cppGUI

interfacce utente grafiche in C++: FLTK e FLUID

2013.07

Indice

Programmazione di interfacce grafiche

I moderni sistemi di elaborazione con la potenza messa a disposizione permettono l'esecuzione di programmi che interagiscono con l'utente, in maniera molto semplificata, per mezzo dell'utilizzo di controlli grafici (pulsanti, caselle per l'inserimento di testo, menù a discesa …). Dal punto di vista del programmatore progettare una applicazione che interagisce con l'utente mediante una interfaccia grafica vuol dire tenere conto, a differenza di una applicazione che gira in un terminale, di un paradigma diverso: l'avvio delle azioni previste dal programma dipende dalle azioni compiute dall'utente. L'utente interagisce con i controlli grafici (per es. preme un pulsante) e questo evento avvia l'esecuzione di una serie di azioni associate all'evento (programmazione *event driven*). Il paradigma prevede la scrittura di opportune parti di codice (funzioni **Callback**) associate, per esempio, all'evento generato in seguito alla pressione di un pulsante.

L'affermarsi della programmazione ad oggetti ha consentito di costruire insiemi (i **toolkit**) di elementi grafici, funzioni e/o classi, che possono essere utilizzati per semplificare la programmazione di una GUI (Graphic User Interface): il programmatore ha a disposizione una serie di funzionalità già pronte che possono essere utilizzate per realizzare finestre, tab, pulsanti personalizzati. Il mondo del Software Libero mette a disposizione parecchi toolkit che hanno anche un ulteriore vantaggio: la disponibilità dei codici sorgenti ne permette l'utilizzo come base per lo sviluppo di software per piattaforme diverse (oltre Linux tipicamente Windows, MacOSX) senza preoccuparsi di modificare il sorgente. Si sviluppa una sola volta e, ricompilando, si ha a disposizione la versione, dello stesso programma, che gira su quel sistema operativo. La quasi totalità dei toolkit disponibili per ambiente Linux, e per vari linguaggi di programmazione, sono disponibili anche per l'installazione su altri S.O.

In questi appunti si introduce, con esempi di uso, il toolkit **FLTK** (si legge, come suggerito dall'autore, *fulltick* ed è acronimo di Fast Light ToolKit). Si tratta di un toolkit disponibile per tutte le piattaforme già elencate in precedenza, facile da apprendere, che permette notevole velocità di sviluppo e che basa i suoi punti di forza sulle caratteristiche di Fast (velocità) e Light (leggerezza).

La velocità è ottenuta utilizzando in modo massiccio puntatori a classi che consentono di avere programmi reattivi ed evitando di appesantire la libreria definendo funzionalità che non hanno a che fare con la costruzione di GUI (per es. gestione stringhe o altro). Gli elementi grafici sono implementati per mezzo di classi, e spesso sono utilizzate caratteristiche del linguaggio C per sfruttarne la velocità.

La leggerezza è ottenuta includendo, nell'oggetto risultante dalla compilazione, solo le funzioni di libreria utilizzate effettivamente nell'applicazione sviluppata e non tutto il toolkit. Il programma compilato non necessita dell'installazione, nella macchina di destinazione, di librerie per poter girare: tutto quello che serve è già incluso nell'oggetto.

Un esempio indicativo di applicazione concreta delle doti di velocità e leggerezza di FLTK si può trovare nello sviluppo di un completo ambiente desktop, che gira anche in ambiente Linux, e che permette di utilizzare interfacce grafiche anche in sistemi hardware poco performanti e con poca RAM: EDE Equinox Desktop Environment.

Nell'immagine EDE gira in ambiente Minix.

L'installazione delle librerie comprende anche **FLUID** (Fast Light User Interface Designer. Si avvia con il comando: `fluid`) un applicativo che permette di disegnare la GUI di un programma, inserendo nella progettazione dell'interfaccia gli elementi grafici (**Widgets**) semplicemente selezionandoli da una finestra che ne mostra l'elenco. Le personalizzazioni del widget possono essere effettuate tramite la finestra delle proprietà associata al widget stesso.

Il programma genera i sorgenti pronti per la compilazione. Si possono inserire anche ulteriori frammenti di codice utilizzando FLUID anche se, non essendo questo un ambiente di sviluppo, non è provvisto delle funzionalità di un editor per programmatori. Si tratta, appunto, di un Designer per interfacce grafiche.

Si suggerirà in seguito un modo come utilizzarlo assieme, per esempio, ad Emacs.

