At-NOW

progetti C++ con SQLite e FLTK

2016.04

Indice

Di cosa si tratta. Fondamenti di SQLite

At-NOW (ovvero All togheter NOW) è la presentazione del progetto di una gestione semplificata di una biblioteca che prende spunto da quello presentato nella parte finale degli appunti *Cpp* e rappresenta un punto di convergenza di quanto trattato anche in *cppGUI* e *DB&SQL* che si danno per letti e i cui concetti già presenti non sono qui affrontati. Il progetto *Gestione Semplificata Biblioteca* qui trattato mette assieme la programmazione C++, le interfacce grafiche e i database per presentare una sintesi di quanto trattato isolatamente negli appunti citati. Per questioni didattiche sono effettuate semplificazioni nell'insieme delle funzionalità offerte dal programma. Alcune possibili estensioni saranno suggerite nel paragrafo finale.

Il database utilizzato, con cui si interfaccia il programma, è basato su SQLite. Esistono librerie per C++ che si interfacciano con i più comuni Database (per esempio con MySQL trattato in *DB&SQL*) ma qui si è fatta la scelta di utilizzare un diffuso (utilizzato, come riportato anche nel sito, nei sistemi Android, in Facebook, Mozilla e tanti altri) e semplice motore di Database con cui si interagisce con SQL. La semplicità di gestione di tale soluzione è dovuta a diversi fattori:

- SQLite non usa una architettura Client/Server: non ci sono server da installare o configurare. Il Database è contenuto in un unico file che può risiedere in qualunque posizione. I diritti di gestione sono quelli della directory in cui si trova il file. Il file può essere spostato in qualunque altra posizione.
- Il file che contiene il Database è *cross-platform*: si può spostare senza alcuna modifica sotto altre piattaforme.
- Oltre a quelle già disponibili direttamente esistono diverse librerie per C++ che permettono una gestione semplificata delle interazioni con il Database. In questi appunti si utilizza una classe ripresa da un sito, il cui URL è nel paragrafo dei riferimenti, e in cui sono definiti tre metodi (per collegarsi al Database e associare il nome del file, per effettuare la query, per chiudere la comunicazione con il Database) per la gestione delle interazioni con il Database.

L'installazione di SQLite (presente, assieme a `sqlite3-dev` necessario per lo sviluppo di programmi che utilizzano i database generati da esso, nei repository delle distribuzioni) rende disponibile un ambiente per l'interazione da terminale con i Database:

```
$ sqlite3 biblio.sqlite
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

il comando `sqlite3` genera il database il cui nome è specificato dopo se il file con quel nome non esiste. Se esiste apre il Database per permettere l'accesso ai dati. Il comando introduce ad un ambiente per molti versi simile al client `mysql` trattato in *DB&SQL*. Al prompt si possono inserire comandi SQL, terminanti con il carattere ; anche distribuiti su più righe, o comandi dell'ambiente. L'elenco dei comandi riconosciuti dall'ambiente è accessibile digitando `.help`.

Dall'ambiente si può generare l'unica tabella utilizzata nel progetto:

```
sqlite> .schema
CREATE TABLE libri(
codice text,
titolo text,
```

```
autore text,
argomento text,
disponibile integer,
cognome text,
nome text,
recapito text);
sqlite>
```

Il comando mostra lo statement SQL utilizzato per generare la tabella `libri`. In SQLite i tipi si possono ridurre a `text`, `integer` (`int`), `real` che hanno affinità con i tipi ammessi dagli altri DBMS (vedere documentazione del sito per le affinità).

Insieme alla classe `Database`, in download dal sito specificato nei riferimenti, è compreso un piccolo programma dimostrativo dell'uso della classe:

```
#include <iostream>
#include <vector>
#include <string>
#include <Database>                                                    /*1*/
using namespace std;

int main()
{
  Database *db;                                                        /*2*/
  db = new Database("Database.sqlite");                                /*3*/
  db->query("CREATE TABLE a (a INTEGER, b INTEGER);");                 /*4*/
  db->query("INSERT INTO a VALUES(1, 2);");                            /*4*/
  db->query("INSERT INTO a VALUES(5, 4);");                            /*4*/
  vector<vector<string> > result = db->query("SELECT a, b FROM a;"); /*5*/
  for(vector<vector<string> >::iterator it = result.begin();           /*6*/
                                       it < result.end(); ++it)
  {
    vector<string> row = *it;                                          /*7*/
    cout << "Values: (A=" << row.at(0) << ", B="                       /*8*/
         << row.at(1) << ")" << endl;
  }
  db->close();                                                         /*9*/

  return 0;
}
```

Nell'esempio proposto (`1`) il file contenente il codice della classe è stato spostato nella directory standard degli include. Se si voleva lasciare il file nella directory contenente i file del progetto la riga andava sostituita da:

