PRÁCTICA 2 **Estructuras de datos y algoritmos**

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

Objetivo:

Diseñar una implementación arborescente y en memoria dinámica en C++ el TAD genérico "*colección con marca de tiempo*", y utilizar dicha implementación, y la del TAD "*ronda de selección*" de la práctica anterior, para implementar un TAD "*concursos de preguntas"*.

Fecha límite de entrega: **21-12-2017 (incluido)**

Descripción detallada:

Se trata de implementar un TAD genérico, particularizarlo y utilizarlo, junto con el TAD "*ronda de selección*" y el TAD "*Participantes*" de la práctica anterior, para implementar un TAD "*concursos de preguntas"* que gestione un concurso basado en comprobar el conocimiento de los concursantes sometiéndoles, por turno, a una serie de preguntas de respuesta única.

Tienes que realizar las siguientes tareas:

1) Desarrollar una implementación dinámica en C++ , utilizando un **árbol binario de búsqueda** (ABB), del TAD genérico para representar colecciones de elementos de la forma *(clave,dato,tiempo)*, con claves no repetidas, y que pueden tener o no un valor de tiempo, sin límite en el número total de elementos que pueda contener la colección, y a la cual llamaremos *"colección con marca de tiempo"*. La especificación del TAD colecciónConMarcaTiempo que vamos a considerar es la que se incluyó ya en el enunciado de la práctica 0.

```
espec colecciónConMarcaTiempo
  usa booleanos, naturales, caracteres, cadenas
parametros formales
  géneros clave, dato, tiempo
  operaciones {se requiere que estén definidas las siguientes operaciones de comparación y de
               transformación a cadena:}
    generaCadena: clave o -> cadena
    generaCadena: dato v -> cadena
    generaCadena: tiempo t -> cadena

    mayorClave: clave o1, clave o2 -> booleano {Devuelve cierto si y sólo si o1 es mayor que o2}
    tiempoPosterior: tiempo t1, tiempo t2 -> booleano {Devuelve cierto sii t1 es posterior a t2}
fpf

género cmt
{Los valores del TAD representan conjuntos de ternas (clave,dato,tiempo), en los que no están
permitidas claves repetidas. Toda terna de la colección deberá tener siempre asignados valores para
sus dos primeras componentes: clave y dato, pero su tercera componente podrá tener o no un valor
asignado. Por tanto, toda terna de la colección será de la forma (k,v,t) o (k,v,-), donde '-'
representa que aún no tiene asignado ningún valor para la tercera componente de la terna. El TAD
cuenta, entre otras, con una operación para dar valor a la tercera componente de una terna.}

operaciones
  crear: -> cmt
   {Crea una colección vacía, sin ternas}

  esVacío?: cmt c -> booleano
   {Devuelve verdad si y sólo si c no contiene ninguna terna.}

  existeClave?: cmt c , clave k -> booleano
   {Devuelve verdad si y sólo si en c existe una terna con clave k.}

  introducir: cmt c , clave k , dato v -> cmt
   {Si en c no existe ninguna terna con clave k (not existeClave?(c,k)), devuelve la colección
  resultante de añadir en c una terna (k,v,-). Si en c ya existe una terna (k,v1,t) o (k,v1,-)
  entonces devuelve la colección resultante de sustituir dicha terna en c, por una terna (k,v,-)}

  tieneTiempo?: cmt c , clave k -> booleano
   {Si en c existe una terna con clave k y un valor de tiempo asignado (k,v,t), devuelve verdad.
  En cualquier otro caso, devuelve falso.}

  parcial obtenerDato: cmt c , clave k -> dato
   {Si en c existe una terna (k,v,t) o (k,v,-), devuelve su dato v.
  Parcial: la operación no está definida si not existeClave?(c,k).}

  parcial obtenerTiempo: cmt c , clave k -> tiempo
   {Si en c existe una terna (k,v,t) devuelve su valor t.
  Parcial: la operación no está definida si not existeClave?(c,k) o si not tieneTiempo?(c,k).}

  parcial marcarTiempo: cmt c , clave k , tiempo t -> cmt
   {Si en c ya existe una terna (k,v,t1) o (k,v,-), entonces devuelve la colección resultante de
```

```
  sustituir dicha terna en c, por una terna (k,v,t).
  Parcial: la operación no está definida si not existeClave?(c,k).}

  borrar: cmt c , clave k -> cmt
   {Si en c existe una terna con clave k, entonces devuelve una colección igual a la resultante de
  eliminar de c la terna con clave k. Si not existeClave?(c,k), devuelve una colección igual a c.}

  tamaño: cmt c -> natural
   {Devuelve el nº de ternas en la colección c.}

  listar: cmt c -> cadena
   {Devuelve una cadena que contiene la representación, como cadenas de caracteres, de todas las
  ternas de c, tengan o no valor de tiempo asignado, en orden por clave de menor a mayor, y de tal
  forma que: las ternas estén separadas entre sí con el carácter de salto de línea; y para cada terna,
  sea de la forma (k,v,t) o (k,v,-), su información se incluya con el siguiente formato:
  un carácter '[', seguido de la cadena que represente el valor la clave k, a continuación una cadena
  ":::", seguida de la cadena que represente el valor del tiempo t o del carácter '-' si es que la
  terna no tiene tiempo asignado, seguido del carácter de salto de línea, y seguido a continuación de
  la cadena que represente el valor del dato v, seguido de un carácter de salto de línea, y seguido
  del carácter ']'.
     [k:::t             ó     [k:::-
     v                        v
     ]                        ]
  }

  iniciarIterador: cmt c -> cmt
   {Inicializa el iterador para recorrer las ternas de la colección c, de forma que la siguiente
  terna a visitar sea la primera (situación de no haber visitado ninguna terna).}

  existeSiguiente?: cmt c -> booleano
  {Devuelve verdad si y sólo si queda alguna terna por visitar con el iterador de la colección c.}

  parcial siguienteClave: cmt c -> clave
   {Devuelve la clave de la siguiente terna a visitar de c.
  Parcial: la operación no está definida si ya se ha visitado la última terna.}

  parcial siguienteDato: cmt c -> dato
   {Devuelve el dato de la siguiente terna a visitar de c.
  Parcial: la operación no está definida si ya se ha visitado la última terna.}

  parcial siguienteTiempo: cmt c -> tiempo
   {Devuelve el tiempo de la siguiente terna a visitar de c.
  Parcial: la operación no está definida si ya se ha visitado la última terna, o si la siguiente terna
  a visitar es de la forma (k,v,-) y por tanto no tiene asignado un valor de tiempo.}

  parcial avanza: cmt c -> cmt
   {Prepara el iterador para visitar la siguiente terna de la colección c.
  Parcial: la operación no está definida si ya se ha visitado la última terna.}

fespec
```

