Trabajo práctico Labo: Implementacin en C++
"Cartas Enlazadas"

Normativa

Límite de entrega: Mircoles 9 de septiembre de 2015 a las 21:00 hs.
Normativa completa: Para más detalles véase "Información sobre la cursada" en el sitio Web de la materia. (`http://www.dc.uba.ar/materias/aed2/2015/2c/informacion`)

1. Normas de entrega adicionales

- **Este TP se realiza en forma individual.** No es necesario informar el grupo.
- **La entrega es en forma electrónica únicamente**. Usar el siguiente enlace: `http://tinyurl.com/qeps9ay`

2. Enunciado

En el juego de mesa de Cartas Enlazadas los jugadores se sientan en una mesa y van sacando de alguno de los dos mazos de cartas (Rojo y Azul), una carta a la vez. El juego de mesa Cartas Enlazadas es uno de los juegos de cartas más complicado que existe, comparable con el más complicado de todos[1], por ende estamos desarrollando un pequeño sistema para poder seguir el puntaje y determinar quien gana. Las reglas descriptas aquí se refieren a la edición simplificada para Algoritmos y Estructuras de datos II.

Los jugadores se sientan de a uno por vez en la mesa y el primero que se le otorgan los dos mazos de cartas. Los jugadores que se agregan luego lo hacen siempre al lado del jugador que tiene el mazo Azul. Las diferentes cartas de los mazos van indicando las acciones que los jugadores tienen que hacer.

Uno de los jugadores que tiene un mazo debe revelar la carta que está en el tope del mismo. La carta revelada debe resolverse antes de proseguir. Las posibles acciones que se deben realizar son:

- Adelantar el mazo n jugadores.
- Eliminar el o los jugadores que cumplen cierta condición.
- Sumar k puntos al jugador que tiene el mazo.

El juego termina cuando se acaba el mazo y el ganador es el jugador que haya acumulado más puntos y esté sentado en la mesa. Debido a la característicasdel mazo nunca se puede dar un empate.

Se pide:

1. Implementar en C++ la clase paramétrica `CartasEnlazadas<T>`, cuya interfaz se provee en el archivo `.h` adjunto, junto con la implementación de todos los métodos públicos que en ella aparecen.
   No pueden agregar nada público. Sí pueden, y deben, agregar cosas privadas, en particular los campos que les parezcan pertinentes. También pueden agregar funciones auxiliares, tanto de instancia como estáticas, clases auxiliares, etc., pero nada en la parte pública de la clase.
2. Implementar las funciones de test no implementadas en `tests.cpp`. El correcto funcionamiento de los test **no es garantía** de aprobación.
3. La implementación dada no debe perder memoria en ningún caso. Al momento de la corrección se hará el chequeo pertinente usando *valgrind*.
4. Se sugiere chequear las precondiciones con **assert** para facilitar la depuración de errores.
5. Está prohibido utilizar la biblioteca estándar de C++ (STL) con la excepción de las siguientes librerías: **iostream**, **string** y **cassert**.
6. Se sugiere repasar la sección 10.2 del Cormen.

3. Recomendaciones

El objetivo de este trabajo práctico es familiarizarse con el lenguaje C++ y las características del mismo que se usarán en esta materia (templates, memoria dinámica, etc.), de manera de llegar mejor preparados a afrontar un desarrollo más grande y complicado como el TP3.

[1] `https://www.youtube.com/watch?v=0oOD9U9VQ5Y`

Respecto de los archivos provistos, si intentan compilar `tests.cpp` podrán hacerlo, pero no podrán linkearlo y generar un binario porque, por supuesto, va a faltar la implementación de todos los métodos de la clase testeada.

Una sugerencia para empezar es dejar la implementación de todos los métodos necesarios escrita, pero vacía, de manera de poder compilar. Pueden comentar todos los tests que requieran métodos aún no implementados de manera de poder usar la aplicación para el testing a medida que van implementando. Tengan en cuenta que algunos métodos pueden ser necesarios para muchas de las funciones de test, por lo tanto, es aconsejable empezar por esos test.

Además de los casos de test provistos por la cátedra les recomendamos fuertemente que realicen sus propios tests.

Dado que su implementación no debe perder memoria, es una buena práctica utilizar la herramienta *valgrind* durante el desarrollo, como mínimo en la parte de testing final.