```
#include “Database”
```

La `2` dichiara un puntatore ad un oggetto della classe `Database` e nella `3` si associa l'oggetto generato al file fisico con il nome specificato come parametro. Se il file non esiste viene generato.

Le `4` richiamano, per l'oggetto, il metodo per effettuare, ognuna, una query al database. La query va specificata come parametro di tipo `string` (una stringa o una variabile stringa). La query può restituire un risultato, come in `5`, e allora tale risultato va assegnato ad una variabile che è un `vector` di `vector` di stringhe. Il `vector` più esterno contiene, per ogni elemento, una riga della tabella risultato della query e, d'altra parte, una riga è un `vector` che contiene come elementi i campi del record.

Il ciclo `6` scorre le righe della tabella risultato della query. Ogni riga è un `vector` di stringhe (`7`) e i singoli campi sono accessibili specificandone la posizione (`8`) per come elencati nella `SELECT`.

Il metodo richiamato in `9` chiude il collegamento con il database.

Un programma sorgente `prova.cxx` che utilizza la classe Database va compilato con il comando per il link alla libreria `sqlite`:

```
g++ prova.cxx -o prova -lsqlite3
```

Se, come nel progetto della gestione della biblioteca, è necessario utilizzare anche le librerie FLTK, la riga di comando della compilazione per il file `biblio.cxx` diventa:

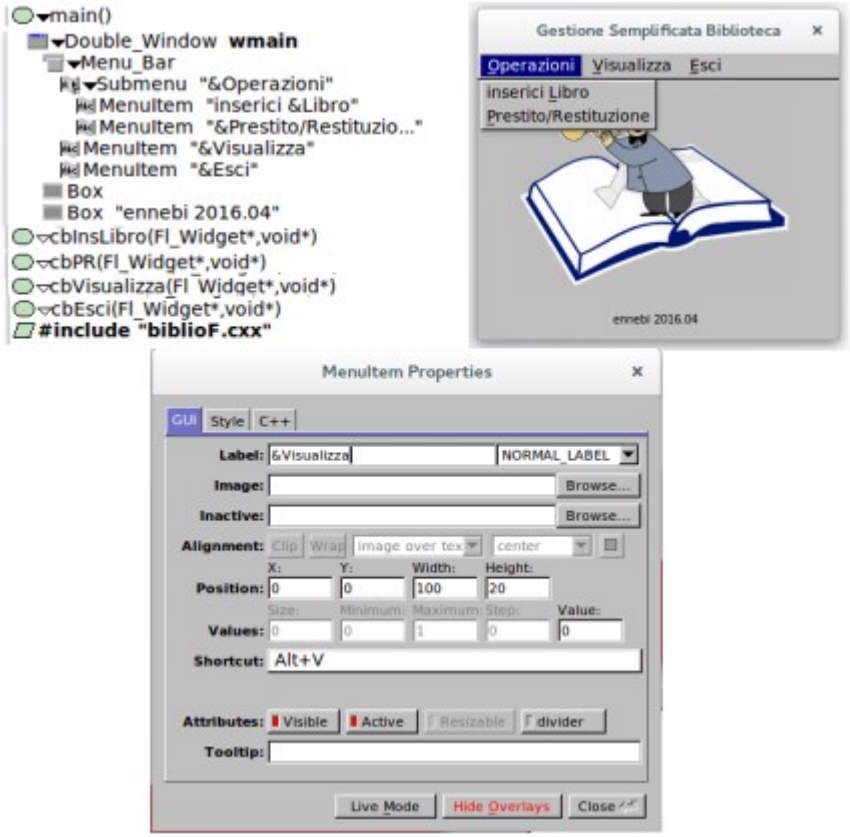
```
g++ biblio.cxx -o biblio `fltk-config --ldflags` -lsqlite3
```

Attenzione!: la stringa `fltk-config --ldflags` non è racchiusa da semplici virgolette ma da virgolette inverse ottenute premendo <ALTGR>+' e che servono a racchiudere un comando che sarà sostituito dal risultato della sua esecuzione. Nel comando la stringa è sostituita dalle librerie necessarie per l'utilizzo del toolkit FLTK.

In definitiva il programma `biblio.cxx` è compilato collegandolo con le librerie per le GUI (FLTK) e per la gestione del database (sqlite3).

Gestione Semplificata Biblioteca

Il programma prevede l'inserimento di libri nella dotazione della biblioteca, le operazioni di prestito/restituzione e diverse possibilità di visualizzazione della dotazione di libri della biblioteca. Come già specificato in altra sede (*cppGUI*) tutte le callback sono codificate nel file `biblioF.cxx` che viene incluso nel file principale `biblio.cxx` in cui è definita, assieme all'altro file `biblio.h` generato da FLUID, l'interfaccia del programma.



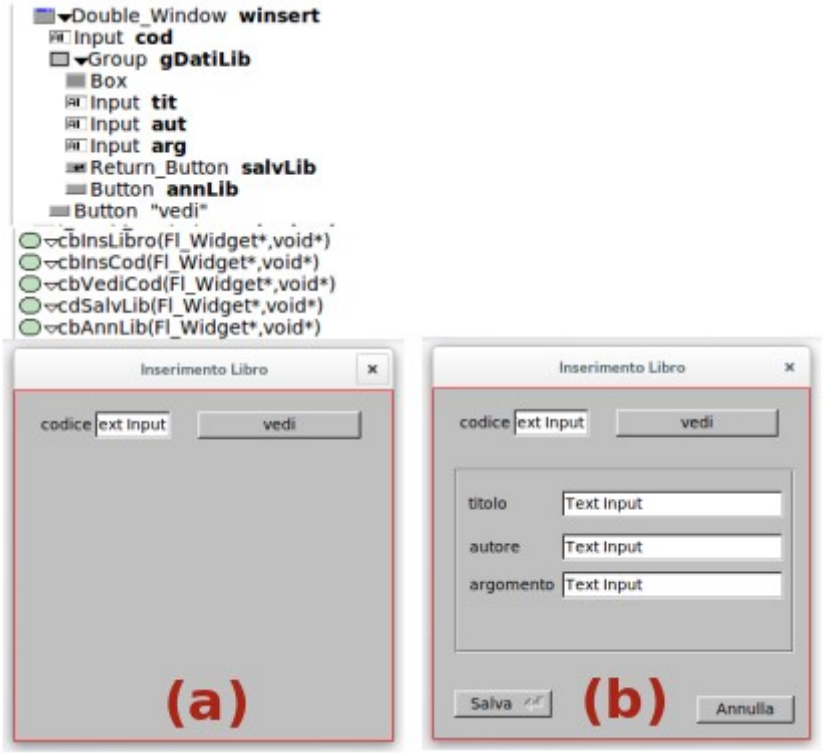
Nella finestra principale del programma è contenuto un widget `Menu_Bar` (sesto gruppo di pulsanti FLUID da sinistra, primo in alto a sinistra) che, a sua volta contiene un `Submenu` (cui è associata la label `Operazioni`) e due `Menuitem` (cui sono associate le label `Visualizza` ed `Esci`). I due widget si trovano nello stesso sesto gruppo del Widget Bin di FLUID e sono i pulsanti, rispettivamente, destra in basso e destra centro.

- La differenza fra `Submenu` e `Menuitem` consiste nel fatto che il primo contiene al suo interno altri `Menuitem` (*Inserisci libro* e *Prestito/Restituzione*).
