	**Universidad de Costa Rica** **Escuela de Ciencias de la Computación e Informática** **Semestre I - 2021** **Curso CI-0113 - Programación II** **Profesor:** Edgar Casasola Murillo

Tarea programada II

https://git.ucr.ac.cr/ci0113/tarea2

Forma de entrega: Grupos (3)	**Fecha de entrega: Jueves 8 de junio - 11:55 p.m.**

Forma de entrega: Subir a plataforma.ecci.ucr.ac.cr según se indica en el enlace respectivo. Solo tiene que subir la tarea 1 integrante de cada grupo. Deben entregar documentación interna y externa, en formato PDF.

Descripción del problema

Se realizará un Solucionador Genérico capaz de resolver problemas solucionables. Cada estudiante diseñará e integrará una implementación de Solucionador y una implementación de Problema original. Además, cada equipo deberá programar cualquier .cpp que consideren necesario para la funcionalidad de la aplicación. Para esta tarea deberán de utilizar como base los archivos aportados por el profesor.

Clases

Un ***Problema*** solucionable debe contar con:

- Un Estado inicial
- Un método para detectar cuando se ha llegado a la solución.
- Un método para calcular una Heurística: este método hará un estimado de qué tan cerca se está dado un estado actual de llegar al Estado meta. Mide la distancia entre el estado actual y el estado meta. No todos los problemas pueden tener esta medida por lo que en caso de ser este el caso, el método retornará infinito para indicar que es desconocido. Esta heurística puede ser utilizada por un solucionador para resolver el problema.
- Un método para pedirle a un estado que retorne la lista de estados posibles que se generan inmediatamente después de él
- Lista getSiguientes(Estado *) Retorna los siguientes estados

Por ejemplo, para problemas *como el del 8-puzzle* ***(ESTO ES SOLO UN EJEMPLO)*** (consiste en una matriz de números que deben ser ordenados, moviéndose solo si hay un espacio en blanco en su cercanía. Más información del ejemplo: https://en.wikipedia.org/wiki/15_puzzle). El método getSiguientes() será que dado una matriz de números, devolverá un una lista de estados con matrices que representan todos estados a los que se llega después de realizar un movimiento válido.

Un ***Estado*** debe contar con:

- El Estado debe ser amigo del Problema específico. Dado que **el Problema es el que debe retornar con el método getSiguientes(...)** los siguientes estados posibles, no el Estado.
- Sobrecarga del operador de >>: este método permitirá que cada estado pueda ser cargado con sus respectivos datos.
- Sobrecarga del operador de <<: este método permitirá que se pueda imprimir un Estado.
- Sobrecarga del operador == : se utilizará para comparar Estados.
- Sobrecarga del operador!=

Recordar que cada problema sabe a cuál estado debe de llegar, por ende, el Estado utilizará esta información para indicar que el problema de cierto tipo fue resuelto.

Por último, se tendrá un **Solucionador** que recibe desde su **Problema** el conjunto de estados resultante del método que recibe un Estado y se encarga de expandir ese Estado y retornar una lista de estados siguientes utilizando el método (Lista * getSiguientes(…) que existen en todos los problemas)
y según el tipo de solucionador utilizará una técnica para decidir cuál de estos estados recibidos expandir. En otras palabras, decide cómo “explorarlos”.

Un ***Solucionador*** debe contar con:

- Un método solucione(Problema *) y retorna una instancia de Solucion

Utilización de Fábricas

Se utilizará un patrón del modelo fábrica abstracta y producto abstracto, donde una fábrica puede generar dos tipos de **Producto**: **Problemas** y **Solucionadores.** La fábrica será la clase padre de la cual deberán heredar las clases hijas según su función, además de indicar los métodos abstractos que cada hijo implementará posteriormente. Más específicamente, la fábrica deberá tener un método **producir()**, el cual es un método virtual puro, que devuelve un puntero a un Producto, en este caso puede ser un Solucionador o un Problema. La Fabrica tendrá un método para preguntarle si produce un “nombre de producto” particular (int produce(char * nombreProducto)

Se contará con un **Registro**, el cual contiene vector con una fábrica en cada posición. Cuando una fábrica es creada, se inscribe en una posición del registro. Puesto que cada fábrica identifica con el nombre de su **Producto**, el Registro se utiliza para comparar este nombre contra los nombres reconocidos por todas las fábricas inscritas en él, para determinar cuál es la fábrica asociada a ese Producto. Al terminar el programa, el registro destruye las fábricas inscritas.

Main

El main fue diseñado y construido en clase y al igual que los .h abstractos **no podrá modificarse**. En este se creará el Estado (se escoge uno de la Fábrica de estados) y pasa al Solucionador escogido el problema.

Los parámetros de entrada son:

- Nombre del problema
- Nombre del solucionador

Documentación Interna

Debe venir en cada archivo .cpp . Para cada método, debe venir explicado en un comentario su: *función, parámetros, retorno.*

Documentación Externa

Debe iniciar con una copia del enunciado de la tarea para que se sepa que hace esa tarea. Expliquen cómo solucionaron el problema (diagramas de clase pueden ser útiles aquí). Hagan un análisis de su trabajo, cosas que se pueden mejorar, puntos donde el programa puede fallar para saber si ustedes lo detectaron antes y están conscientes del problema. Por ejemplo alguna validación de entrada de datos, que se acaba el archivo en forma abrupta etc.

Es conveniente capturar pantallas y mostrar casos de funcionamiento del programa, además de cómo se compila y ejecuta. Es como un manual para que alguien vea como funciona , y que efectivamente funciona. Para eso capturan pantallas y las van explicando paso a paso

La tarea debe entregarse en Mediación Virtual, no es suficiente con tener el código en Gitlab. Entregar una tarea por equipo.