Práctica Laboratorio 2 – Venta de entradas y árbol binario

Datos alumno:

Nombre: Marcos Barranquero

Dni: 51120194N

Detalles y justificación de TADS:

TAD's creados:

- ColaCliente (con Nodo)
- Lista (con Nodo)
- Árbol Binario (con NodoArbol)

Definición de TAD's:

- Nodo: es un nodo que contiene un puntero a un objeto cliente y un puntero al siguiente nodo. Utilizado para formar la cola de clientes ColaCliente y lista.
  - Argumentos:
    - Cliente: recibe un cliente que guardará en memoria con un puntero hacia él.
    - Puntero siguiente: puntero que apuntará al siguiente Nodo. Es un argumento **opcional**, y si no se pasa como argumento, el atributo puntero siguiente apuntará a NULL.

- ColaCliente: es un conjunto ordenado de nodos Nodo. Permite ver al primero y al último, además de insertar por el final, y eliminar por el principio.
  - Argumentos: no recibe ningún argumento al instanciarlo. Para operar con él se usarán sus métodos contenidos, que paso a describir:
    - Insertar: tiene por argumento un cliente, que meterá en un nodo y lo pondrá como el último de la cola. No devuelve nada. El atributo size aumenta en 1.
    - Eliminar: no tiene argumentos, simplemente elimina el primer elemento de la cola, devolviéndolo al llamar a este método. El atributo size disminuye en 1.
    - verPrimero y verUltimo: permiten ver **sin eliminar** a los clientes del nodo primero y último. Usado para comprobar al insertar ordenadamente entre otras cosas. Devuelven en cliente primero ó último.
    - Mostrar: muestra por pantalla todos los clientes que se encuentran en la cola.

        - getSize: devuelve el atributo entero size, o tamaño de la cola. Es decir, el número de nodos que hay dentro de esta.
        - Vaciar: vacía la cola, es decir, elimina todos los nodos que contiene.

- Lista: es una secuencia ordenada de nodos Nodo conteniendo a clientes, y ordenada por su ID.
    - Argumentos: ninguno.
        - La mayoría de métodos de ColaCliente están incluidos en lista.
        - InsertarPorId: recibe un cliente como argumento, y en función de su Id, lo inserta en la cola ordenadamente. Si está vacía, lo inserta como primero. La cola debe de estar previamente ordenada antes de meter otro cliente ordenando por Id. No devuelve nada. El atributo size aumenta en 1.

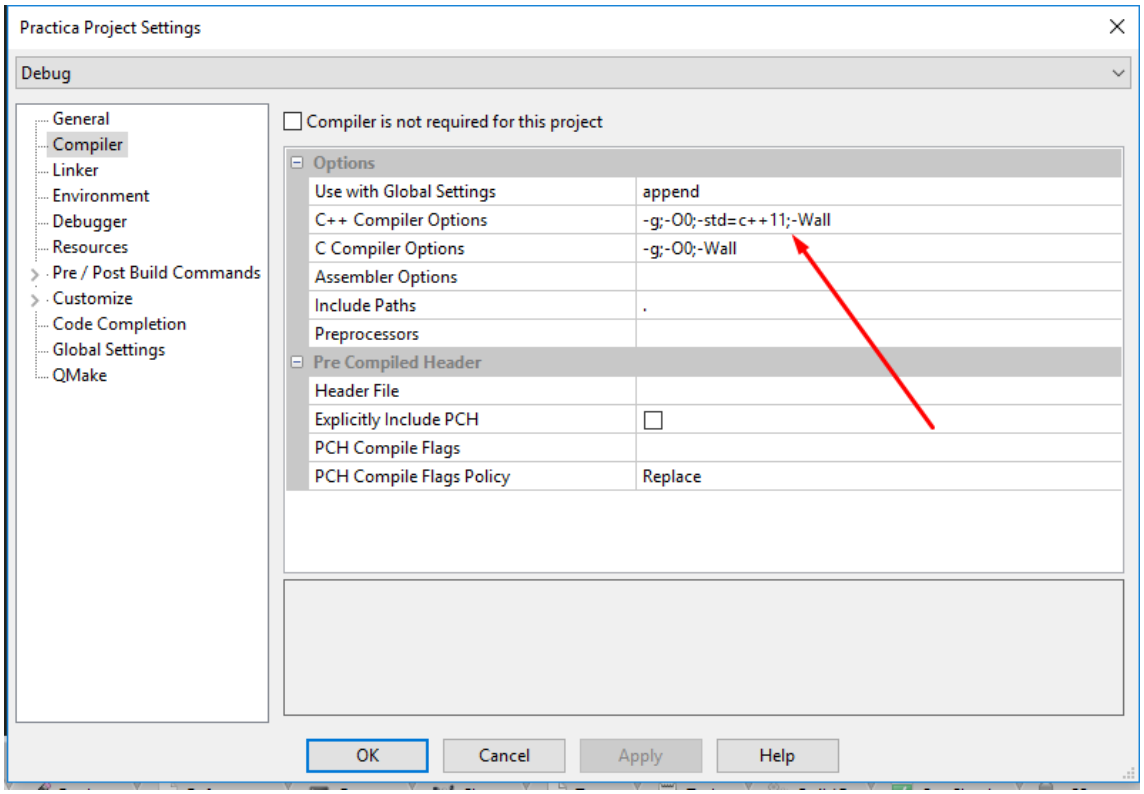
- Nodo Árbol Binario: es un nodo de la EEDD. De árbol binario. Contiene un cliente, un puntero a su hijo izquierdo, y otro a su hijo derecho.
    - Argumentos: cliente, opcionalmente hijo izquierdo e hijo derecho.

