27 novembre 2017

(Il seguente testo è stato scritto di getto e l'italiano è pessimo. Tra un paio di millenni potrebbe essere sostituito da uno pensato con maggiore cura)
Il file è venuto lungo solo perché è prolisso e prevede alternative: in realtà non c'è niente di complicato.

Antefatto 1: vi serve un compilatore

Dato che sto studiando il C++ proprio in questo periodo, per me è abbastanza naturale pensare al C++ nella sua ultima versione, ossia lo standard 2017 (C++17); perciò quando scrivo codice, non mi pongo il problema di quanto sia compatibile con le versioni precedenti. Se invece non lo usate spesso, è possibile che sulle vostre macchine sia installata una versione del compilatore che non consente di utilizzare le ultime funzionalità del linguaggio.
Se vi trovate in difficoltà a compilare, un primo consiglio è quindi quello di **aggiornare il compilatore**.
Qui prenderò in esame solo GCC – nello specifico, la versione per Windows di GCC.

Antefatto 2: vi serve la libreria SQLite

A) Avete SQLite installato

Se avete SQLite già installato, la libreria si trova da qualche parte nelle directory di sistema della vostra macchina e dovete solo trovarla. Se però poi vedete che il programma non 'linka' oppure non funziona, potrebbe essere che per qualche ragione quella versione della libreria sia stata compilata solo per C; in tal caso, dovrete seguire uno dei passaggi successivi.

B) Non avete SQLite installato

Se non lo avete installato, **non** siete obbligati a farlo, ma comunque dovrete ottenere la libreria.
Come prima cosa andate a questa pagina:
https://www.sqlite.org/download.html
Qui vi si offrono diverse opzioni:

1) Scaricare la libreria già compilata.

SQLite è scritto in C, ma le funzioni sono precedute dal codice "extern C" che viene incluso quando 'linkate' per C++. Perciò in teoria tutto dovrebbe funzionare bene. Se non dovesse funzionare bene o voleste cambiare nome alla libreria, potete seguire il passaggio successivo. Il file da scaricare è "Precompiled Binaries for…" + il vostro sistema operativo.

La libreria a 32 e quella a 64 bit hanno lo stesso nome, perciò dovrete fare una scelta. Questa scelta vi condizionerà da questo momento in poi, nel senso che se scaricate la versione a 64bit, dovete verificare che il vostro compilatore compili correttamente a 64bit. Se siete in dubbio, potete fare tutto a 32bit e funzionerà allo stesso modo (io sto testandolo sia a 32 che a 64 bit e non trovo differenze).

E… no, non potete semplicemente rinominare la libreria per averle tutt'e due. Una libreria dinamica contiene delle definizioni al suo interno e il nome del file dev'essere quello che è stato deciso in fase di compilazione.
Però potete rinominarla seguendo la prossima opzione.

2) Scaricare il sorgente da compilare.

Io volevo sul pc sia la versione della libreria a 32bit che quella a 64bit, perciò mi sono trovato costretto a seguire questa strada; nel caso per qualche motivo i metodi sopra non funzionassero, vi riassumo cosa fare.
In questo caso il file da scaricare è lo "amalgamation". È stato chiamato così perché, per facilitare la vita agli sviluppatori, i prototipi delle funzioni sono stati inclusi nel codice sorgente, perciò in pratica non vi dovete preoccupare di avere lo header.
Se scaricate il pacchetto "sqlite-amalgamation-3210000.zip" ci trovate dentro ben 4 files. A noi interessa solo "sqlite3.c" giacché, se gli date un'occhiatina dentro, vedrete che già comprende lo "sqlite3.h". Comunque sia, visto che una volta decompresso il pacchetto vi fa trovare a disposizione "sqlite3.h" nella stessa cartella, *per ora* non dovete fare proprio nulla.

Se usate GCC, potete ottenere una libreria dinamica (*shared*) per Windows con l'istruzione:
`gcc -shared sqlite3.c -o sqlite3.dll`
come descritto in questa pagina:
https://www.sqlite.org/howtocompile.html
(Le istruzioni complete per la compilazione sono qui https://www.sqlite.org/compile.html, se proprio volete farvi del male)

Questo è il momento giusto per **scegliere il nome**, se volete.

Linux

Intanto, le librerie dinamiche (*shared libraries*) in Linux di solito si chiamano lib*qualcosa*.so. C'è tutto un meccanismo cervellotico per dare loro il nome che trovate descritto qui:
http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html.
La cosa fondamentale da capire è che se in Linux chiamate una libreria "libmario.so.3.11.433", il suo nome è "mario". Per trovarla, al compilatore dovrete dire "mario".
Se quindi volete una libreria a 32 e una a 64 bit, dovrete fare in modo che la parte di nome fra "lib" e ".so" sia diversa (es: libsqlite3_32bit.so e libsqlite3_64bit.so).

Windows

Potete fare come vi pare, basta che l'estensione del file sia ".dll".
Quindi nomi come sqlite3_32bit.dll e sqlite3_64bit.dll vanno benissimo.

