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 1 (`C++98`, `C++0x`)

Implemente una clase `linked_list` doblemente enlazada en `C++` (con los elementos disponibles en el lenguaje hasta antes de `C++11` (`C++0x`)) que cumpla los siguientes requerimientos:

1. Definir las clases genéricas `linked_list` y `node`.
   a. Introducir lo que es un `template` en `C++` enfocado a la genericidad y cómo funciona de manera abreviada.
2. Definir miembros de datos necesarios de ambas clases.
   a. ¿Qué significan *por valor*, *por puntero* y *por referencia* en `C++`? ¿Cómo funciona esto en memoria?
   b. ¿Cuál es la filosofía en el uso de la memoria defendida por `C++`?
   c. Usar `typedef` para simplificar nombres de tipos.
3. Definir constructores básicos de `C++` y el operador `=`.
   a. ¿Qué hace cada uno de ellos? ¿Cuándo se llaman?
   b. Explicar la inicialización de campos.
   c. ¿Se puede hacer *list-initialization* al estilo `C#`?
   d. ¿Cómo funciona el paso de parámetros cuando se llama a una función?
   e. ¿Cuándo se deben utilizar parámetros *por valor*, *por puntero* o *por referencia*?
   f. Constructores con un solo argumento.
   g. Constructores `explicit`.
4. Definir un constructor que reciba un `vector<T>`.
   a. Utilizar el iterador de `vector<T>` para recorrerlo.
   b. ¿Se puede hacer *list-initialization* al estilo `C#`?
   c. ¿Cómo funciona el paso de parámetros cuando se llama a una función?
   d. ¿Cuándo se deben utilizar parámetros *por valor*, *por puntero* o *por referencia*?

   e. Constructores con un solo argumento.
   f. Constructores `explicit`.
5. Definir el *destructor* de la clase
   a. ¿Qué es un *destructor*?
   b. ¿Cuándo se debe definir el *destructor* como virtual?
   c. Explicar los operadores `delete` y `delete[]`
6. Definir funciones `length`, `Add_Last`, `Remove_Last`, `At`, `Remove_At`.
   a. Funciones `inline` v.s Macros de `C`.
   b. ¿Cuándo usar funciones `inline`?
   c. ¿Cómo se comportan las variables por valor, punteros y las referencias como retorno de una función?
   d. Explicar las funciones `const`.
   e. ¿Cómo se capturan y lanzan las excepciones?
7. Definir el operador `[]` para `linked_list`. Definir el operador `()` con parámetros `(int start, int count)` para `linked_list`, el cual crea una nueva instancia de la clase que contiene una copia de los count elementos consecutivos empezando en start.
   a. Sobrecargar operador `=` de `node` para que pueda recibir un elemento de tipo `T`.
   b. Limitaciones del operador `[]`.
   c. Usar una función como `lvalue` (`list->At(3) = 6;`) "`C++` Return by Reference".
   d. Cómo debe ser el tipo de retorno del operador `()`?
8. Crear un puntero a función `Function<R,T>` que devuelva un valor de tipo `R` y recibe un parámetro de tipo `T`.
   a. Definir una función genérica `Map` en `linked_list` en R, que reciba un puntero a función que transforma un elemento `T` (parámetro genérico del tipo) en uno `R` (parámetro genérico del método); 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 `typedef`.
   c. ¿Se puede crear un `Function<R, T...>` con un número variable de `T`s?
9. Sobrecargar operadores `<<` y `>>` para poder utilizar la clase con la salida estándar.
   a. Uso de la palabra reservada `friend`.