- Árbol binario: es una estructura de Nodos de Árbol binario ordenada por el DNI de los clientes que contienen.
    - Argumentos: ninguno
    - Métodos destacables:
        - Insertar: inserta a un cliente en el árbol binario, ordenadamente según su dni. Diferencia los casos de que sea una hoja o que no lo sea, y dentro de esta segunda opción, si debe de crear un nodo o avanzar un nodo.
        - Mostrar preorden, inorden, postorden: muestra por pantalla la lista de clientes contenidos en el árbol según se lea en preorden, inorden o postorden.
        - Buscar cliente: dado un DNI, busca en el árbol binario a un cliente cuyo DNI coincida. Si no lo encuentra, devuelve un cliente con Dni “No encontrado”. Si lo encuentra, lo imprime por pantalla.
        - Imprimir árbol: imprime una representación gráfica del árbol.

Solución adoptada y dificultades encontradas:

1.- Uso de la clase string:

Uno de los problemas que he encontrado es al crear arrays de char para atributos como Dni, id, de la clase cliente. Reservaba espacio en memoria para 5 char, pero se guardaba basura que luego se imprimía por pantalla. La solución ha sido utilizar la std::string que implementa **C++ 11.** Para compilar sin problemas, se debe habilitar esta opción en Codelite. (En clion no es necesario):



También he utilizado la strings para mostrar datos con la función toString y mostrar del objeto cliente.

2.- Insertar ordenadamente por ID.

Una de las tareas que más me ha llevado tiempo ha sido crear correctamente el método insertarPorId(cliente) de la clase ColaCliente. Estuve cerca de 3 horas probando y probando como funcionaban los punteros. El concepto de lo que hacía era correcto, sin embargo, cuando insertaba, hacía que un puntero siguiente apuntase a un pnodo **variable local** del método. Esto causaba que al finalizar el método, ese pnodo se eliminase y se convirtiese en basura, haciendo que crashease el programa las más de las veces, o imprimiese un conjunto de caracteres basura en lugar del método mostrar del cliente.

3.- Problemas con el compilador.

La nueva actualización de Windows me ha roto el compilador del programa y no he podido corregirlo. Como no he tenido tiempo de formatear, la solución ha sido trabajar sobre una máquina virtual de w10 con vmware.

4.- Búsqueda por DNI en el árbol binario:

Al comparar std::strings de forma normal (str1 > ó < str2), c++ devuelve 0 si son iguales y 1 si son diferentes. Puesto que no puedo saber cuál es mayor, la solución ha sido convertir los números del string dni de cliente a enteros, descartando la letra. De esta forma, puedo ver cuál es mayor o menor.

