Práctica 5. Árboles AVL

Sesiones de prácticas: 2

Objetivos

Implementar la clase `AVL<T>` utilizando **patrones de clase y excepciones**. Programa de prueba para comprobar su correcto funcionamiento.

Descripción de la EEDD

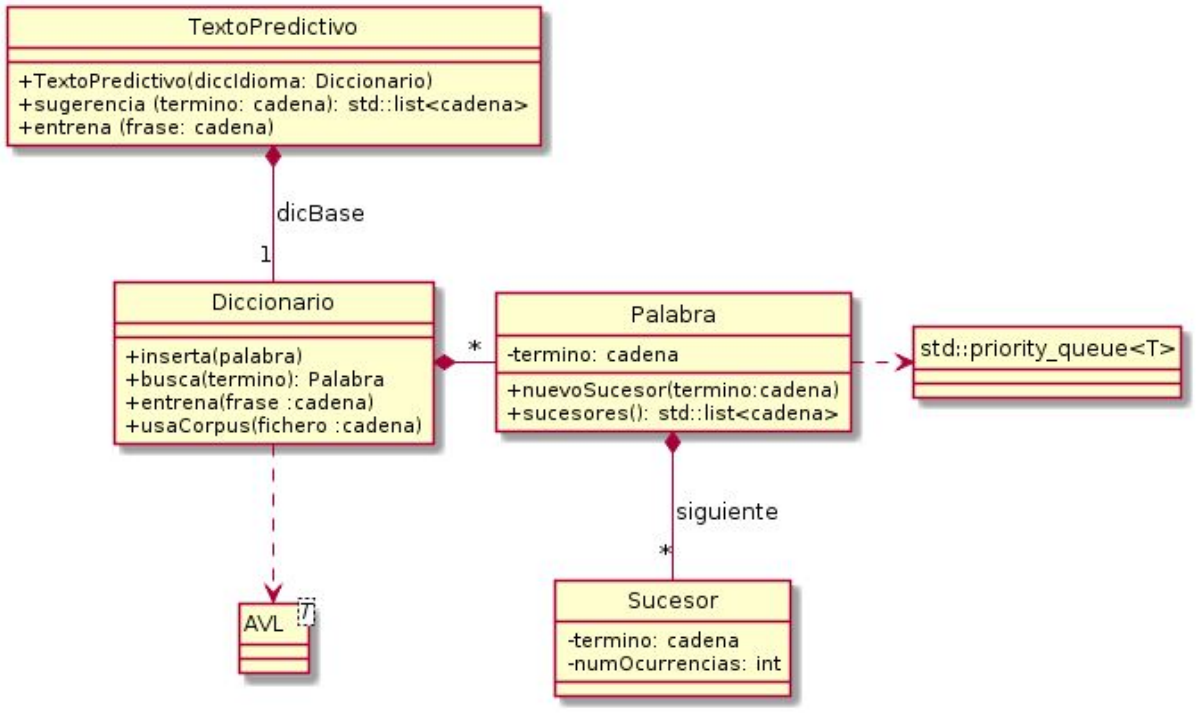
Implementar la clase `AVL<T>` para que tenga toda la funcionalidad de un árbol equilibrado AVL en memoria dinámica, tal y como se describe en la Lección 11, utilizando patrones de clase y excepciones. Los métodos a implementar serán los siguientes:

En concreto se usará:

- Constructor por defecto `AVL<T>`
- Constructor copia `AVL<T>(const AVL<T>& origen)`.
- Operador de asignación (=)
- Rotación a derechas dado un nodo `rotDer(Nodo<T>* &nodo)`
- Rotación a izquierdas dado un nodo `rotIzq(Nodo<T>* &nodo)`
- Operación de inserción `bool inserta(T& dato)`
- Operación de búsqueda `bool busca(T& dato, T& result)`
- Recorrido en inorden[1] `void recorreInorden()`
- Número de elementos del AVL `unsigned int numElementos()`
- Altura del AVL, `unsigned int altura()`
- Destructor correspondiente

Nuevamente, la funcionalidad básica de la práctica no cambiará. Sin embargo, se realizará un pequeño cambio en su diseño. Se añadirá la clase de sistema `TextoPredictivo` que se compondrá de un `Diccionario`. Además, el contenedor de `Palabra` en `Diccionario` pasará de ser de una estructura lineal a un árbol AVL. El resto de EEDD se mantendrán como en la Práctica 4.