- La `&` davanti ad una lettera indica l'abbreviazione che si può usare da tastiera, assieme al tasto <ALT>, per selezionare la voce del menu (shortcut). Per esempio `&Visualizza` si può selezionare, oltre che col mouse, con la combinazione di tasti <ALT>+v. La lettera risulterà in output sottolineata. La lettera maiuscola è utilizzata solo per mettere in evidenza l'abbreviazione.
- Non è necessario specificare esplicitamente lo shortcut se si tratta di `Submenu` o `Menuitem` contenuti in un `Submenu`. È necessaria l'esplicitazione (come evidenziato in figura), in *shortcut* della linguetta *GUI*, se si tratta di `Menuitem` da soli. Per ottenere ciò basta fare clic su *Shortcut* e poi digitare la combinazione di tasti.

Ad ogni voce che può essere selezionata dal menu è associata una callback il cui prototipo compare nel widget browser di FLUID ma il cui codice si trova nel file `biblioF.cxx` la cui inclusione si trova nell'ultima riga del Widget Browser. Le altre finestre associate alle operazioni previste non sono riportate nel grafico per permettere di concentrare l'attenzione sulla funzionalità trattata. Verranno trattate ognuna con la descrizione del codice associato alla funzionalità associata.

Chiudono la definizione della finestra principale due `box` (pulsante in alto a sinistra dell'ultimo gruppo del Widget Bin di FLUID): uno che contiene un immagine specificata in *Image* del tab *GUI* della finestra delle proprietà, un altro che contiene una *label* con il numero di versione del programma.

Inserisci Libro



La selezione dell'opzione *Inserisci Libro* dal menu *Operazioni* avvia la callback `cbInsLibro` che mostra una finestra (con nome `winsert`) in cui è possibile inserire il codice del libro. Se il codice non esiste vengono visualizzati gli input di testo dove inserire gli altri dati. Se il codice è già stato inserito è mostrata una finestra di avviso e si torna all'inserimento di un nuovo codice (il codice deve essere unico). Si può confermare l'inserimento e salvare i dati nel Database o annullare l'operazione e tornare alla finestra principale. Il pulsante *vedi* mostra una nuova finestra (la gestione è trattata nel prossimo paragrafo) con i codici dei libri già inseriti.

Nella finestra, come evidenziato dal *Widget Browser*, sono definiti un input di testo (`cod` pulsante in alto a sinistra del quinto gruppo nella *Widget Bin*) e un gruppo (`gDatiLib` pulsante centrale in alto nel secondo gruppo). Il gruppo contiene un box per il disegno del riquadro che contorna gli input, tre input (per l'inserimento di titolo, autore ed argomento del libro) e due pulsanti.

All'input del codice è associata la callback `cbInsCod` e ai due pulsanti, rispettivamente, le callback `cbSalvLib` (`Return_Button` *Salva*) e `cbAnnLib` (`Button` *Annulla*). Il `Button` *vedi*, fuori dal gruppo

è gestito dalla callback `cbVediCod`.

```
#include <FL/fl_ask.H>                                          /*1*/
#include <Database>                                             /*2*/
#include <string>
#include <vector>
using namespace std;
```

Il file `biblioF.cxx` nel quale sono codificate tutte le callback contiene, nelle prime righe, le inclusioni necessarie fra le quali la `1` per le finestre di avviso in caso, per esempio nella fase di inserimento di un libro, di un codice duplicato. L'inclusione `2`, come specificato in precedenza, contiene la definizione della classe per l'accesso al database sqlite3.

