| | | |
|---|---|---|
| Universidad de Costa Rica<br>Escuela de Computación | **Examen único**<br>10-Jul-2014 | CI-1201 Programación II - 2014a<br>Profesor Jeisson Hidalgo-Céspedes |

*En cada ejercicio se evaluará la eficiencia del código y las buenas prácticas de programación como: el uso de identificadores significativos, la indentación, escritura correcta de llaves {} y el uso adecuado de la palabra reservada const. Se dispone de tres horas para entregar la prueba y debe realizarse en forma estrictamente individual.*

Una agencia de eventos competitivos organiza frecuentemente torneos o campeonatos de diversa índole. Por ejemplo, campeonatos de ajedrez, fútbol, "yaxis", contar chistes, música, baile, y hasta campeonatos de programación. En cada uno de ellos se sigue el mismo procedimiento: organizar los competidores en grupos, clasificar la mitad con mejor rendimiento de cada grupo, entrelazar un grupo con el siguiente (grupo A vs grupo B, C vs D), y realizar la eliminatoria confrontando a los que van ganando en las fases: cuartos de final, semifinal, tercer puesto y final.

El trabajo es repetitivo y se hace manualmente, a veces bajo presión durante el mismo evento. En algunas ocasiones se cometen errores en el diseño o surgen inconformismos de los competidores, por ejemplo, argumentando que los grupos se conformaron con alevosía o falta de equidad. Por esto la agencia consideró que sería mejor que una computadora realice el diseño del campeonato conformando aleatoriamente los competidores en grupos, permita registrar los resultados de los encuentros, e imprimir el estado del campeonato.

Dada que la naturaleza de los competidores puede variar mucho de una competencia a otra (jugadores de ajedrez, equipos de fútbol, parejas de baile, equipos de programación), y el sistema de puntuación (gane/empate/pérdida en ajedrez, goles en fútbol, votación promedio del jurado en baile, puntaje en programación), usted decide por programar una clase `Campeonato` -si programa en inglés, *Championship*- genérica en C++. Usted debe escoger el mecanismo para proveer esta generalidad, sea usando plantillas o polimorfismo.

1. [5%] **Competidores**. Debe almacenar los competidores en alguna estructura de datos que pueda crecer dinámicamente. Si gusta, puede utilizar clases contenedoras de la biblioteca estándar de C++.
2. [5%] **Agregar competidores**. Implemente algún mecanismo que permita agregar competidores al `Campeonato`, por ejemplo un método `agregarCompetidor()` -en inglés, *addCompetitor()*-, o el operador `<<`.
3. [10%] **Grupos de competidores**. Diseñe un mecanismo para indicar quienes van a competir en el campeonato. Por ejemplo, implementar un método `establecerCantidadGrupos()` -en inglés *setGroupCount()*- que le indica a la clase la cantidad de grupos que se quieren formar. Implemente un método `hacerGrupos()` -*makeGroups()*- que distribuya aleatoriamente los competidores en los grupos indicados por el usuario de la clase. Este método no debe cambiar la semilla del generador de números aleatorios. Si lo necesita puede utilizar el algoritmo `std::random_shuffle(itr1, itr2)`. Usted puede imponer las restricciones que necesite, por ejemplo, la cantidad de competidores debe ser múltiplo de la cantidad de grupos, y en caso del usuario no cumplirlas, debe indicarlo de alguna forma, como retornar un booleano o lanzar una excepción.
4. [5%] **Imprimir los grupos**. Los competidores estarán ansiosos por saber en qué grupo quedaron y con quienes tienen que enfrentarse primero. Implemente un método `imprimirGrupos()` que reciba por parámetro el archivo `ostream` donde se quiere imprimir la lista de grupos con sus competidores.
5. [10%] **Registro de encuentros**. Su clase `Campeonato` debe mantener registro de todos los encuentros celebrados y por celebrarse, además del marcador final en aquellos celebrados. Implemente una clase anidada `Encuentro` -en inglés *Match*- que relaciona dos competidores (apuntadores) y el puntaje (*score*) que obtuvo cada uno en dicho encuentro. Asuma que este puntaje es siempre entero. Un encuentro debe poderse imprimir en un archivo que recibe por parámetro. Un encuentro debe ser capaz de saber si se ha celebrado o no.
6. [10%] **Actualizar encuentros**. Implemente un contenedor de `Encuentro`s en su clase `Campeonato`, y un método `actualizar()` que se encarga de llenar este contenedor con todos los encuentros conocidos hasta el momento. Los encuentros dependen del estado del campeonato y La clase `Campeonato` debe conocer este estado: no iniciado, iniciado, eliminatoria, semifinal, final. Por ejemplo, al llamar el método `actualizar()` cuando el campeonato está en estado *no iniciado*, provocará que se agreguen al contenedor de encuentros las permutaciones de los equipos de cada grupo. Una vez actualizado el campeonato, pasará al estado *iniciado*.

7. [5%] **Imprimir encuentros**. Implemente un método que imprima todos los encuentros conocidos con sus respectivos equipos y puntajes, se hayan efectuado o no.
8. [15%] **Iterador de encuentros**. Implemente una clase especial `Iterador` -en inglés *Iterator* o *MatchIterator*- que permite recorrer los encuentros conocidos del campeonato. Un iterador debe ser capaz de obtener una referencia al `Encuentro` que "apunta" con el operador asterisco (`*`), pasar al próximo encuentro con el operador de preincremento (`++`), comparar si dos iteradores representan el mismo `Encuentro` con los operadores `==` y `!=`. Implemente en el iterador un método `asignarResultado(a,b)` -en inglés `setScores(a,b)` que recibe dos enteros, uno para el marcador del primer equipo y el otro para el marcador del segundo equipo. Implemente un método `inventarResultado(max)` que asigna un marcador aleatorio a cada equipo.
9. [5%] **Permitir iterar**. Implemente en su clase `Campeonato` un método `primerEncuentro()` que retorna un iterador al primer encuentro, un método `ultimoEncuentro()` que retorna un iterador a un encuentro no válido, y un método `primerEncuentroPendiente()`, que retorna un iterador al primer encuentro que aún no se ha celebrado.
10. [15%] **Pasar a eliminatoria**. Modifique su método `actualizar()` para que cuando se encuentre en estado *iniciado* revise todos los encuentros pendientes utilizando iteradores. Si todos los encuentros han sido efectuados, pasa al estado *eliminatoria*. Ordena cada grupo por los competidores con mejor rendimiento. Aquí debe imponer restricciones sobre la clase `Competidor`. Si es una plantilla documente qué debe implementar (por ejemplo, el operador `<`, si es una clase base polimórfica, implemente algún método virtual puro). Haga que la mitad de competidores de cada grupo pase a la eliminatoria, asignándoles nuevos encuentros (los cuales se agregan al contenedor de encuentros), de tal forma que se mezclen los grupos en orden: el mejor del grupo A contra el mejor del grupo B, el segundo mejor del grupo A contra el segundo mejor del grupo B; luego los mejores del grupo C contra los del grupo D, y así sucesivamente.
11. [15%] **Utilice su clase**. Provea un ejemplo de cómo se usa su clase implementando un campeonato de la disciplina que guste (fútbol, piques de autos, programando por un sueño, etc.). Debe implementar al menos una clase que representa a cada competidor y la función `main()`. Agregue competidores ficticios, invente marcadores hasta iniciar la eliminatoria.
12. [10% opcional] Implemente su solución en código y preséntelo en la plataforma educativa.