Guía de instalación y manual de usuario

Detección de errores en bases de datos químicas

Jesús Navarro Merino

Julio 2023

1. Introduccion

En este documento se describirán los procesos de instalación de las herramientas necesarias para replicar la ejecución del proyecto 'Detección de errores en bases de datos químicas'. El trabajo de implementación se ha llevado a cabo en Windows 10 bajo una arquitectura de 64 bits usando el IDE Microsoft Visual C++. Como el código base pertenece a OpenBabel (ver *aquí* el repositorio oficial), ellos mismos tienen su guía de instalación que se puede ver *aquí*. Pero la guía está un poco desactualizada y al hacerlo en Windows quizás no se entiende del todo el procedimiento, por lo que en este documento se explican los pasos que se han seguido.

Para compilar el proyecto es necesario lo siguiente:

- El código fuente de OpenBabel, ya sea el desarrollado para este proyecto (*aquí*), o el original (*aquí*). Obviamente, si se quieren probar los cambios introducidos, hará falta utilizar el primero.
- Un compilador de C++, el que menos problemas da es el MSVC++.
- CMake, mínimo en su versión 3.1. OpenBabel utiliza CMake para construir el proyecto y las distintas soluciones.

2. Pasos previos

2.1. CMake instalación

Para instalar CMake en Windows lo más recomendable es descargar directamente el instalador para su última versión estable desde el sitio oficial de descargas *aquí*. Concretamente en la Sección 'Latest Release' y 'Windows x64 Installer'.

Una vez instalado siguiendo el wizard, hay que asegurarse de añadir a las variables del sistema PATH la ruta a la carpeta '\bin' de CMake. Si se sigue la instalación por defecto sería 'C:\Program Files\CMake\bin'.

2.2. Microsoft Visual C++ instalación

Antiguamente el compilador MSVC++ se podía instalar de manera independiente, pero a día de hoy forma parte del IDE Microsoft Visual C++, por lo que será necesaria su instalación (se puede seguir si se desea la guía oficial de microsoft para su insalación *aquí*). Para ello, nos descargamos la versión Community 2022 desde su página oficial *aquí*. Al correr el instalador se nos muestra la siguiente pantalla de la Figura 1.

Aquí, nos servirá con seleccionar en el menú de Cargas de trabajo la opción 'Desarrollo para el escritorio con C++'. Esto incluye todas las herramientas necesarias para trabajar con C++ en Windows. Este proceso de instalación puede tardar unos minutos.

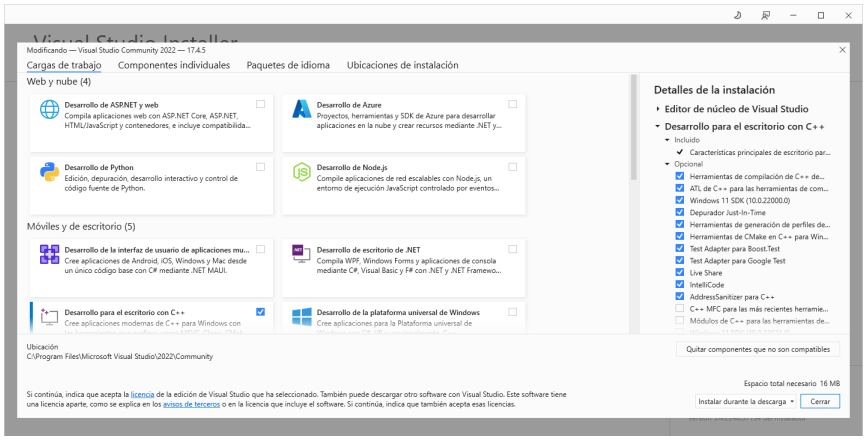

Figura 1: Componentes de instalación

Tras una breve configuración de personalización, se abrirá el menú de creación de proyectos. Desde aquí, deberemos seleccionar el proyecto de openbabel, pero primero hay que generar dicho proyecto.

2.3. Generación y compilación del proyecto

Una vez descargado el '.zip' con el código fuente mencionado en la Sección 1, lo descomprimimos en la ubicación que deseemos, y nos posicionamos en 'TFG-openbabel-v0.1\code\'. Esa será nuestra carpeta de trabajo.