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

2) Utilizar la **implementación del TAD *colecciónConMarcaTiempo* del punto 1) de esta práctica**, y la implementación del TAD *rondaSelección* y del TAD *Participantes* de la práctica anterior, para implementar en C++ el TAD "*concursos de preguntas*" que se especifica a continuación.

El concurso contará con una colección de preguntas que será una *colecciónConMarcaTiempo*, utilizada concretando su tipo genérico *clave* con un tipo de números *enteros*, el tipo genérico *tiempo* se particularizará en un tipo *Instante* (con la información de hora y minuto), y el tipo *dato* se particularizará en un tipo *Pregunta*. El tipo *Pregunta* deberá poder utilizarse para representar toda la información que corresponda a un concepto de pregunta que incluya: el texto o interrogante con el que se formula o expresa la pregunta (de tipo *cadena*), dos opciones de respuesta entre las que elegir (respuesta 1 y respuesta 2, sendas *cadenas*), y cuál de las dos respuestas será considerada como la respuesta correcta (la 1ª o la 2ª). **El tipo *Instante* y el tipo *Pregunta* se deberán implementar en sendos TADs: *Instantes* y *Preguntas***, de acuerdo a las indicaciones dadas en la asignatura, y ser diseñados con las operaciones y propiedades que se consideren adecuadas.
El TAD *Instantes* deberá contar con una operación que dado un *instante* devuelva una cadena que contenga la información de la hora y minutos del instante, en formato *horas*:*minutos*. El TAD *Preguntas* deberá contar con una operación que dada una *pregunta* devuelva una cadena que contenga la información de la pregunta con el siguiente formato:

- la cadena "<*", seguida del texto del interrogante de la pregunta, seguido de la cadena "*>:", seguido de un salto de línea, y a continuación la cadena "<A) " seguida del texto de la primera opción de respuesta de la pregunta, seguida de la cadena " ; B) " seguida del texto de la segunda opción de respuesta de la pregunta, seguida de la cadena " ; OK: ", seguido de un 1 o un 2 según cual sea la respuesta correcta, seguida de la cadena " >", y finalmente un salto de línea.

El concurso contará con una colección de concursantes que será una *rondaSelección*, utilizada concretando su tipo genérico *clave* con el tipo cadenas de caracteres, el tipo genérico *valor* se particularizará con el tipo descrito e implementado en el TAD *Participantes* de la práctica 2.

La especificación del TAD "*concursos de preguntas*" que vamos a considerar es:

espec concursosDePreguntas
  usa booleanos, naturales, cadenas, Participantes, Instantes, Preguntas, rondaSelección(cadena,Participante), colecciónConMarcaTiempo(natural,Pregunta,Instante)[1]

género concurso *{Los valores del TAD representan concursos consistentes en poner a prueba el conocimiento de sus concursantes sometiéndoles a una serie de preguntas de respuesta única, descartando a los concursantes que alcancen el máximo de respuestas equivocadas permitido.*
Para ello, el concurso contará con una colección de preguntas y una colección de concursantes, es decir, los participantes en el concurso.

