1 Seminarios C y C++

Los requerimientos de cada ejercicio del seminario serán expuestos desde el punto de vista práctico y teórico; es decir, para su exposición, cada equipo se basará en el caso práctico en cuestión para introducir y explicar el elemento teórico requerido. La exposición no es una mera enunciación de código. Preguntas como: *¿Por qué?*, *¿Basándose en qué?*, *¿Cómo se logra esto en el lenguaje* `X`*?* entre otras, deben hacerse.

Todos los miembros del equipo deben participar en la solución del ejercicio y estar preparados para exponer todo el trabajo. **La persona a exponer** se decide el día de la exposición. Quién no esté presente en la exposición de su equipo tiene `0` en la evaluación. (Note que estas notas se promedian y hay distinción entre 0 y 2).

1.1 Seminario 2 (C++11, C++14):

Implemente una clase `linked_list` doblemente enlazada en `C++` (haciendo uso extensivo de los elementos novedosos en el lenguaje desde `C++11`) que cumpla los siguientes requerimientos:

1. Definir las clases genéricas `linked_list` y `node`.
   a. No es necesario una exposición, ya el **Equipo 1** tocó este punto.
2. Definir miembros de datos necesarios de ambas clases.
   a. ¿Cuáles son los nuevos elementos introducidos a partir de `C++11` que permiten un manejo más *"inteligente"* de la memoria?
   b. ¿Cómo deben inicializarse?
   c. ¿Cuál es la filosofía en el uso de la memoria defendida por `C++`?
   d. Usar *alias* para simplificar nombres de tipos.
3. Definir los constructores clásicos de `C++` (`C++0x`, el constructor `move` y las sobrecargas del operador `=`.
   a. ¿Qué hace cada uno de ellos? ¿Cuándo se llaman?
   b. ¿Qué es un `lvalue` y un `rvalue`?
   c. Explique `std::move`.
4. Definir un constructor que permita hacer *list-initialization* lo más parecido a `C#` posible.
   a. Compare la utilización del `{}` v.s `()`.
5. Definir un constructor que reciba un `vector<T>`.
   a. Usar `for_each` con *expresiones lambda*.
6. Definir el destructor de la clase.
   a. ¿Hace falta?
   b. ¿Para qué casos haría falta un puntero crudo (*raw pointer*)?

7. Definir funciones `length`, `Add_Last`, `Remove_Last`, `At`, `Remove_At`
   a. Explique `Noexcept`.
   b. Inferencia de tipo en `C++` (`auto`, `decltype`, `decltype(auto)`). Explicar todos, pero no obligatoriamente usarlos.
8. Crear un puntero a función `Function<R,T...>` que devuelve un valor de tipo `R` y recibe un número variable de parámetros de tipo `T`.
   a. Definir una función genérica `Map` a `linked_list` en `T` y `R`, que recibe un puntero a función que transforma un elemento `T` en uno `R`; de manera que `Map` devuelve una instancia de `linked_list<R>` resultado de aplicar a todos los elementos `T` de la lista original la función de transformación.
   b. Crear punteros a funciones usando *alias*.
   c. Crear un puntero a función `Function` que permita cualquier cantidad de parámetros de cualquier tipo.