```
// ------------------------------ Inserisci Libro

void cbInsLibro(Fl_Widget* o,void*)
{
  gDatiLib->hide();                                             /*1*/
  cod->value("");                                               /*2*/
  winsert->show();                                              /*3*/
};
```

La callback associata alla scelta *Inserisci Libro* della voce *Operazioni* del menu della finestra principale inibisce la visualizzazione del gruppo in cui inserire i dati del libro (`1`), inizializza il codice (`2`) e mostra la finestra per la registrazione del libro (`3`).

Nella finestra attivata **(a)** è visibile, per il momento solo l'Input per il codice (`cod`) e un pulsante (*vedi*) per la visualizzazione dei codici già inseriti. Al widget `cod` è associata la callback `cbInsCod` e al pulsante `cbVediCod`. La `cbInsCod` viene avviata quando il focus esce dal controllo (digitando il tasto <TAB>).

```
// controlla codice e permette inserimento dati

void cbInsCod(Fl_Widget* o,void* v)
{
  Database *db;                                                 /*1*/
  string q, temp;
  vector<vector<string> > result;                               /*2*/

  db = new Database("biblio.sqlite");                           /*3*/

  // verifica non esistenza codice per proseguire

  q = "SELECT codice FROM libri ";                              /*4*/
  q += "WHERE codice=\'"+temp.assign(cod->value())+"\'";        /*5*/
  result = db->query(q);                                        /*6*/

  db->close();

  if(result.empty()){
    gDatiLib->show();                                           /*7*/
    tit->value("");
    aut->value("");
    arg->value("");
    tit->take_focus();
  }
  else{
```

```
      fl_alert("Codice Esistente");                                    /*8*/
      cod->take_focus();
    };
  };
```

In 1 si dichiara un puntatore ad oggetto della classe `Database` e nella 3 si inizializza l'oggetto puntato al file che contiene il database. In 2 si dichiara la variabile che conterà il risultato della query.

Dalla 4 si prepara una stringa che conterrà la query da effettuare sul database. Si cerca, se esiste, il codice inserito (5). Tale codice essendo una stringa deve essere racchiuso da apici.

Nella 6 si effettua la query che restituisce la tabella risultato in `result`. Se la tabella è vuota (codice non inserito) si visualizza il gruppo contenete gli `Input` degli altri dati da inserire (7), altrimenti viene segnalata (8) una situazione di errore e si richiede di digitare un nuovo codice.

Se il codice è unico la finestra assume l'aspetto **(b)** con, oltre gli input, i pulsanti *Salva* e *Annulla* cui sono associate le callback `cbSalvLib` e `cbAnnLib`.

```
// salva dati libro nel DB

void cbSalvLib(Fl_Widget* o,void*v)
{
  Database *db;
  string q, temp;
  db = new Database("biblio.sqlite");

  // salva dati nel DB

  q  = "INSERT INTO libri ";                                           /*1*/
  q += "VALUES(\'"+temp.assign(cod->value())+"\',";
  q += "\'"+temp.assign(tit->value())+"\',";
  q += "\'"+temp.assign(aut->value())+"\',";
  q += "\'"+temp.assign(arg->value())+"\',";
  q += "1,";                                                           /*2*/
  q += "\'\',\'\',\'\')";                                              /*3*/

  db->query(q);
  db->close();

  // aggiorna Inseriti se visualizzata

  if(winelcod->shown())                                                /*4*/
    cbVediCod(o,v);                                                    /*5*/

  // prossimo libro

  cbInsLibro(o,v);                                                     /*6*/
};
```

Nella callback si prepara (1) la query dichiarando la disponibilità del libro (2) e inizializzando i dati del socio che prende in prestito il libro (3).

Se la finestra che mostra i codici dei libri già inseriti è visualizzata (4) si rilancia la callback che la gestisce (5) in modo che l'elenco sia aggiornato in tempo reale con il nuovo inserimento.

Alla fine (6) ci si prepara per un nuovo inserimento. Devono essere passati due parametri: gli stessi

che sono passati al metodo attuale e che, di fatti, non sono mai utilizzati facendo riferimento sempre per come si è impostato il progetto, a variabili globali.

```
// annulla/chiude registrazione libri

void cbAnnLib(Fl_Widget* o,void*)
{
  winsert->hide();                                                    /*1*/
  wmain->take_focus();
};
```

Il pulsante *Annulla* ha l'effetto di nascondere (`1`) la finestra di Inserimento Libro e ritornare alla finestra principale.

Visualizza codici esistenti

La pressione del pulsante *vedi* della finestra di registrazione di un libro comporta la visualizzazione di una nuova finestra che mostra l'elenco dei codici, e dei titoli associati, già inseriti. La finestra contiene solo un `Text_Display` e, per come trattato in un esempio precedente (cppGUI), deve essere definito un widget `Text_Buffer` associato ad esso.