En la colección de preguntas no podrá haber varias preguntas con el mismo identificador (número natural). Cada pregunta podrá estar o no marcada con un instante de tiempo, que corresponderá al momento en el que ha sido utilizada por última vez.

Las respuestas de los concursantes se registrarán adecuadamente en el detalle de la información de cada uno de ellos, contabilizando sus aciertos, fallos y el número de veces que ha elegido pasar sin responder. En la colección de concursantes no podrá haber varios concursantes con el mismo identificador (cadena).

Un concurso estará en todo momento en uno de sus tres estados o fases posibles: Documentación, Inscripción o Juego. Inicialmente el concurso estará en fase de Documentación, después podrá pasar a fase de Inscripción, y por último pasará a la fase de Juego.

Para garantizar la equidad de los concursos, únicamente se podrán añadir o borrar preguntas a la colección de preguntas del concurso cuando este se encuentre en fase de Documentación. Igualmente, solo se podrán añadir o borrar participantes al concurso, es decir concursantes, mientras este se encuentre en fase de Inscripción.

Cuando el concurso esté en su fase de Juego, los concursantes serán puestos a prueba planteándoles, por turnos, preguntas a las que deberán responder con una de las dos respuestas posibles o eligiendo pasar sin responder. El primer concursante en tener el turno y ser puesto a prueba, será aquel con el menor identificador de concursante de todos los que estén concursando. El turno irá pasando de un concursante al siguiente, según el orden de menor a mayor identificador de concursante, es decir pasará al concursante con menor identificador mayor que el que cede el turno. Cuando el turno lo tenga aquel con el mayor identificador de concursante de todos los que estén participando en el concurso, el turno pasará a continuación al concursante con menor identificador de concursante de todos los que queden en el concurso.

Cuando un concurso es creado, se fijan ciertos límites máximos que no podrán ser cambiados posteriormente, estos son: el número máximo de veces que se permitirá a cada concursante elegir pasar sin responder a una pregunta; el número máximo de fallos que se permitirá a cada concursante, cuando un concursante alcance dicho número máximo de fallos será descalificado eliminándolo del concurso; el número máximo de concursantes que podrán ser considerados como ganadores del concurso, aunque el concurso pueda seguir y acabar con aún menos concursantes.}

operaciones
  crearConcurso: natural PMax, natural FMax, natural numGmax -> concurso
   {Crea un concurso de preguntas vacío, sin preguntas ni concursantes, y en fase de Documentación, pero en el que una vez se entre en la fase de Juego se permitirá pasar a cada concursante un máximo de PMax veces, se descalificará a un concursante al alcanzar FMax fallos, y que tendrá como objetivo seleccionar numGmax concursantes, que se serán considerados los ganadores del concurso.}

  máximoPasesPermitidos: concurso c -> natural
   {Devuelve el número total de pases que se permitirá a cada participante del concurso.}
  máximoFallosPermitidos: concurso c -> natural
   {Devuelve el número total de fallos con los que se descalificará a cada participante del concurso.}
  máximoNúmeroGanadores: concurso c -> natural
   {Devuelve el número máximo de ganadores a seleccionar con el concurso.}
 enDocumentación?: concurso c -> booleano
  {Devuelve verdad si y sólo si el concurso c está en fase de Documentación.}
 enInscripción?: concurso c -> booleano
  {Devuelve verdad si y sólo si el concurso c está en fase de Inscripción.}
 enJuego?: concurso c -> booleano
  {Devuelve verdad si y sólo si el concurso c está en fase de Juego.}

 parcial añadirPregunta: concurso c, natural IDpreg, pregunta p -> concurso
  {Si en la colección de preguntas del concurso c no hay ninguna pregunta identificada con IDpreg, devolverá un concurso igual al resultante de añadir a su colección de preguntas una nueva pregunta con clave IDpreg, información p, y marcada indicando que aún no ha sido utilizada.
 Si en la colección de preguntas del concurso c ya había una pregunta identificada con IDpreg,

[1] Las especificaciones de los TAD genéricos *rondaSelección* y *colecciónConMarcaTiempo*, que aquí utilizamos con elementos de los tipos definidos respectivamente en los TADs: *Participantes*, *Preguntas*, *Instantes*, *cadenas* de caracteres y números *naturales*.