[1] Para poder mostrar por pantalla los datos del arbol es necesario que la clase T haya implementado el operador << http://en.cppreference.com/w/cpp/language/operators



Programa de prueba 1: Comprobación de la estructura de datos

Para comprobar la funcionalidad implementada, se declarará un `AVL<int>` en el que se introducirá la siguiente secuencia de enteros: 5, 2, 8, 7, 6, 9, que forzará a usar dos rotaciones simples. A continuación, copiar/asignar en un nuevo `AVL<int>`. Mostrar el resultado de los árboles recorriéndolos en inorden antes y después de cada operación.

Programa de prueba 2: Mostrar los términos más habituales

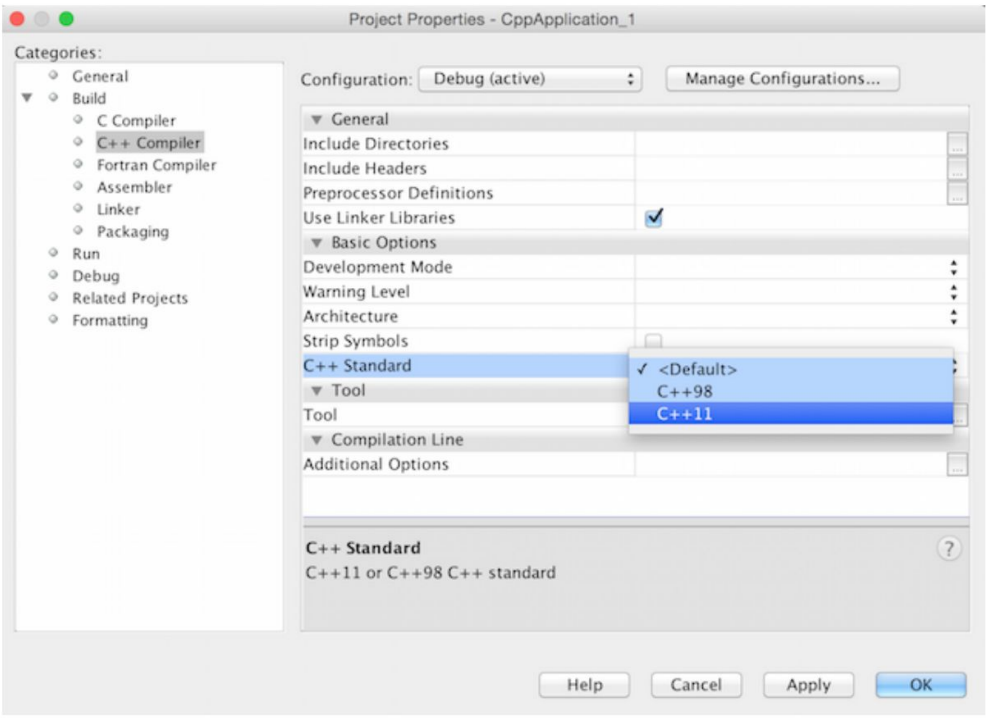
El programa de prueba mostrará palabras recomendadas, ordenadas por prioridad, a aquellas que introduzcamos por pantalla, al igual que en las Prácticas 3 y 4. De forma adicional, mediremos el tiempo usado por el sistema en introducir el conjunto de palabras en el diccionario y en entrenarlo usando un corpus. Para ello usaremos el estándar de C++11 `chronos` para comparar el tiempo de la versión con el `std::vector<Palabra>` de la Práctica 4 y el nuevo `AVL<Palabra>`. A continuación tenéis un ejemplo de uso:

```
auto start = std::chrono::system_clock::now();
for (int i=0; i<nElements; ++i)
       vector[i] = i;
auto end = std::chrono::system_clock::now();

auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end-
start).count();

cout << "Time spent (ms) : " << elapsed_ms << endl;
```

A partir de esta práctica se usarán características del estándar de C++11 como `auto` o `std::chrono`. En algunos IDE como Netbeans hay que activarlo explícitamente en las propiedades del proyecto como se muestra en la captura siguiente:



Estilo y requerimientos del código:

1. El código debe ser claro, tener un estilo definido y estar perfectamente indentado, para ello se pueden seguir algunos de los estilos preestablecidos para el lenguaje C++ (http://geosoft.no/development/cppstyle.html).

2. Deben comprobarse todas los posibles errores y situaciones de riesgo que puedan ocurrir (desbordamientos de memoria, parámetros con valores no válidos, etc.) y lanzar las excepciones correspondientes, siempre que tenga sentido. Leer el tutorial de excepciones disponible en el repositorio de la asignatura en docencia virtual.

3. Se valorará positivamente la calidad general del código: claridad, estilo, ausencia de redundancias, etc.