```
// buffer elenco codici e titoli

buf = new Fl_Text_Buffer();
dispcod->buffer(buf);
```

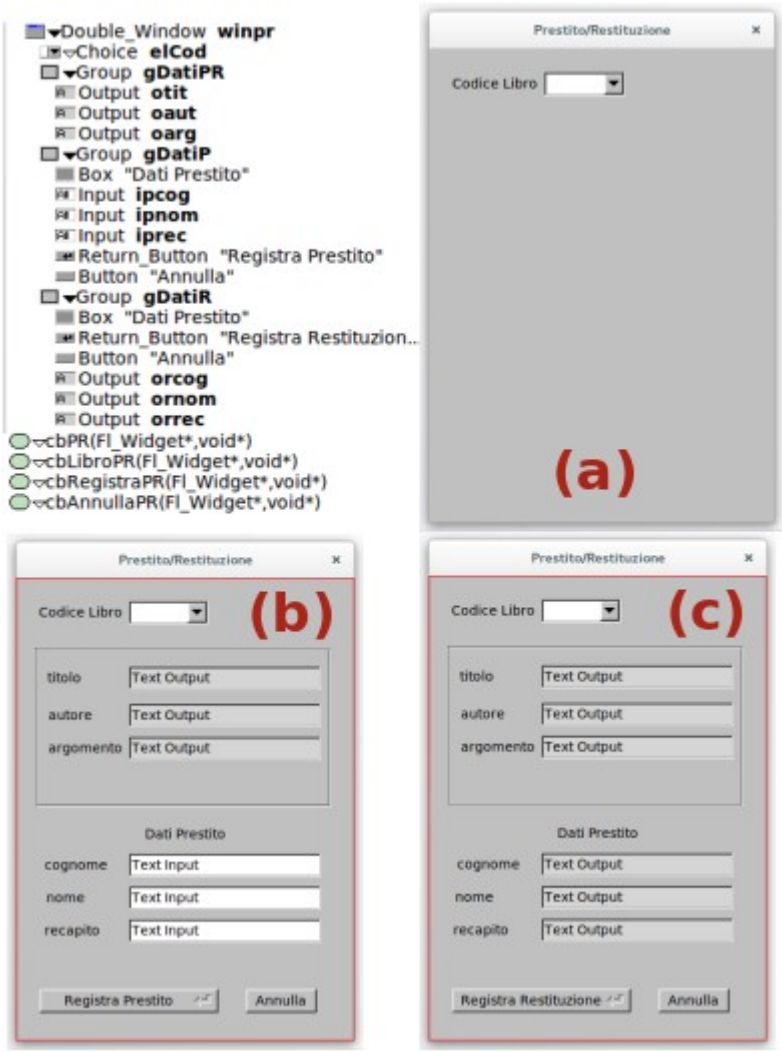
il codice è inserito utilizzando il widget `Code` direttamente da FLUID.

```
// vedi codici libri inseriti

void cbVediCod(Fl_Widget* o,void*)
{
  Database *db;
  string q, temp;
  vector<vector<string> > result;
  vector<vector<string> >::iterator it;
  vector<string> row;
  char s[50];

  winelcod->show();

  // recupera cod e titoli dal DB
```

```
    db = new Database("biblio.sqlite");
    q  = "SELECT codice,titolo FROM libri ";                          /*1*/
    q += "ORDER BY codice";
    result = db->query(q);
    db->close();

    // inserisci nel buffer

    buf->text("");
    for (it=result.begin(); it!=result.end(); it++) {
      row = *it;
      temp = row.at(0)+" "+row.at(1);                                 /*2*/
      strcpy(s,temp.c_str());                                         /*3*/
      buf->append(s);                                                 /*4*/
      buf->append("\n");                                              /*4*/
    };

};
```

Si effettua sul database la query per rintracciare codice e titolo dei libri inseriti (1), si mettono assieme in un unica stringa i due dati (2), si trasforma la stringa C++-type in stringa C-type (3) e si aggiunge, per la visualizzazione, la riga al buffer (4).

Prestito/Restituzione



La selezione dell'opzione *Prestito/Restituzione* dal menu *Operazioni* avvia la callback `cbPR` che mostra una finestra (con nome `winpr`) con inizialmente **(a)** un menu di scelta a scomparsa (`elCod`) da cui si può selezionare il codice del libro su cui effettuare l'operazione. Le due operazioni sono simili: se il campo `disponibile` della tabella `libri` ha valore `1` si vuole effettuare un prestito, se ha valore `0` una restituzione.

Nella finestra è definito un gruppo `gDatiPR` in cui sono contenuti i dati del libro e che viene visualizzato in seguito alla scelta del codice. Se il libro è disponibile la finestra assume l'aspetto **(b)** con la visualizzazione del gruppo `gDatiP` predisposto per la registrazione del prestito dopo aver riempito i campi con i dati del socio. Se il libro non è disponibile vuol dire che si tratta di una restituzione e la finestra assume l'aspetto **(c)** con la visualizzazione del gruppo `gDatiR`.

La callback `cbLibroPR` è associata alla scelta del codice dal menu a discesa. Al pulsante *Registra Prestito/Restituzione* è associata `cbRegistraPR` a prescindere dal tipo di operazione, al pulsante *Annulla* `cbAnnullaPR`.

