Homework 1

I102 – Paradigmas de Programación

Fecha límite de entrega: 21/03/2025 – 22hs

La entrega se realizará mediante un archivo de texto que contenga los enlaces a los repositorios de GitHub con la solución de cada ejercicio, organizados en carpetas separadas. Además, el archivo debe incluir un texto que indique cómo compilar el código de cada ejercicio. Cada carpeta de los ejercicios debe contener únicamente archivos sin compilar, como .h, .hpp, .cpp, entre otros (también se puede incluir un archivo .gitignore).

1. a. Crear una función que dado un valor entero "n", positivo y mayor a uno, devuelva una matriz cuadrada con valores como en los siguientes ejemplos para n=2 y n=3:

$$M_2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad M_3 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

b. Proveer un código que imprima el contenido de la matriz, utilizando un único ciclo for, desde el mayor valor al menor, es decir, para el ejemplo de Para M_2:

$$M_2[1][1] = 4$$
$$M_2[1][0] = 3$$
$$M_2[0][1] = 2$$
$$M_2[0][0] = 1$$

Nota: recuerde que se deben imprimir los índices de la matriz.

2. En muchos sistemas, es importante registrar todo lo que sucede mientras están en funcionamiento. Para ello, se utiliza un sistema de log que almacena los eventos relevantes. Cada evento recibe una etiqueta que indica su nivel de importancia o gravedad. Las etiquetas más comunes son: DEBUG, INFO, WARNING, ERROR y CRITICAL.

a. En este ejercicio, se pide crear un sistema log que permite agregar entradas a un archivo mediante el llamado a una función *logMessage* definida en pseudo código de la siguiente manera:

void logMessage(String mensaje, Integer/Otro NivelSeveridad)

Donde *NivelSeveridad* corresponderá con unas de las leyendas previamente mencionadas. El formato esperado en una línea del archivo de log es el siguiente:

[ERROR] <Mensaje>
[INFO] <Mensaje>
etc.

Verifique su funcionamiento con al menos una entrada de cada tipo.

b. En un proyecto usualmente se solicitan cambios para mejorar o agregar funcionalidad. Para el caso del código del ejercicio 2.a, se requiere tener la habilidad de agregar mensajes personalizados para registrar otro tipo de eventos. Los requisitos son los siguientes:

i. Todos los nuevos mensajes deben ser invocados con *logMessage.*
ii. Se requiere la posibilidad de registrar errores, indicando el mensaje de error, el archivo y la línea de código donde sucedió este error, es decir:

logMessage(String Mensage_de_Error, String Archivo, Int Línea_de_Código)

iii. Se requiere la posibilidad de registrar un mensaje de "Acceso de Usuario" a la aplicación. Este mensaje debe tener una leyenda nueva: [SECURITY]. La misma debe ser ingresada de la siguiente manera:

logMessage(String Mensaje_De_Acceso, String Nombre_de_Usuario)

Los mensajes de acceso pueden ser: *Access Granted, Access Denied*, etc.

iv. Se requiere un código que pruebe que el sistema verifica la funcionalidad requerida y que además demuestre que puede capturar un error en runtime, crear una entrada en el log y después detener la ejecución del programa y salir del mismo con un código de error (return 1).

3. Implemente una lista enlazada que utilice nodos que simplemente contengan un valor y una dirección de memoria de un nodo. Adicionalmente, agregue las siguientes funciones para manejar la lista:

i. create_node(): devuelve un nodo.
ii. push_front(): inserta un nodo al frente de la lista.
iii. push_back(): inserta un nodo al final de la lista.
iv. insert(): inserta un nodo en la posición que se le pase a la función. Si se le pasa una posición mayor al largo de la lista, se debe indicar lo ocurrido y se debe de agregar el nodo al final de la lista.
v. erase(): borra un nodo en la posición que se le pase a la función. Similar a la función insert(), si la posición es mayor que el largo de la lista, se debe de borrar el último nodo.
vi. print_list(): imprime la lista completa, separando el valor en cada nodo con "->".

Presentar ejemplos que verifiquen el funcionamiento requerido en las funciones i-vi y, muy importante para el ejercicio, sólo utilizar smart pointers.

4. Recursión y ejecución en tiempo de compilación:
   a. Se requiere el código de una función recursiva que compare dos variables que contengan texto e indique mediante una variable bool si son iguales (true si son iguales, false si no lo son). Explique su elección entre los tipos string y char* (lea el siguiente item de este ejercicio antes de elegir el tipo de variable) y demuestre la funcionalidad de su código con un texto de, al menos, 64 caracteres.

   b. El componente high_resolution_clock de <chrono> permite calcular el tiempo de ejecución de un proceso al hacer:

      ```
      #include <chrono>
      …

      auto startTime = std::chrono::high_resolution_clock::now();
      miProcesoAMedir();
      auto endTime = std::chrono::high_resolution_clock::now();
      auto elapsedTime = std::chrono::duration_cast<std::chrono::nanoseconds>(
      endTime - startTime);

      std::cout << “A miProcesoAMedir le tomó: “ << elapsedTime.count() << “
      nanosegundos” << std::endl;
      ```

      Utilice esté código y las modificaciones necesarias que crea conveniente para verificar cuanto tiempo toma la ejecución del código del ejercicio 4.1.

   c. Modifique el código del ejercicio 4.1 para que la comparación de los textos se realice en tiempo de compilación y obtenga el tiempo de ejecución. Compare este tiempo con el obtenido en el ejercicio 4.2 y justifique brevemente la diferencia (puede escribir su conclusión como un comentario al final del código de este item).