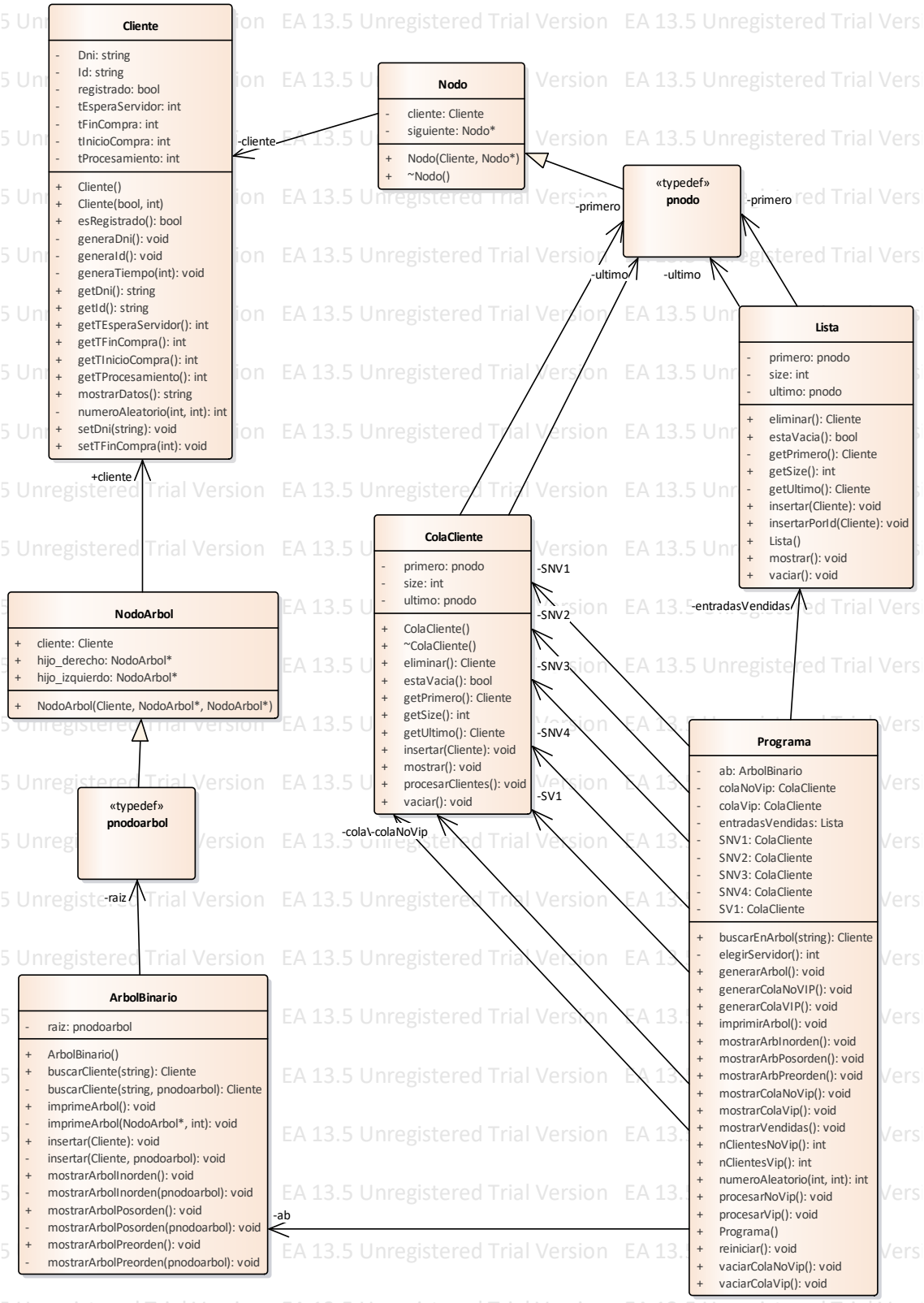
5.- Imprimir el árbol por pantalla:

He utilizado una implementación basada en la creada en este vídeo:
https://www.youtube.com/watch?v=eZfXKHrMyo8

Imprime, de izquierda a derecha, los nodos superiores a inferiores. Los hijos izquierdos se muestran debajo del nodo, y los derechos encima de éste.

Diseño de relación de clases y TAD's.