Una caratteristica di FLUID è che non genera tonnellate di codice sorgente di difficile comprensione e utili solo a se stesso come avviene per altri ambienti, ma soltanto il file sorgente e il file di intestazione del progetto su cui si lavora. Se il progetto si chiama `prova.fl` (`fl` è l'estensione utilizzata da FLUID), gli altri file generati saranno: `prova.cxx` (utilizza per default l'estensione `cxx`: una delle estensioni standard dei file sorgenti C++) e `prova.h`.

Nel toolkit è disponibile inoltre un comodo comando per la compilazione del progetto che provvede anche all'aggancio delle librerie necessarie.

```
fltk-config --compile prova.cxx
```

l'esecuzione della linea di comando provoca la generazione dell'eseguibile `prova` nella stessa directory dove si trova il sorgente e quindi in definitiva si avranno, oltre all'oggetto eseguibile: `prova.fl` (il sorgente per FLUID che contiene il disegno dell'interfaccia), `prova.cxx` (il programma principale con il codice per l'eseguibile), `prova.h` (l'intestazione con le inclusioni necessarie).

Da C a C++ e viceversa

Il toolkit FLTK implementa un ampio insieme di classi (principalmente) e funzioni necessarie per la costruzione di interfacce grafiche ma per il resto si appoggia alle funzionalità disponibili nel linguaggio C/C++. Per esempio le stringhe che vengono spesso utilizzate come parametri nei metodi delle classi, sono definite alla maniera del C. Una stringa in C è un vettore di `char` e il nome rappresenta un puntatore al primo carattere e in questo modo si può trattare in modo più veloce di quanto lo si possa, per esempio, come oggetto della classe `string`, come nel C++, anche se dal punto di vista della facilità della scrittura del codice un oggetto della classe `string` permette, con i metodi che rende disponibili, elaborazioni più comode.

In questo paragrafo si tratterà della gestione delle stringhe in C in modo da avere gli strumenti per le elaborazioni più comuni che possono essere effettuate con i mezzi disponibili delle librerie C. Qualora occorressero elaborazioni che potessero essere svolte più facilmente per mezzo dei metodi della classe `string` del C++ si vedrà un esempio di codice per la conversione da un tipo all'altro.

Le stringhe in C sono vettori di tipo carattere la cui fine è segnalata da un carattere `NULL` (carattere terminatore), indicato come `'\0'`. Il carattere `NULL` è il primo codice ASCII corrispondente al valore binario `00000000` e non ha niente a che vedere con il carattere `0` che ha, in ASCII, codice binario `00110000`.

```
char a[10];
```

dichiara un vettore costituito da un massimo di dieci caratteri e:

```
char frase[] = "Oggi non piove";
```

dichiara il vettore monodimensionale `frase` il cui numero di caratteri è determinato dalla quantità di caratteri presenti fra doppi apici più uno (il carattere `NULL` che chiude la stringa).

Nonostante il linguaggio C non possieda un dato di questo tipo, per il trattamento delle stringhe sono disponibili parecchie funzioni, delle quali in questa sede vengono trattate quelle che possono essere interessanti per le elaborazioni proposte.

- **`strcpy`**. Questa funzione copia una stringa in un'altra. Può essere utilizzata, per esempio, per assegnare un valore a una stringa. La sua sintassi è:

  ```
  strcpy(s1,s2);
  ```

  Si può immaginare che tale funzione equivale, come effetto, all'assegnamento nelle variabili di altro tipo (in altri termini è come se si scrivesse: `s1=s2`): assegna, cioè, alla stringa `s1` il valore contenuto in `s2`. Il secondo parametro può anche essere direttamente una stringa racchiusa tra doppi apici.

- **`strcat`** e **`strcmp`**. La funzione:

  ```
  strcat(s1,s2);
  ```

  concatena la stringa `s2` alla fine della stringa `s1`:

  ```
  strcpy(s1,"buon");      // stringa specificata in s1
  strcpy(s2,"giorno");    // lo stesso per s2
  strcat(s1,s2);          // s1 ora conterrà "buongiorno"
  ```

  La funzione:

```
strcmp(s1,s2);
```

può essere utilizzata per effettuare comparazioni sul contenuto di due stringhe. In particolare tale funzione:

Restituisce un valore positivo se vale `s1>s2`

Restituisce un valore negativo se vale `s1<s2`

Restituisce `0` se `s1` ed `s2` sono uguali

Per tenere conto mnemonicamente di tali valori, basta pensare al confronto come ad una sottrazione: fra l'altro ciò non è molto distante da quello che avviene in effetti. Se da `s1` si sottrae `s2` si avrà un valore positivo nel caso `s1>s2`, negativo se `s1<s2` e nullo se sono uguali

Le funzioni esaminate per il trattamento delle stringhe richiederebbero, per poter essere utilizzate, l'inclusione della libreria che le contiene (`#include <cstring>`) ma tale libreria è già inclusa nei file di definizione di FLTK e, quindi, si possono utilizzare, laddove servono, senza necessità di specificare l'inclusione.

