# Diagrama de clases

Diagrama

Descripción generada automáticamente

# Documentación clase plantilla: VECTOR

La clase plantilla VECTOR se hace con el ánimo de tener una clase a disposición con algunos atributos y métodos del contenedor vector de la STL, nuestra implementación claramente no maneja la misma eficiencia que dicho contenedor, pero para la dimensión de este programa es una buena opción.

La clase VECTOR consta de dos atributos, que son:

* **longitud:** Este atributo guardara la longitud del arreglo VECTOR y que además es un dato de tipo unsigned short ya que según nuestro criterio nuestro programa no llegará a tener VECTORES con una longitud mayor 65.5365 que es el máximo número entero que soporta el tipo de dato elegido, de hecho, se espera contar con VECTORES con tamaños muy por debajo de este valor.
* **estructura:** Es un puntero que apunta a el arreglo dinámico que contiene los elementos del VECTOR, es un puntero de tipo T\*, aquí es donde se usa la plantilla T para que se pueda crear un arreglo dinámico de cualquier tipo de dato.

Los métodos que contiene dicha clase y sus respectivas descripciones se listan a continuación:

* **-reservarEspacio():void**

Este método aumenta un espacio más a la capacidad del arreglo al que apunta el atributo estructura, lo que hace es crear un arreglo dinámico auxiliar del mismo tamaño que la estructura, luego se copia todo el contenido de la estructura a el arreglo auxiliar, después se libera estructura y se le reserva nuevamente un arreglo dinámico de tipo T pero ahora de longitud + 1, se copia nuevamente todo lo del arreglo auxiliar en estructura y se libera el arreglo auxiliar.

* **-recortarEspacio():void**

Este método quita un espacio de la capacidad del arreglo al que apunta el atributo estructura, lo que hace es crear un arreglo dinámico auxiliar del mismo tamaño que la estructura, luego se copia todo el contenido de la estructura a el arreglo auxiliar, después se libera estructura y se le reserva nuevamente un arreglo dinámico de tipo T, pero ahora de longitud - 1, se copia nuevamente todo lo del arreglo auxiliar en estructura y se libera el arreglo auxiliar.

* -copiarConte(T \*estructOrigin, T \*estrucDestin):void

Hace una copia uno a uno de los elementos de la estructura Origen en la estructura destino.

* +append(T element):void

Este método permite agregar un elemento a la última posición de memoria del arreglo al que apunta el atributo estructura, internamente llama al método reservarEspacio y luego asigna a la última posición del arreglo el elemento pasado como parametro

* +delPos(T pos): T

Elimina el elemento almacenado en la posición pasada por parámetro, lo que se hace es correr todos los elementos del arreglo al que apunta el atributo estructura hacia la izquierda a partir de la posición ingresada como parámetro y al final llama al método recortarEspacio para recortar la última posición del arreglo y luego disminuye el atributo longitud en una unidad.

* +len():int

Retorna el valor del atributo longitud del VECTOR.

* +getfirst():T \*

Retorna la dirección de memoria del primer elemento del arreglo dinámico de tipo T al que apunta el atributo estructura.

* +index(T element) const: int

Retorna la posición del elemento pasado como argumento dentro del arreglo dinámico al que apunta estructura.

Los constructores de esta clase platilla son dos:

* VECTOR ()

Este es el constructor por defecto, es decir, si no ingresan parámetros en la inicialización del objeto se usará este, el atributo longitud se inicializará en cero y el atributo estructura apuntará a un arreglo dinámico de tipo T de longitud cero.

* VECTOR (int tam)

Este constructor se podrá usar si se desea reservar un tamaño especifico desde la inicialización para el arreglo al que apunta el atributo estructura, el atributo longitud se inicializará con el unsigned short pasado por parámetro y el atributo estructura apuntará a un arreglo dinámico de tipo T de tamaño igual al atributo longitud.

# Análisis del Desafío 2

Lo primero a mencionar es que este desafío es totalmente diferente al desafío 1, ya que en este nos dan una problemática mucho más cercana a nosotros, ya que en Medellín al igual que en México existe una red metro, por lo que es más fácil imaginar este problema, además el problema en requiere el sentarnos a pensar acerca de cálculos u operaciones complicadas, el problema es más aterrizado a una problemática definida, Cree un algoritmo

capaz de modelar una red metro.

Entonces después de leer el documento observamos que lo primero es poder definir las clases que van a hacer parte fundamental para una red metro, estos son

* Estaciones: Esta clase la vemos como la base de la red metro, ya que en esta vamos a definir el nombre de las estaciones, el tiempo que hay entre una u otra estación (delantera y trasera), y también sabremos si es o no una estación de transferencia. Para esta parte que hablamos de estación de transferencia podemos ver como si cada estación fueron un vértice en un grado, donde para una estación que no es de transferencias este vértice tendrá un grado de 1(si la estación final o inicial) o 2 (si la estación tiene estaciones delanteras y traseras). Ahora si este grado es mayor a 2 quiere decir que la estación es de transferencia, en ese caso se le pedirá al usuario con que línea quiere que haga conexión la estación
* Líneas: Esta clase se puede ver como un conjunto de estaciones, donde dentro de estas líneas no hay bucles, no hay bifurcaciones, pero si hay algo importante a mirar y es que estas líneas son bidireccionales, por lo que depende la estación de origen y la estación de llegada para observar de en qué sentido vamos.

Además, existen diferentes tipos de líneas ya que puede ser del metro, o del tranvía o lo que el usuario quiera agregar, por lo que para poder diferenciar las líneas y las estaciones se tendrá un nombre donde primero será el nombre de la estación después un guion bajo finalmente el tipo o la letra la Línea a la cual pertenece esa estación.

* RedMetro: Esta clase es un conjunto de líneas, donde todas están conectadas, en esta clase no hay inconvenientes como tal, mientras la clase líneas y estaciones están bien definidas, la RedMetro es solo funciones del manejo de las líneas que conforman esa red.

Esto es el modelamiento de las clases, dentro de cada una hay diferentes funciones dependiendo de la necesidad del desafío. Por último, se habla de poder encontrar el tiempo de llega a otra, este problema se manejará desde la clase estructura que es donde se define el tiempo que hay entre las estaciones anteriores y posteriores.