devolverá un concurso igual al resultante de actualizar en su colección de preguntas la pregunta con clave IDpreg, actualizando su información a p, y marcada indicando que aún no ha sido utilizada.
Parcial: la operación no está definida si no es verdad enDocumentación?(c).(situación: concursoYaDocumentado).}

parcial borrarPregunta: concurso c, natural IDpreg -> concurso
{Si en la colección de preguntas del concurso c hay alguna pregunta identificada con IDpreg, devolverá un concurso igual al resultante de eliminar de su colección de preguntas la pregunta con clave IDpreg. Si en la colección de preguntas del concurso c no hay ninguna pregunta identificada con IDpreg, devolverá un concurso igual a c.
Parcial: la operación no está definida si no es verdad enDocumentación?(c).(situación: concursoYaDocumentado).}

totalPreguntas: concurso c -> natural
{Devuelve el número total de preguntas en la colección de preguntas del concurso c.}

existePregunta?: concurso c, natural IDpreg -> booleano
{Devuelve verdad si y sólo si en la colección de preguntas del concurso c hay alguna pregunta identificada con IDpreg.}

parcial obtenerPregunta: concurso c, natural IDpreg -> pregunta
{Si en la colección de preguntas del concurso c hay una pregunta identificada con IDpreg, devuelve la información de la pregunta identificada con IDpreg.
Parcial: la operación no está definida si en la colección de preguntas del concurso c no hay ninguna pregunta identificada por IDpreg (situación: preguntaNoExistente).}

parcial preguntaUtilizada?: concurso c, natural IDpreg -> booleano
{Si en la colección de preguntas del concurso c hay una pregunta identificada con IDpreg, devuelve verdad si y sólo si la pregunta está marcada indicando que ha sido utilizada.
Parcial: la operación no está definida si en la colección de preguntas del concurso c no hay ninguna pregunta identificada por IDpreg (situación: preguntaNoExistente).}

parcial obtenerÚltimoUsoPregunta: concurso c, natural IDpreg -> instante
{Si en la colección de preguntas del concurso c hay una pregunta identificada con IDpreg, devuelve el instante que corresponde al tiempo de la última vez que ha sido utilizada.
Parcial: la operación no está definida si en la colección de preguntas del concurso c no hay ninguna pregunta identificada por IDpreg (situación: preguntaNoExistente), o si la pregunta aún no ha sido utilizada (situación: preguntaNoUtilizada).}

parcial marcarPregunta: concurso c, natural IDpreg, instante t -> concurso
{Si en la colección de preguntas del concurso c hay alguna pregunta identificada con IDpreg, devolverá un concurso igual al resultante de actualizar en su colección de preguntas que la pregunta con clave IDpreg ha sido utilizada por última vez en el instante de tiempo dado t.
Parcial: la operación no está definida si en la colección de preguntas del concurso c no hay ninguna pregunta identificada por IDpreg. (situación: preguntaNoExistente).}

iniciarInscripción: concurso c -> concurso
{Si enDocumentación*?(c) y totalPreguntas(c)>0, devuelve un concurso igual al resultante de poner c en estado de Inscripción. Si no es verdad* enDocumentación*?(c), o si totalPreguntas(c)=0, devuelve un concurso igual a c.}*

parcial añadirConcursante: concurso c, cadena IDconcursante, participante p -> concurso
{Si en la colección de participantes en el concurso c no hay ninguno identificado con IDconcursante, devolverá un concurso igual al resultante de añadir a su colección de concursantes uno con clave IDconcursante e información p.
Si en la colección de concursantes de c ya había uno identificado con IDconcursante, devolverá un concurso igual al resultante de actualizar en su colección de concursantes que para el concursante con clave IDconcursante su información es p.
Parcial: la operación no está definida si no es verdad enInscripción?(c). (situación: concursoNoEnInscripción).}

parcial borrarConcursante: concurso c, cadena IDconcursante -> concurso
{Si en la colección de concursantes de c hay alguno identificado con IDconcursante, devolverá un concurso igual al resultante de eliminar de su colección al concursante con clave IDconcursante.
Si en la colección de concursantes de c no había ninguno identificado con IDconcursante, devolverá un concurso igual a c.
Parcial: la operación no está definida si no es verdad enInscripción?(c). (situación: concursoNoEnInscripción).}

existeConcursante? concurso c, cadena IDconcursante -> booleano
{Devuelve verdad si y sólo si en la colección de concursantes de c hay algún concursante identificado con IDconcursante.}

parcial obtenerInfoConcursante: concurso c, cadena IDconcursante -> participante
{Si en la colección de concursantes de c hay un concursante identificado con IDconcursante, devuelve la información de dicho concursante.
Parcial: la operación no está definida si en la colección de concursantes de c no hay ningún concursante identificado por IDconcursante (situación: concursanteNoExistente).}

totalConcursantes: concurso c -> natural
{Devuelve el número total de concursantes que hay en el concurso c.}

