Trabajo práctico Labo: Implementación en C++
"Arturo"

Normativa

Límite de entrega: Viernes 19 de Septiembre de 2014 a las 20:00 hs.
Normas de entrega: Ver "Información sobre la cursada" en el sitio Web de la materia.
(`http://www.dc.uba.ar/materias/aed2/2014/1c/informacion`)

1. Normas de entrega adicionales

- **Este TP se realiza de a dos.** No es necesario informar el grupo.
- **La entrega es presencial**; si alguien no puede asistir, con justificativo debe comunicarse con los JTPs.
- El día de la entrega deberán disponer de una copia del código que pueda ejecutarse en los laboratorios.
- Para aprobar, además del correcto funcionamiento del TP deben estar presentes ambos integrantes y responder las preguntas del corrector.

2. Enunciado

El monarca ideal, también conocido como Rey Arturo, fue un caudillo británico que dirigió la defensa de Gran Bretaña contra los invasores sajones a comienzos del siglo VI. Arturo fue un personaje de la literatura, pero se ha planteado que pudo haber sido una persona real.

Arturo reinaba en Camelot, reino en el cual cualquier asunto relacionado con la seguridad de su reino, le gustaba discutirlo con sus caballeros en una mesa redonda.

En búsqueda de expandir su reinado, Arturo trae a la mesa un tema muy complejo y quiere (como siempre) tener el absoluto control de la discusión. Para ello, quiere tomar algunas acciones en particular.

Cuando Arturo lo cree conveniente puede elegir sumar un caballero a la mesa para incrementar la discusión entre los presentes. Si hay algún caballero que no esta siguiendo las ideas brillantes e inobjetables de del Rey, puede elegir expulsarlo de la mesa.

Cuando Arturo decide que el caballero que esta hablando actualmente, ya dijo suficiente, puede hacer que hable el siguiente. Puede pasar que el Rey tenga compasión de un caballero al que le acaba de sacar el turno y elija devolvérselo. También puede pasar que Arturo quiera interrumpir al actual orador para hablar el. Si esto ocurre, el próximo orador es el siguiente del interrumpido.

Cuando hay una discusión entre 2 caballeros y el rey quiere parar la pelea sin expulsarlos, puede elegir levantarse de su silla e ir a sentarse en el medio de los caballeros que se están peleando.

Se pide:

1. Implementar en C++ la clase paramétrica[1] `Arturo<T>`, cuya interfase 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*.

[1] Por ejemplo, Arturo podría decidir tener no solo una mesa redonda, sino varias.

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.

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.