 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 CodeLite, el proyecto de la práctica deberá estar configurado con la opción `"Enable C++11 features [-std=c++11]"` activada (localizable haciendo clic sobre la carpeta del proyecto en CodeLite y seleccionando sucesivamente: `"Settings.."->"Compiler"->"C++ compiler options"`)
- 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.
- **Todas las indicaciones que se dan en los enunciados de las prácticas respecto a nombres de ficheros, programas, opciones del programa, formatos de los ficheros de entrada o de los ficheros de salida que deban generarse, etc., deben cumplirse escrupulosamente para que la práctica sea evaluada.**
- 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.
- La ruta del fichero de entrada deberá ser "rondaentrada.txt" (no "rondaentrada1.txt", "entrada.txt", "datos/rondaentrada.txt", ni similares). Ídem para el fichero "rondasalida.txt".
- La salida debe seguir las especificaciones del enunciado. Por ejemplo, cuando escribimos "inscripcion CERRADA: " en el fichero de salida, está escrito con la primera palabra en minúsculas y la segunda palabra en mayúsculas, sin tilde, y con un espacio en blanco tras ':'. De forma similar, será obligatorio cumplir todos los formatos descritos en este enunciado.

Material a entregar. Instrucciones.

- La práctica solo deberá someterla **uno** de los miembros del equipo de prácticas desde su cuenta de hendrix, y preferiblemente siempre el mismo para todas las prácticas.
- Conectarse a hendrix-ssh.cps.unizar.es según se explica en el documento "Realización y entrega de prácticas en los laboratorios del DIIS" disponible en moodle.
- Crear un directorio, llamado J_p1, si tu profesor tutor es Jorge Bernad, o Y_p1, si tu profesora tutora es Yolanda Villate, donde se guardará un directorio *practica1* que contenga todos los ficheros desarrollados para resolver la práctica (este directorio, *practica1*, deberá contener todos los ficheros con código fuente C++ necesarios para resolver la práctica y los ficheros de texto ***rondaentrada.txt*** y ***rondasalida.txt*** con los formatos explicados, pero que sean significativamente diferentes a los proporcionados como ejemplo, y con los que habréis probado la implementación realizada en vuestra práctica). A la hora de evaluar la práctica se utilizará tanto el fichero de prueba que se entregue, como ficheros de prueba entregados por otros compañeros, o ficheros propios de los profesores.
- Crear el fichero X_p1.tar, con X igual a J o Y, dependiendo de quién sea tu profesor tutor, con el contenido del directorio X_p1 ejecutando el comando
  ```
  tar –cvf X_p1.tar X_p1
  ```
- Enviar el fichero X _p1.tar, con X igual a J o Y, dependiendo de quién sea tu profesor tutor, mediante la orden
  ```
  someter –v eda_17 X_p1.tar
  ```

  ADVERTENCIA: la orden `someter` no permite someter un fichero si el mismo usuario ha sometido antes otro fichero con el mismo nombre y para la misma asignatura, por lo tanto, **antes de someter vuestra práctica, aseguraos de que se trata de la versión definitiva que queréis presentar para su evaluación.**