iniciarJuego: concurso c -> concurso

{Si enInscripción?(c) y totalConcursantes(c)>máximoNúmeroGanadores(c), devuelve un concurso igual al resultante de poner c en estado de Juego, y preparar los concursantes para que el primero que sea puesto a prueba sea aquel con menor IDconcursante.
Si no es verdad enInscripción?(c), o si totalConcursantes(c)≤ máximoNúmeroGanadores(c), devuelve un concurso igual a c.}

existeConcursanteActual?: concurso c -> booleano
{Si enJuego?(c) y totalConcursantes(c)>0, devolverá verdad indicando así que existe un concursante que actualmente tiene el turno para jugar, es decir, que le toca responder a preguntas. Devolverá falso en cualquier otro caso.}

parcial obtenerConcursanteActual: concurso c -> cadena
{Si enJuego?(c) y totalConcursantes(c)>0, devuelve la cadena que corresponde al IDconcursante del concursante que actualmente tiene el turno para jugar, es decir, de aquel al que le toca responder a preguntas.
Parcial: la operación no está definida si no es verdad enJuego?(c)(situación: concursoNoEnJuego), o si es verdad enJuego?(c) y totalConcursantes(c)=0 (situación: sinConcursantes).}

hayGanadores?: concurso c -> booleano
{Devuelve verdad si y sólo si es verdad enJuego?(c) y totalConcursantes(c)≤ máximoNúmeroGanadores(c). Devuelve falso en cualquier otro caso.}

parcial probarConcursanteActual: concurso c, entero IDpreg, instante t, natural resp -> concurso
{Si enJuego?(c) y totalConcursantes(c)>0, la operación registrará que el concursante actual ha dado la respuesta resp a la pregunta identificada por IDpreg, usada en el instante t, y actualizará los datos del concursante actual y del concurso c adecuadamente:
- *Si la resp dada es un 0, indicará que el concursante ha elegido pasar sin responder:*
  - *Si el número de veces que dicho concursante ha pasado es menor que máximoPasesPermitidos(c), se incrementará en uno el número de veces que el concursante ha elegido pasar, y se pasará el turno al siguiente concursante.*
  - *Si el número de veces que dicho concursante ha pasado no es menor que máximoPasesPermitidos(c), no se modificarán los datos del concursante ni se pasará el turno al siguiente concursante.*
- *Si la resp dada es un 1 o un 2, indicará que el concursante ha elegido como respuesta a la pregunta identificada por IDpreg, respectivamente, la primera o la segunda de las opciones de respuesta disponibles para dicha pregunta:*
  - *Si dicha opción elegida es la respuesta correcta a la pregunta, se incrementará en 1 el número de veces que dicho concursante ha acertado. Si no es la respuesta correcta, se incrementará en 1 el número de veces que dicho concursante ha fallado. Si con este último fallo el concursante ha alcanzado el máximoFallosPermitidos(c), será eliminado del concurso. En cualquier caso, a continuación el turno de preguntas pasa al concursante siguiente al que ha sido puesto a prueba.*
- *Si el concursante no ha elegido pasar sin responder, se actualizará la colección de preguntas del concurso, registrando que el último uso de la pregunta identificada por IDpreg ha ocurrido en el instante t dado.*

Parcial: la operación no está definida si: no existePregunta?(c,IDpreg) (situación: preguntaNoExistente), o si la respuesta recibida resp no es 0≤resp≤2 (situación: respuestaNoVálida), o si no es verdad enJuego?(c)(situación: concursoNoEnJuego), o si es verdad enJuego?(c) y totalConcursantes(c)=0 (situación: sinConcursantes).}

listarPreguntas: concurso c -> cadena
{Devuelve una cadena que contiene la información de la colección de preguntas de c, formada conteniendo consecutivamente las siguientes cadenas:
- *una nueva línea con la cadena "--- LISTADO DE PREGUNTAS ---" , seguida de un salto de línea,*
- *seguida de la cadena "TOTAL preguntas: ", seguida del número resultante de totalPreguntas(c) y seguido de un salto de línea,*
- *seguida del resultado de listar la colección de preguntas de c, y*
- *finalizando con una nueva línea con la cadena "------".}*

ListarConcursantes: concurso c -> cadena
{Devuelve una cadena que contiene la información de la colección de concursantes de c, formada conteniendo consecutivamente las siguientes cadenas:
- *una nueva línea con la cadena "-*- LISTADO DE CONCURSANTES -*-" , seguida de un salto de línea,*
- *seguida de una nueva línea con la cadena "TOTAL concursantes: ", seguida del número resultante de totalConcursantes(c), seguido de un salto de línea,*
- *seguida de una nueva línea con el resultado de listar la colección de concursantes de c, y*
- *finalizando con una nueva línea con la cadena "-*-*-*-".}*

