Ejercitación – Punteros

Se recomienda resolver los ejercicios en orden. En CLion se encuentran disponibles los siguientes targets:

- ejN para $1 \le N \le 5$, ejecuta los tests.
- ejN_correrValgrind para $N \in \{3, 4, 5\}$, ejecuta el test indicado con la herramienta valgrind, verificando que no haya problemas con el manejo de la memoria dinámica.

Los targets también pueden compilarse y ejecutarse sin usar CLion. Para ello:

- 1. En una consola pararse en el directorio raíz del projecto. En este debería haber un archivo CMakeLists.txt.
- 2. Ejecutar el comando \$ cmake . (incluyendo el punto). Esto generará el archivo Makefile.
- 3. Ejecutar el comando \$ make TARGET donde TARGET es uno de los targets mencionados anteriormente. Esto creará un ejecutable con el nombre del target en el directorio actual.
- 4. Ejecutar el comando \$./TARGET siendo TARGET el nombre del target utilizado anteriormente. Esto correrá el ejecutable.

La herramienta valgrind también puede ejecutarse manualmente, con el siguiente comando: valgrind --leak-check=full ./TARGET

Sistema de Mensajes

Queremos implementar un sistema de mensajes para jugadores de un juego. El juego en cuestión puede jugarse con entre 1 y 4 jugadores. El sistema de mensajes debe poder crear una conexión para cada uno de los jugadores. Además debe poder permitir enviar un mensaje a un jugador individual o a todos.

Los jugadores están modelados en la clase ConexionJugador y almacenan una dirección IP a la cual enviar los mensajes. Además poseen el método ConexionJugador::enviarMensaje(string mensaje); que envía el mensaje al IP con el que fue creada la instancia. Para más información revisar ConexionJugador.h

Ejercicio 1

Dada la clase SistemaDeMensajes definida en SistemaDeMensajes.h, completar la definición de las siguientes funciones en SistemaDeMensajes.cpp:

SistemaDeMensajes::SistemaDeMensajes();

```
• void SistemaDeMensajes::registrarJugador(int id, string ip);
```

- void SistemaDeMensajes::enviarMensaje(int id, string mensaje);
- bool SistemaDeMensajes::registrado(int id);

El sistema de mensajes empieza sin tener ningún jugador registrado. La función enviarMensaje requiere que esté registrado el jugador del identificador parámetro.

Revisar la definición de la clase SistemaDeMensajes. Notar que por cuestiones de eficiencia, los jugadores se guardan como un arreglo de ConexionJugador*

Ejercicio 2

Correr valgrind sobre el ejecutable ej1. Si no implementaron el destructor de la clase SistemaDeMensajes::~SistemaDeMensajes(); es esperable que estén perdiendo memoria.

De ser necesario, implementar el destructor.

Correr valgrind sobre el ejercicio 1 y asegurarse no devuelva errores.

Ejercicio 3

Queremos que el sistema de mensajes permita des-registrar un ip para un jugador existente. Esto implica eliminar la instancia de ConexionJugador. Además, queremos que se pueda registrar un ip nuevo para un ip ya registrado. Para ello es necesario crear una instancia nueva de ConexionJugador de forma de sobreescribir la original.

para que si el jugador ya está registrado, lo actualice con el ip parámetro.

Implementar void SistemaDeMensajes::desregistrarJugador(int) y modificar void SistemaDeMensajes::regist:

Correr los tests del ejercicio y luego correr valgrind para asegurarse de no estar pediendo memoria o tener otros errores.

Proxy

Para ser aún más eficientes, vamos a hacer que quien requiera mandar un mensaje a un jugador tenga un acceso directo al mismo; sin pasar por el sistema de mensajes. Para ello, el usuario, deberá pedir al sistema de mensajes el Proxy del id de usuario y será la clase Proxy la que permita mandar el mensaje.

La clase Proxy se encuentra (parcialmente) implementada en Proxy.h y Proxy.cpp.

Notar que la misma guarda internamente un puntero a ConexionJugador.

Ejercicio 4

Agregar en Sistema De Mensajes. h el método Proxy* Sistema De Mensajes::obtener Proxy(int id); que dado un id de jugador cree un proxy para este jugador y devuelva un puntero al mismo.

Ejercicio 5

En la implementación simple del ejercicio anterior, se están realizando new de los Proxy sin que se realicen sus delete, lo que implica que se pierde memoria.

Correr valgrind sobre los tests del ejercicio 4 y ver que se está perdiendo memoria.

Para corregir esto, agregar en la parte privada de SistemaDeMensajes un vector<*Proxy> donde se deben almacenar todos los punteros de Proxy creados. Luego, en el destructor de la clase, eliminar todos los punteros.

Esto asume que todo el código que usa algún proxy no lo va a seguir usando luego de que se termine la vida del SistemaDeMensajes.

Ejercicio 6

En la implementación sencilla del ejercicio 4, también se perdió soporte a cambiar la conexión del jugador. Si el SistemaDeMensajes modifica el puntero a ConexionJugador, los Proxy no se enteran del cambio y se quedan con un puntero inválido.

Nuestra propuesta para resolver el problema es modificar Proxy de forma que pase a tener como miembro interno un ConexionJugador** _conn. Esto es un puntero al puntero que lleva a la ConexionJugador.

Completar la nueva implementación de Proxy en Proxy2.cpp y SistemaDeMensajes.cpp. Ver Proxy2.h para referencia.

Ejercicio 7 (Bonus)

Reorganizar el código de forma que Proxy sea una clase miembro de SistemaDeMensajes. Esto significa que los archivos Proxy2.h y Proxy2.cpp deberían dejar de existir. Además, SistemaDeMensajes pasaría a tener la siguiente pinta:

```
// En SistemaDeMensajes.h
class SistemaDeMensajes {
   public:
        class Proxy; // Forward declaration
        /* Interfaz SistemaDeMensajes */
        class Proxy {
```

```
public:
            /* Interfaz Proxy */
            private:
            /* Parte privada de Proxy */
        };
    private:
    /* Parte privada de SistemaDeMensajes */
};
// En SistemaDeMensajes.cpp
SistemaDeMensajes::SistemaDeMensajes() {
    // Cuerpo método
}
void SistemaDeMensajes::registrado(int id) {
    // Cuerpo método
}
/* Más métodos SistemaDeMensajes ... */
SistemaDeMensajes::Proxy::Proxy(ConexionJugador* conn) : _conn(conn) {
}
SistemaDeMensajes::Proxy::enviarMensaje(string msg) {
    // Cuerpo método
}
/* Posiblemente más métodos SistemaDeMensajes::Proxy ... */
```