Diagrama UML:



Explicación de métodos más destacados:

Los métodos más destacados de los TAD's los doy por explicados en el apartado de Definición de TADS. Paso a explicar los métodos más destacados de la clase programa y cliente:

- Programa:
  - GenerarColaVip y GenerarColaNoVip: rellena los dos atributos colaVip y colaNoVip de la clase con 15 y 25 clientes generados aleatoriamente, respectivamente.
  - VaciarColaVip y VaciarColaNoVip y vaciar vendidas: vacía de clientes la cola vip y no vip, y de entradas vendidas.
  - MostrarColaVip y MostrarColaNoVip y mostrarVendidas: muestra por pantalla de consola la todos los clientes de la colaVip o de la colaNoVip, o de la lista de entradas vendidas.
  - nClientes Vip y nClientesNoVip: devuelve el número de clientes que hay en cada cola respectiva.
  - numeroAleatorio: genera un número aleatorio entre un mínimo y un máximo. (Inclusivos).
  - procesarVip y procesarNoVip: simulan el procesamiento en servidores de los clientes vip y no vip. Para ello, utiliza las colas SV1 (servidor vip 1) para los vip, y SNV1, SNV2, SNV3, SNV4 para los servidores de clientes no vip, y los almacena en la lista de entradas vendidas, ordenados directamente por ID.
  - Generar árbol binario: genera un árbol binario partiendo de la lista de entradas vendidas.
  - Mostrar árbol en preorden, inorden, posorden e imprimir árbol: llaman al método concreto definido en el TAD.
  - Reiniciar: reinicia todo el programa. Pone todo a 0, vacía colas, listas, árboles binarios, etc.

- Cliente:
  - Todos los get devuelven el valor del atributo al que referencian.
  - El constructor recibe un booleano que refiere a si el cliente es registrado o no, y un entero que setea el tiempo de inicio de compra.
  - mostrarDatos(): devuelve una representación del cliente y sus datos.
  - generaDni: genera un DNI utilizando el método num. Aleatorio para generar 9 números aleatorios. En función de esos 9 números, calcula la letra correspondiente a ese DNI. Finalmente lo concatena todo en una string.
  - GeneraId(): Partiendo de un diccionario completo, coge 4 letras aleatorias y las concatena en una string. Inicialmente era un array de chars, pero causaba problemas explicados en la sección de solución adoptada y dificultades.
  - Setters de tiempo, dni, etc.: permiten setear un valor del cliente, bien puede ser el tiempo de inicio, de finalización, etc. , o el DNI.

Explicación del comportamiento del programa:

El programa simula una compra de entradas por parte de clientes que pueden ser registrados o no registrados.

Utilizando la opción A) y D) del menú, podremos generar una cola con solicitudes de clientes con datos generados aleatoriamente comprando entradas.

Con la opción B) y E), podremos imprimir estas colas de solicitudes por pantalla.

Con la opción C) y F), podremos vaciar ambas colas respectivamente, limpiándolas de usuarios.

Con la opción G), simularemos el procesamiento de solicitudes de clientes VIP por parte de un servidor, siendo finalmente insertados en una lista de clientes procesados, ordenados por su Id.

Con la opción H), simularemos el procesamiento de solicitudes de clientes no VIP por parte de 4 servidores, siendo finalmente insertados en una lista de clientes procesados, ordenados por su Id.

Con la opción I) mostraremos la lista de clientes ya procesados y por lo tanto con entradas compradas.

Con la opción J) vaciaremos todas las listas, colas, y árbol binario, y pondremos todos los contadores a 0, reiniciando el proceso de compra.

Con la opción K) crearemos un árbol binario tomando a los clientes de la lista de entradas vendidas, ordenado por el DNI de estos clientes.

Con la opción L) podemos buscar en el árbol binario a un cliente introduciendo su DNI. Si no existe, se imprimirá un error.

Con las opciones M), N), O), podremos imprimir a los clientes del árbol binario en preorden, inorden y postorden.

Con la opción P) podremos dibujar el árbol binario en pantalla.

Con la opción S) saldremos del programa.

Código fuente:

Se halla en la carpeta código del .Rar

Bibliografía:

Stack Overflow (https://stackoverflow.com/)

Youtube (Empieza a programar, Jesús Conde, Programación ATS…)

Diapositivas de clase.