ListarConcurso: concurso c -> cadena
{Devuelve una cadena que contiene la información del concurso c, formada conteniendo consecutivamente las siguientes cadenas:
- *una línea con la cadena "********** ESTADO del CONCURSO **********" , seguido de un salto de línea,*
- *seguida de la cadena "FASE: ", seguida de la cadena "Documentacion", "Inscripcion" o "Juego", adecuada según la fase en la que se encuentre el concurso c, y seguida de un salto de línea,*
- *seguida de la cadena "LIMITES maximos-> Pases: ", seguida del número máximo de pases permitidos a los concursantes, seguido de la cadena " Fallos: ", seguida del número máximo de fallos fijado para el concurso, seguido de la cadena " Ganadores: ", seguida del número máximo de ganadores a seleccionar con el concurso, y seguido de un salto de línea,*
- *si el concurso está en fase de Juego: seguirá una línea con la cadena "JUGANDO concursante: ", seguida del identificador del concursante al que le toca responder a preguntas, o de la cadena "---" si es que no existe tal concursante, seguida de un salto de línea,*
- *seguida del resultado de ListarConcursantes(c), seguido de un salto de línea,*

```
    -   seguida del resultado de listarPreguntas(c), seguido de un salto de línea,
    -   y finalizando con una nueva línea con la cadena "******************************".}

fespec
```

Las operaciones *listarPreguntas* y *ListarConcursantes* deberán implementarse usando las operaciones *listar* de los TADs *colecciónConMarcaTiempo* y *rondaSelección* respectivamente.

3) Utilizar los TADs implementados en las tareas anteriores para implementar un programa de prueba que nos permita probar la implementación del TAD *concursosDePreguntas*, de acuerdo a lo que se describe a continuación.

El código fuente del programa de prueba (*main*) deberá encontrarse en un fichero llamado *"practica2.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 un concurso vacío (sin preguntas, ni concursantes) cuyo máximo de pases sea igual a 2, el máximo de fallos sea 2, y el máximo de concursantes ganadores sea 1. A continuación se leerán las instrucciones de las operaciones a realizar sobre el concurso desde un fichero de texto denominado "***concursoentrada.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 marcadas en negrita: ***ip*** *(introducir pregunta),* ***lp*** *(listar una pregunta),* ***bp*** *(borrar pregunta),* ***lc*** *(listas todas las preguntas),* ***mp*** *(marcar con tiempo una pregunta),* ***ipa*** *(inscribir concursante),* ***bpa*** *(borrar concursante),* ***mpa*** *(mostar concursante),* ***oc*** *(obtener concursante actual),* ***lr*** *(listar concursantes),* ***ii*** *(iniciar inscripción),* ***ij*** *(iniciar juego),* ***pca*** *(probar concursante actual),* ***hg*** *(hay ganadores),* ***lt*** *(listar todos los datos del concurso).*

El programa finalizará cuando haya procesado todas las instrucciones del fichero. Supondremos que el fichero "concursoentrada.txt" tendrá siempre una estructura como la descrita, sin errores.

Si la instrucción es *ip*, las 5 líneas siguientes contendrán los valores de: en la primera línea, la clave con la que se deberá introducir la pregunta en el concurso; en la segunda línea, el texto o interrogante con el que se formula la pregunta; en la tercera línea, el texto correspondiente a su primera opción de respuesta; en la cuarta línea, el texto correspondiente a su segunda opción de respuesta; y en la quinta línea, un número: 1 ó 2, según sea la respuesta correcta la dada como primera opción o la segunda, respectivamente.

Si la instrucción es *mp*, las 2 líneas siguientes contendrán los valores de: en la primera línea, la clave de la pregunta que se desea marcar con un valor de tiempo; en la segunda línea, el valor de tiempo a utilizar, en formato *horas:minutos*.

Si la instrucción es *lp* o *bp*, en la siguiente línea del fichero aparecerá un número entero que será la clave de la pregunta del concurso que se deberá listar o borrar, respectivamente.

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 el concurso; 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 del concurso con el que se quiere utilizar la instrucción.

Si la instrucción es *pca*, las siguientes tres líneas del fichero contendrán: en la primera línea, un número entero que será la clave de la pregunta que se le formula al concursante; en la segunda línea, un tiempo en formato *horas:minutos*; y en la tercera línea, un número con la opción elegida por el concursante como correcta (0, si pasa).

Si la instrucción es *oc*, *dc*, *pt*, *lr*, *lc*, *lt*, *ii*, *ij*, o *hg*, 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).

Obervad que todas las instrucciones, salvo *ii, ij, lt, hg, y pca* han aparecido en las prácticas 0 y 1, anteriores, y tienen la misma descripción en lo referido al formato de las instrucciones del fichero.