```
// ----------------------------- Prestito/Registrazione

void cbPR(Fl_Widget* o,void* v)
```

```
{
  Database *db;
  string q, temp;
  vector<vector<string> > result;
  vector<vector<string> >::iterator it;
  vector<string> row;
  char c[10];

  gDatiPR->hide();                                                    /*1*/
  gDatiP->hide();
  gDatiR->hide();
  winpr->show();

  // recupera cod dal DB

  db = new Database("biblio.sqlite");
  q  = "SELECT codice FROM libri ";                                   /*2*/
  q += "ORDER BY codice";
  result = db->query(q);
  db->close();

  // inserisci nel menu a discesa

  elCod->clear();
  for (it=result.begin(); it!=result.end(); it++) {
    row = *it;
    temp = row.at(0);
    strcpy(c,temp.c_str());
    elCod->add(c);                                                    /*3*/
  };
};
```

`cbPR` viene avviata dalla scelta dell'operazione dal menu. La funzione si occupa di rendere invisibili i gruppi e di visualizzare la finestra dell'operazione (`1`). Si recuperano dal database i codici introdotti (`2`) e si inseriscono come scelte nel menu (`3`) dopo averle trasformate in stringhe C-type.

```
struct libro {
  char titolo[50];
  char autore[50];
  char argomento[50];
  char disponibile[3];
  char cognome[20];
  char nome[20];
  char recapito[50];
};

// operazione sul libro selezionato

void cbLibroPR(Fl_Widget* o,void* v)
{
  Database *db;
  string q, temp;
  libro lib;
  vector<vector<string> > result;
  vector<vector<string> >::iterator it;
  vector<string> row;

  // ci deve essere il codice
```

```
    if(elCod->value()<0){                                              /*1*/
      fl_alert("E\' necessario scegliere un codice");
      return;
    };

    // rintraccia libro

    db = new Database("biblio.sqlite");
    q  = "SELECT * ";
    q += "FROM libri ";
    q += "WHERE codice=\'"+temp.assign(elCod->text())+"\'";             /*2*/

    result = db->query(q);
    db->close();

    it = result.begin();                                               /*3*/
    row = *it;

    strcpy(lib.titolo,row.at(1).c_str());                              /*4*/
    strcpy(lib.autore,row.at(2).c_str());
    strcpy(lib.argomento,row.at(3).c_str());

    // dati comuni a P e R

    otit->value(lib.titolo);                                           /*5*/
    oaut->value(lib.autore);
    oarg->value(lib.argomento);
    gDatiPR->show();                                                   /*6*/

    // distingue se P o R

    strcpy(lib.cognome,row.at(5).c_str());
    strcpy(lib.nome,row.at(6).c_str());
    strcpy(lib.recapito,row.at(7).c_str());

    if(row.at(4)=="1"){                                                /*7*/
      ipcog->value(lib.cognome);
      ipnom->value(lib.nome);
      iprec->value(lib.recapito);
      gDatiP->show();                                                  /*8*/
      ipcog->take_focus();
    }
    else{
      orcog->value(lib.cognome);
      ornom->value(lib.nome);
      orrec->value(lib.recapito);
      gDatiR->show();                                                  /*8*/
    };
  };
```

La callback prepara il libro associato al codice all'operazione.

Per prima cosa si controlla che il codice (1) sia stato inserito. Si ricerca la registrazione del libro cui si è selezionato il codice (2) che, in ragione dell'unicità del codice, è unico (3). Si trasformano le informazioni C++-type in stringhe C-type (4) e si copiano nei controlli del gruppo comune alle due operazioni (5) che successivamente viene visualizzato (6).

Nella 7 si distingue, in base al valore di disponibile, se si tratta di prestito o restituzione e di conseguenza si stabilisce il gruppo, definito nella finestra, da visualizzare (8).

```
// registra operazione

void cbRegistraPR(Fl_Widget* o,void *v)
{
  Database *db;
  string q, temp;

  if(!strcmp(ipcog->value(),"") && !strcmp(ipnom->value(),"") &&     /*1*/
     !strcmp(iprec->value(),"")){
    fl_alert("Almeno uno dei dati del socio deve essere inserito");
  }
  else{

    // registra dati operazione

    db = new Database("biblio.sqlite");

    if(gDatiP->visible()){     // prestito                          /*2*/
      q  = "UPDATE libri ";
      q += "SET disponibile = 0, ";
      q += "cognome=\'"+temp.assign(ipcog->value())+"\',";
      q += "nome=\'"+temp.assign(ipnom->value())+"\',";
      q += "recapito=\'"+temp.assign(iprec->value())+"\' ";
      q += "WHERE codice=\'"+temp.assign(elCod->text())+"\'";
    }
    else{                      // restituzione                      /*2*/
      q  = "UPDATE libri ";
      q += "SET disponibile = 1, ";
      q += "cognome=\'\',";
      q += "nome=\'\',";
      q += "recapito=\'\' ";
      q += "WHERE codice=\'"+temp.assign(elCod->text())+"\'";
    };
    db->query(q);
    db->close();
  };