Los pasos a seguir son los siguientes:

1. Se recomienda hacer el *build* en un directorio separado:

```
$ mkdir build
```

2. Utilizamos **cmake** para configurar la *build*. Esto detecta los archivos CMakeLists y genera los ficheros de proyecto para Visual C++.

```
$ cd build
$ cmake ..\openbabel-openbabel-3-1-1\
```

3. Volvemos a Microsoft Visual C++ y desde el menú de creación de proyectos ya podremos abrir el archivo principal 'openbabel.sln', generado por el paso anterior (Figura 2).

4. Veremos una interfaz como en la Figura 3. En la ventana de la izquierda 'Solution explorer' aparecen todas las soluciones del proyecto (marcadas en la figura en naranja). Para compilar todo, hacemos click derecho en ALL_BUILD y seleccionamos *Build*. Este proceso de compilación suele tardar ≈3 minutos.

Una vez termine el proceso de compilación, si no se ha producido ningún error, se generará un directorio '*bin\Debug*' con los ejecutables que se llaman por línea de órdenes. Podemos en este caso:

- Posicionarnos en dicho directorio y trabajar desde ahí con llamadas tipo '*./obabel.exe <argumentos>*'
- O bien añadir esa ruta (*...\build\bin\Debug*) a las variables del PATH y utilizarlas como llamadas al sistema desde una terminal en cualquier sitio con llamadas tipo '*obabel <argumentos>*'

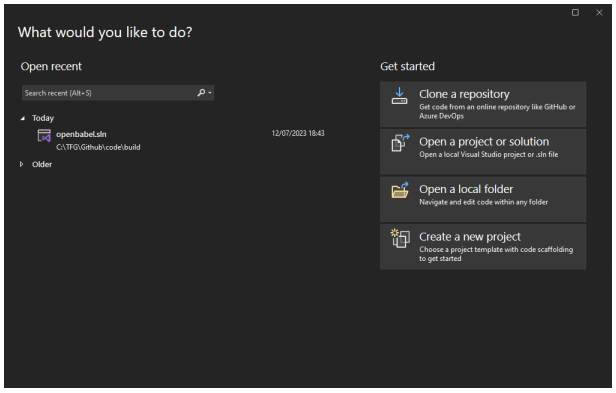

Figura 2: Menú de creación de proyectos

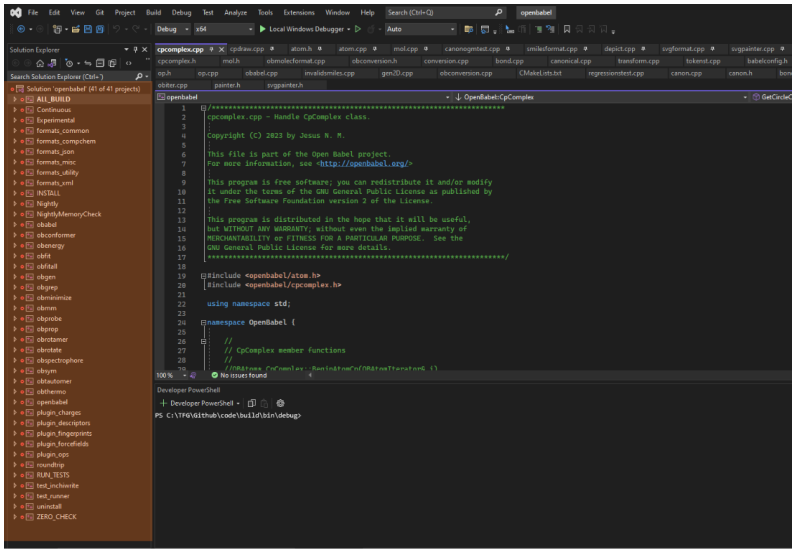

Figura 3: Interfaz del proyecto en Visual C++

3. Ejemplos de uso

Podemos usar la Windows Powershell, o más cómodo, la terminal que el propio IDE ofrece. En *este pdf* con la documentación al completo se describen todas las opciones de ejecución para openbabel. Las que se han utiliazado para el proyecto son las funcionalidades centrales a través de obabel, usando órdenes tipo

obabel <ficherosEntrada>-O <ficherosSalida><opcionesSalida>

A continuación, algunos ejemplos con las llamadas más importantes para generar los resultados (sustituir las rutas de los ficheros de entrada y salida por las adecuadas):