Como resultado de su ejecución, el programa deberá ejecutar las instrucciones leídas del fichero "***concursoentrada.txt***", sobre el concurso creado inicialmente, y además creará un fichero de texto "***concursosalida.txt***" que constará de las siguientes líneas:

- Por cada instrucción de tipo *ip*: **si es posible introducir los datos de una nueva pregunta en el concurso**, se escribirá una línea en el fichero de salida que empiece con la cadena "INSERCION REALIZADA: "; **si no es posible introducir los datos de una nueva pregunta en el concurso pero sí que se actualiza una pregunta en el concurso**, se escribirá una línea en el fichero de salida que empiece con la cadena "ACTUALIZACION REALIZADA: "; **si no es posible introducir o actualizar los datos en el concurso**, se escribirá una línea en el fichero de salida que empiece con la cadena "INSERCION DESCARTADA: ". En cualquiera de los tres casos, se seguirá escribiendo en el fichero la concatenación de,

  a.1 la clave de la pregunta en el concurso, seguida de la cadena ":::",

  a.2 seguida del valor de la marca de tiempo de la pregunta, que podrá ser un carácter '-', si no tiene marca de tiempo, o la información de hora y minuto de su marca de tiempo escritos en formato *horas:minutos*

  a.3 seguida de un salto de línea, y a continuación

  a.4 la información de la pregunta según el formato descrito en la tarea 2) de este enunciado.

- Por cada instrucción de tipo *lp*: **si en el concurso se encuentra una pregunta con la clave dada**, se escribirá una línea en el fichero de salida que empiece con la cadena "PREGUNTA ENCONTRADA: ", y seguida de la misma concatenación descrita en los puntos a.1–a.4, con la información de la clave, marca de tiempo y datos de la pregunta en el concurso. **Si en el concurso no se encuentra una pregunta con la clave dada**, se escribirá una línea en el fichero de salida que empiece con la cadena "PREGUNTA NO encontrada: ", seguida de la clave utilizada para la instrucción.

- Por cada instrucción de tipo *mp*: **si en el concurso se encuentra una pregunta con la clave dada**, se escribirá una línea en el fichero de salida que empiece con la cadena "PREGUNTA MARCADA: ", seguida de la clave utilizada, seguida de un carácter ';', seguido del tiempo utilizado en formato horas:minutos, y seguido de un salto de línea. **Si en el concurso no se encuentra una pregunta con la clave dada**, se escribirá una línea en el fichero de salida que empiece con la cadena "MARCA DESCARTADA: ", seguida de la clave utilizada para la instrucción.

- Por cada instrucción de tipo *bp*: **si en el concurso se encuentra una pregunta con la clave dada**, se escribirá una línea en el fichero de salida que empiece con la cadena "PREGUNTA BORRADA: ", seguida de la clave utilizada, seguida de un salto de línea. **Si en el concurso no se encuentra una pregunta con la clave dada**, se escribirá una línea en el fichero de salida que empiece con la cadena "BORRADO DESCARTADO: ", seguida de la clave utilizada para la instrucción. **Si no es posible borrar preguntas debido al estado del concurso**, se escribirá una línea "BORRADO pregunta NO POSIBLE: ", seguida de la clave utilizada para la instrucción.

- Por cada instrucción de tipo *lc*, se escribirá en el fichero la cadena devuelta por la operación *listarPreguntas* descrita en la especificación del TAD *concursosDePreguntas*.

- Por cada instrucción de tipo *ipa*, **si es posible** añadir el participante en el concurso, se escribirá una línea en el fichero de salida que empiece con la cadena: "participante INSCRITO: ", si en el concurso no existía previamente un participante inscrito con el *alias* dado, o con la cadena "participante ACTUALIZADO: ", si en el concurso 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:

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

  b.2 seguido por el carácter ';'

  b.3 seguido de una cadena que contenga la información del participante con el formato que se ha descrito en la tarea 2 del enunciado de la práctica anterior (práctica 1).

  Si no se puede realizar la inscripción, debido a que la operación no está permitida para el estado actual del concurso, 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 b.1-b.3.

- Por cada instrucción de tipo *bpa*, **si es posible borrar** el participante del concurso, 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 el concurso, 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 del concurso, 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 en el concurso, 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 b.1-b.3.

  Si no se puede encontrar el participante, debido a que el participante no se encuentra inscrito en el concurso, 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 *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 b.1-b.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 del concurso, se escribirá una línea en el fichero de salida que empiece con la cadena "CONSULTA candidato DESCARTADA".

- Por cada instrucción de tipo *lr*, se escribirá en el fichero la cadena devuelta por la operación *listarConcursantes* descrita en la especificación del TAD *concursosDePreguntas*.

- Por cada instrucción de tipo *ii*, **si es posible pasar a la fase de inscripción** de concursantes, se escribirá una línea de texto en el fichero de salida con el mensaje "INSCRIPCION ABIERTA". **Si no es posible** debido a la situación actual del concurso, se escribirá "INSCRIPCION CANCELADA".