In entrambi i casi, se avete un compilatore a 32bit compilerà a 32bit (e funzionerà anche in un sistema operativo a 64bit), e se avete un compilatore a 64 bit compilerà a 64 bit. Perciò, per avere due versioni, dovete avere **due** compilatori (se invece avete un compilatore a 64 bit che può essere impostato per compilare anche a 32, beh, beati voi).

Il fatto!

Non aspettatevi di trovare un makefile.
Scrivere un makefile significherebbe prendere delle decisioni sulla struttura delle cartelle che non ho voglia di prendere oggi. Se però qualcuno di voi ha voglia di pensare già ad un'organizzazione del codice, può procedere senza indugi e io mi accoderò.

Compilare

Per compilare 'a mano' basta:

Fase uno

Mettere tutti i file .h, tutti i file .cpp e tutti i file .so/.dll in una cartella.
Con la parola tutti, intendo ovviamente anche il file sqlite3.h di cui sopra (ma dai, davvero l'avevate già cancellato?)

Fase due

Aprire una console di Windows o un terminale Linux.
(Farà ridere gli utenti Linux, ma ci sono utenti Windows che non hanno mai usato una finestra MS-DOS, perciò era bene specificarlo.)

Fase tre

Stabilire quale versione del C++ il vostro compilatore supporta (l'avete già aggiornato, vero?). Con g++ basta scrivere:

```
g++ --version
```

Se vi viene fuori una versione dalla 5 in avanti, che io sappia dovrebbe andare bene. Il minimo sindacale dovrebbe essere 5.3. Sul mio pc W7 ho la 7.1.0.
La versione del compilatore determina quale linguaggio C++ potete usare. In ordine di preferenza, a calare, queste opzioni per GCC sono:

```
-std=++17
-std=gnu++17
-std=++14
-std=gnu++14
-std=++11
-std=gnu++11
```

Penso vi convenga partire dalla prima e, se il compilatore vi dà errore, sostituirla con quella successiva.
Per errori diversi da questi mi dovete invece contattare di persona, perché potrei aver scritto del codice che richiede, che ne so, almeno la versione C++14 del linguaggio – come dicevo all'inizio, non lo so nemmeno io.

Fase quattro

Compilare. Il comando può essere ad esempio:

```
g++ -std=gnu++17 -Wall -Wextra -pedantic-errors -pipe -O3 main_01.cpp
CheckInitDb.cpp Luogo_01.cpp RSQLite3.cpp LuogoGestore.cpp -L. -lsqlite3_32 -o
main.exe
```

In cui:

- g++ è il nome del compilatore
- -std=gnu++17 è lo standard massimo che la vostra versione del compilatore vi consente
- -Wall -Wextra -pedantic-errors servono per alzare al massimo il livello degli warnings
- -pipe consuma più memoria in cambio di una compilazione più veloce (da non usare su macchine con poca memoria)
- -O3 imposta il massimo livello di ottimizzazione (file più piccoli, ma più lenti in esecuzione); è consigliato il livello 2: -O2
- *.cpp … i file da compilare;
- -L. la directory in cui si trova la libreria dinamica; il punto vuol dire 'qui'; se quest'opzione non viene usata, ma SQLite è regolarmente installato, il compilatore dovrebbe essere in grado di trovare la libreria lo stesso, frugando far le directory di sistema; se l'avevate ricompilata, **non** è quello che avreste voluto succedesse;
- -lsqlite3_32 il nome della libreria da 'linkare'; in una macchina **Windows**, il compilatore cercherà il file "sqlite3_32.dll"; in una macchina **Linux**, cercherà "libsqlite3_32.so"; se usate quella precompilata, dovreste scrivere -lsqlite3;
- -o main.exe il nome dell'eseguibile da creare.

Eseguire

Ah, ah, ah!
Pensavate fosse finita, vero? :-)
Per eseguire il programma bisogna che nella stessa cartella si trovino i file
`CreaBaseDb.txt`
`Luogo_01.txt`
altrimenti va in crash con allarmanti messaggi d'errore.
Ad es. la riga
```
for(std::string line; std::getline(fin, line); cmds.push_back(line));
```
lancia un'eccezione perché si sta tentando di leggere da un file non aperto.

Le conseguenze del fatto

Immagino che vi stiate chiedendo come mai appaiono tutti quegli inquietanti messaggi d'errore. Che ci crediate o no, è una precisa scelta. Quei messaggi mi aiutano a capire come esegue il programma, e poi non sono messaggi di errore.
Ad es., 'SQLITE_ROW' e 'SQLITE_DONE' non sono messaggi di errore, ma creano un output lo stesso. In questa maniera si può indovinare cosa sta facendo il database.

Se tutto è andato bene, nella vostra cartella ci dovrebbe ora essere un file "prova_01.sqlite3" leggibile da qualsiasi applicazione che si interfacci con database SQLite.
Allo stato attuale, il programma non va più in là di qui.