Resultados

- Para convertir una SMILES individual en el SMILES canónico:

```
$ ./obabel.exe -:"O#C[Fe+2]1234([I-])(C#O)[CH]=5[CH]4=[CH]3[CH-]2[CH]51
    " -O "C:\TFG\dataset\output\iron(II).smi"
```

- Para generar la representación 2D de un SMILES individual a un fichero .svg, simplemente cambiar la extensión del fichero de salida:

```
$ ./obabel.exe -:"O#C[Fe+2]1234([I-])(C#O)[CH]=5[CH]4=[CH]3[CH-]2[CH]51
    " -O "C:\TFG\dataset\output\iron(II).svg"
```

- Para convertir un conjunto de SMILES almacenados en un fichero .smi en sus SMILES canónicos, se especifica el fichero de entrada:

```
$ ./obabel.exe "C:\TFG\dataset\complete_dataset.smi" -O "C:\TFG\dataset
    \output\canon_dataset.smi"
```

- Para generar las representaciones 2D de un conjunto de SMILES almacenados en un fichero .smi, de nuevo con cambiar la extensión del fichero de salida es suficiente. Esto no es conveniente hacerlo para conjuntos de moléculas muy grandes (+100) ya que las agrupa en un único fichero de salida en forma de matriz. Si queremos que debajo de cada molécula aparezca un número identificador o un nombre, colocaremos dicho número al final del SMILES tras un espacio en el fichero de entrada (Figura 4)

```
$ ./obabel.exe "C:\TFG\dataset\complete_dataset.smi" -O "C:\TFG\dataset
    \output\dataset_depictions.svg"
```

Figura 4: Número identificador (en rojo) para cada molécula en el fichero de entrada.

- Para hacer lo mismo que la opción anterior, pero esta vez obteniendo un fichero individual de salida para cada SMILES del fichero de entrada, usar la opción '-m'. En el ejemplo siguiente cada fichero de salida tendrá el nombre 'out1.svg', 'out2.svg', 'out3.svg' y sucesivos.

```
$ ./obabel.exe "C:\TFG\dataset\small_dataset.smi" -m -O "C:\TFG\dataset
    \output\output_individuales\out.svg"
```

- Para utilizar las abreviaciones en las representaciones 2D, utilizamos la opcion '$--genalias\ -xA$':

```
$ ./obabel.exe "C:\TFG\dataset\complete_dataset.smi" -O "C:\TFG\dataset
    \output\depict_ALIAS.svg" --genalias -xA
```

Tests

- Para ejecutar los tests, se usan órdenes tipo

$$test\_runner\ <nombreTest><numeroTest>$$

Por ejemplo, para correr cada uno de los tests por separado y volcar el resultado en otro fichero:

```
$ ./test_runner.exe canonogmtest 1 > "C:/TFG/dataset/output/tests/
    salida_test_1.txt"

$ ./test_runner.exe canonogmtest 2 > "C:/TFG/dataset/output/tests/
    salida_test_2.txt"

$ ./test_runner.exe canonogmtest 3 > "C:/TFG/dataset/output/tests/
    salida_test_3.txt"
```

```
$ ./test_runner.exe canonogmtest 4 > "C:/TFG/dataset/output/tests/
    salida_test_4.txt"

$ ./test_runner.exe canonogmtest 5
```

El test 5 genera un fichero .svg en un directorio por defecto en la ruta '*openbabel-openbabel-3-1-1\test\files\ogm*' junto con el resto de ficheros generados durante la ejecución de los test 1-4.

4. Añadir nuevos ficheros al proyecto

Para añadir nuevos ficheros .cpp o tests y que el compilador los incluya al proyecto, simplemente seguir las guías oficiales de OpenBabel orientadas a desarrolladores. Se puede ver el pdf completo de la documentación (*aquí*) o las páginas específicas para añadir ficheros nuevos (*aquí*) y tests (*aquí* y *aquí*).