  cbPR(o,v);
};
```

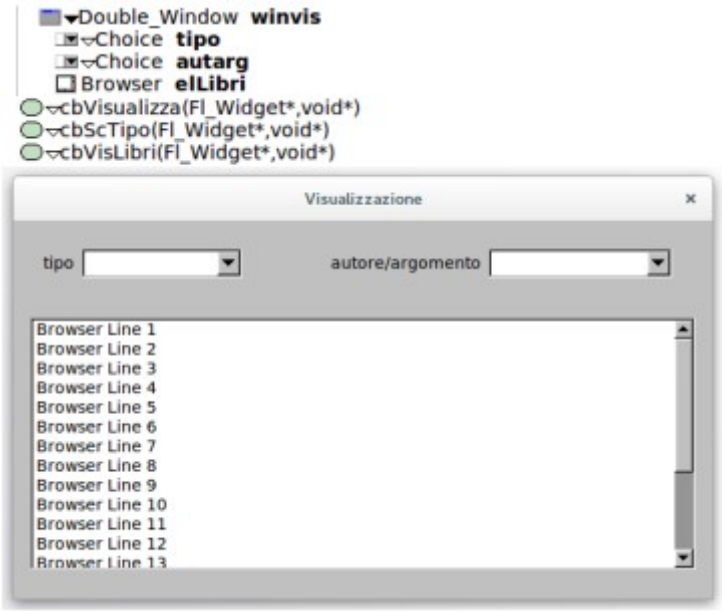
cbRegistraPR gestisce la selezione della registrazione. Almeno uno dei dati del socio deve essere presente (1): se si tratta di un prestito ci deve essere almeno un dato del destinatario del prestito, se è una restituzione, appunto per le 1, ci sarà almeno un dato e la finestra alert varrà solo in caso di operazione di prestito.

La distinzione del tipo di operazione è fatto in base alla condizione di visibilità di un gruppo (gDatiP in 2). Se prestito si registrano i dati del socio, altrimenti si inizializzano e viene reso nuovamente disponibile il libro.

```
void cbAnnullaPR(Fl_Widget* v,void* o)
{
  winpr->hide();
  wmain->take_focus();
};
```

La callback associata al tasto *Annulla* a prescindere dal tipo di operazione, nasconde la finestra dell'operazione e restituisce il focus alla finestra principale.

Visualizza



La selezione della opzione *Visualizza* del menu della finestra principale avvia la callback `cbVisualizza` che attiva la finestra `winvis` da cui si può scegliere il tipo di visualizzazione. Dal menu di scelta *tipo* si può scegliere se visualizzare tutti i libri esistenti o effettuare una selezione per autore o argomento. Negli ultimi due casi si attiva il menu di scelta *autore/argomento* che permette di scegliere fra gli autori/argomenti. Il `Browser` sottostante mostra l'elenco dei libri che soddisfano i requisiti. La callback `cdScTipo` gestisce la scelta del codice e, l'eventuale possibilità di scegliere l'autore o l'argomento.

`cbVisLibri` viene attivata dalla scelta dell'autore/argomento o dalla prima scelta del tipo di visualizzazione.

```
// browser visualizzazione libri

int colonne[]={50,200,160,160,10,160,160,250};
elLibri->column_widths(colonne);
elLibri->column_char('\t');
elLibri->type(FL_MULTI_BROWSER);
```

Il precedente codice, come riportato anche in un altro esempio (*cppGUI*), è inserito in FLUID per specificare le caratteristiche del browser.

```
// ------------------------------ Visualizza

void cbVisualizza(Fl_Widget* v,void* o)
{
  autarg->hide();                                                    /*1*/
  elLibri->clear();
  winvis->show();

  tipo->clear();                                                     /*2*/
  tipo->add("tutti");
```

```
  tipo->add("per Autore");
  tipo->add("per Argomento");
};
```

La callback avviata dalla selezione *Visualizza* del menu della finestra principale, disabilita la visualizzazione del menu di scelta autore/argomento (`1`), pulisce il contenuto del browser e visualizza la finestra. Da `2` si inseriscono le possibilità di scelta sul tipo di visualizzazione.

```
// scelta tipo visualizzazione

void cbScTipo(Fl_Widget* v,void* o)
{
  Database *db;
  string q;
  char riga[50];
  vector<vector<string> > result;
  vector<vector<string> >::iterator it;
  vector<string> row;

  elLibri->clear();
  autarg->clear();

  if(!tipo->value()){                                          /*1*/
    autarg->hide();                                            /*2*/
    cbVisLibri(v,o);                                           /*3*/
  }
  else{
    autarg->clear();                                           /*4*/
    autarg->redraw();                                          /*5*/
    autarg->show();                                            /*6*/
  };

  // distingue i due casi

  db = new Database("biblio.sqlite");
  switch(tipo->value()){
  case 1:                                                      /*7*/
    q  = "SELECT DISTINCT autore FROM libri ";
    q += "ORDER BY autore";
    break;
  case 2:                                                      /*7*/
    q  = "SELECT DISTINCT argomento FROM libri ";
    q += "ORDER BY argomento";
    break;
  };
  result = db->query(q);
  db->close();

  for (it=result.begin(); it!=result.end(); it++) {
    row = *it;
    strcpy(riga,row.at(0).c_str());
    autarg->add(riga);                                         /*8*/
  };
};
```

In conseguenza alla scelta del tipo di visualizzazione viene avviata `cbScTipo`. Se viene scelta la prima opzione (`1`) (*tutti* con `value 0`) si inibisce la visualizzazione della scelta successiva (`2`) e viene avviata la funzione (`3`) che come callback gestisce le azioni associate alle scelte.

Le altre scelte portano ad inizializzare il menu di scelta (4), ridisegnare il menu (5) per ripulirlo da eventuale scelta precedentemente visualizzata e renderlo accessibile (6).

In relazione al tipo di visualizzazione scelta si reperiscono i dati opportuni (7) dal database e si riempie (8) il menu di scelta.