- Por cada instrucción de tipo *ij***, si es posible pasar a la fase de juego** del concurso se escribirá una línea de texto en el fichero de salida con el mensaje "INICIANDO JUEGO: " seguido del total de participantes en el concurso. **Si no es posible**, se escribirá "NO ES POSIBLE iniciar JUEGO".

- Por cada instrucción de tipo *pca*, si es **posible que el concursante actual responda a una pregunta**, se escribirá la concatenación de:

  c.1 la cadena "+++ RESPONDE CONCURSANTE +++", seguida de un salto de línea;

  c.2 la cadena "CONCURSANTE: ", seguida de la información del concursante con el formato explicado en b.1-b.3, seguida de un salto de línea;

  c.3 la cadena "PREGUNTA: ", seguida de la información de la pregunta en el formato explicado en la tarea 2) de este enunciado, seguida de un salto de línea;

  c.4 la cadena "RESPUESTA DADA: ", seguida de un 0, 1 o 2, con la respuesta dada por el concursante, seguido de la cadena " TIEMPO: ", seguida del tiempo en el que se ha dado la respuesta en formato *horas:minutos*, seguido de un salto de línea;

  c.5 si tras responder el concursante ha sido eliminado, se añadirá una línea más con "CONCURSANTE ELIMINADO" , finalizada con un salto de línea;

  c.6 y finalmente la cadena "++++++++++", seguida de salto de línea.

  Si no es posible probar al concursante actual debido a que: no se está en la fase del concurso adecuada, se escribirá "PROBAR concursante DESCARTADO"; si estamos en fase de juego y no hay concursantes inscritos, se escribirá, "ronda VACIA"; si no se han dado los dos casos anteriores y la clave de la pregunta leída del fichero no está en el concurso, se escribirá "PREGUNTA no encontrada: ", seguida de la clave leída; por último, si no se han dado los tres casos anterior y la respuesta dada no es igual a 0, 1 o 2, se escribirá "RESPUESTA NO VALIDA".

- Por cada instrucción *hg*, si la operación *hayGanadores?* del concurso indica que **hay ganadores**, se escribirá un mensaje compuesto de la cadena "<<< CONCURSANTES GANADORES >>>", seguida de un salto de línea y del listado de los concursantes descrito en la operación *listarConcursantes*. **Si no hay ganadores**, se escribirá "SIN GANADORES".

- Por cada instrucción de tipo *lt*, se escribirá en el fichero la cadena devuelta por la operación *listarConcurso* descrita en la especificación del TAD *concursosDePreguntas*

Acompañando a este enunciado se aportan, a modo de ejemplo, un fichero "*concursoentrada.txt*" y su correspondiente fichero "*concursosalida.txt*".

Consideraciones sobre la implementación en C++

En la implementación de la práctica 2, os podéis encontrar que el compilador señala un error indicando algo similar a:

```
note:   template argument deduction/substitution failed
note:   couldn't deduce template parameter 'K' ...
  ...   = nombre_funcion(...sus parámetros);
```

En este caso, el compilador nos está diciendo que al invocar la función genérica "nombre_funcion" (no al declararla, sino al usarla) no puede deducir de qué tipo es el parámetro formal K: K ¿es un string?, ¿un entero? Para solucionar este tipo de errores se debe invocar a la función de la siguiente forma:

```
.... = nombre_funcion<K,V>(...sus parámetros)
```

Esto es, poniendo entre el nombre de la función y la lista de parámetros un "<K,V>", que ayude al compilador a recordar que los parámetros formales son K y V.

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 *"practica2.cpp"*, y cumplir escrupulosamente con el funcionamiento y formatos que se han descrito en el enunciado.
- La ruta del fichero de entrada deberá ser "concursoentrada.txt" (no "concursoentrada1.txt", "entrada.txt", "datos/concursoentrada.txt", ni similares). Ídem para el fichero "concursosalida.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_p2, si tu profesor tutor es Jorge Bernad, o Y_p2, si tu profesora tutora es Yolanda Villate, donde se guardará un directorio *practica2* que contenga todos los ficheros desarrollados para resolver la práctica (este directorio, *practica2*, deberá contener todos los ficheros con código fuente C++ necesarios para resolver la práctica y los ficheros de texto ***concursoentrada.txt*** y ***concursosalida.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_p2.tar, con X igual a J o Y, dependiendo de quién sea tu profesor tutor, con el contenido del directorio X_p2 ejecutando el comando:
  `tar –cvf X_p2.tar X_p2`
- Enviar el fichero X _p2.tar, con X igual a J o Y, dependiendo de quién sea tu profesor tutor, mediante la orden:
  `someter –v eda_17 X_p2.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.**