```
// visualizza selezione libri

void cbVisLibri(Fl_Widget* v,void* o)
{
  Database *db;
  string q,temp;
  char riga[250];
  int i;
  vector<vector<string> > result;
  vector<vector<string> >::iterator it;
  vector<string> row;

  elLibri->clear();

  // query in base alle scelte effettuate

  db = new Database("biblio.sqlite");
  q  = "SELECT * FROM libri ";                                    /*1*/
  switch(tipo->value()){
  case 1:                                                         /*2*/
    q  += "WHERE autore=\'"+temp.assign(autarg->text())+"\' ";
    break;
  case 2:                                                         /*2*/
    q  += "WHERE argomento=\'"+temp.assign(autarg->text())+"\' ";
    break;
  };
  q  += "ORDER BY codice";                                        /*1*/
  result = db->query(q);                                          /*3*/
  db->close();

  // riempimento browser

  for (it=result.begin(); it!=result.end(); it++) {               /*4*/
    row = *it;
    temp = row.at(0);
    for (i = 1; i <= 7; i++) {
      temp += "\t"+row.at(i);                                     /*5*/
    };
    strcpy(riga,temp.c_str());
    elLibri->add(riga);                                           /*6*/
  };
};
```

La callback `cbVisLibri` è associata al widget `Choice autarg` o viene, come esaminato prima, richiamata esplicitamente, come funzione, se la scelta del tipo riguarda tutti i libri.

Si imposta la parte comune della query (1) e in aggiunta (2) la parte dipendente dalla scelta sul tipo di visualizzazione. Si effettua la query (3) e per ogni riga (4) del risultato si costruisce (5) la stringa che, trasformata in C-type, sarà aggiunta (6) al browser.

Esci

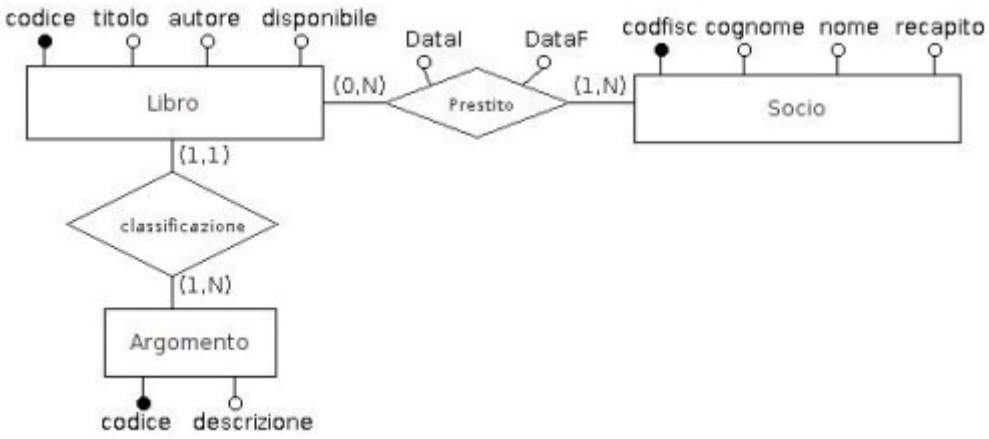
La callback `cbEsci` è associata sia alla scelta *Esci* del menu della finestra principale che alla finestra `wmain` stessa.

```
// ------------------------------ Esci

void cbEsci(Fl_Widget* o,void*)
{
  exit(0);
};
```

La callback che contiene solo lo statement per la chiusura del programma è necessaria perché il programma prevede l'apertura di più finestre e se si chiude una finestra esplicitamente con il pulsante di controllo, questo ha effetto solo sulla finestra corrente. Qui invece si vuole che eventuali finestre ancora aperte vengono tutte chiuse sia se si sceglie *Esci* dal menu che se si chiude la finestra principale.

Possibili estensioni

Lo scopo di questi appunti è quello di mostrare come mettere assieme più librerie: utilizzare interfacce GUI e usare C++ e le estensioni SQL come un linguaggio gestionale. Il database è qui ridotto al minimo utilizzando una unica tabella.



Sarebbe stato più corretto utilizzare il database il cui diagramma E-R avesse previsto più entità anche se tale scelta avrebbe comportato soltanto complicazioni nelle query senza aggiungere altre competenze alla progettazione complessiva del programma.

Ulteriori estensioni del programma potrebbero prevedere per esempio, oltre ad una organizzazione del database diversa, la gestione dei soci: registrazione, possibilità di visualizzare tutti i libri presi in prestito, i libri che sono stati dati in prestito e non restituiti dopo una certa quantità di tempo, la classifica dei libri più letti ecc...

Riferimenti

- `https://www.sqlite.org/` il sito del motore di database SQLite in cui consultarne anche la documentazione
- `http://www.dreamincode.net/forums/topic/122300-sqlite-in-c/` sito da cui effettuare il download del codice della classe `Database` e del programma di esempio riportato nel primo paragrafo di questi appunti
- in questi appunti si fa riferimento a concetti trattati in altri appunti (C++, cppGUI, DB&SQL) disponibili in download da `http://ennebi.solira.org`.