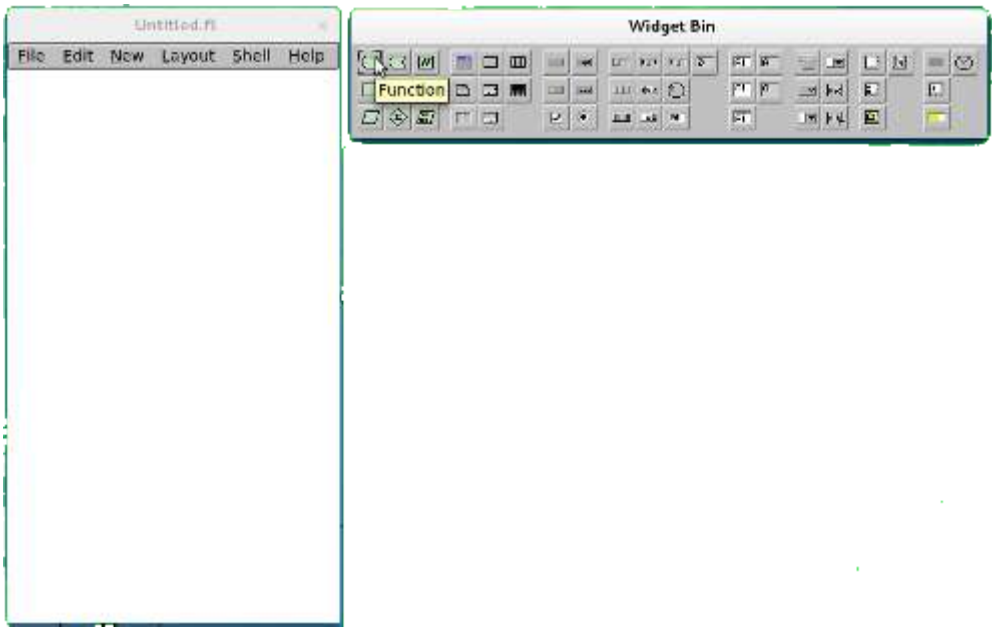
Potrebbe essere utile convertire un dato da un formato ad un altro.

```
// conversione da char[] del C (stringa C type)
// a string del C++ (stringa C++ type)

char a[] = "Prova conversione in string";
string b;

b.assign(a);   // anche b conterrà ora la stessa stringa

// conversione inversa da string a char*

char s[20];
string d = "Al contrario";

strcpy(s,d.c_str());   // ora s contiene la stessa stringa

// da float/double o int a char*

float e=12345.12;
char g[10];

int h=67;
char j[10];

sprintf(g,"%4.2f",e); // g conterrà il valore float trasformato
sprintf(j,"%d",h);    // j conterrà il valore int trasformato

// da char* a float/double o int
// normalmente richiederebbero #include <cstdlib> ma
// la libreria è già inclusa nei file di definizione delle FLTK

char intero[]="123";
char puntoDec[]="456.78";
int valI;
float valF;
```

```
valI = atoi(intero);    // valore numerico
valF = atof(puntoDec); // valore numerico
```

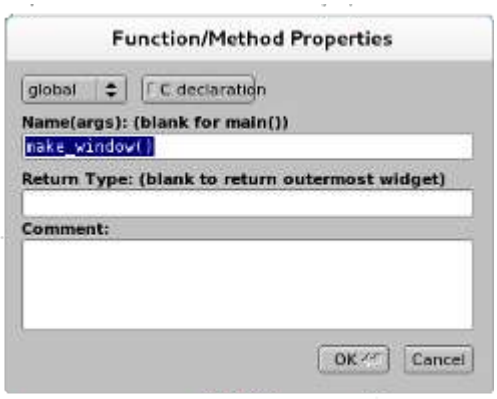
Il codice per la presenza di righe di commento esplicative non richiede ulteriori specificazioni.

Utilizzo di FLUID



All'avvio il programma presenta una finestra a sinistra (*Widget Browser*) in cui saranno elencati, e da cui si possono selezionare, i widget utilizzati nell'interfaccia del programma che si sta progettando. La finestra di destra (*Widget Bin*) presenta in formato grafico l'elenco dei più comuni elementi grafici che possono essere inclusi nel progetto selezionandoli: se si passa il mouse sopra ogni singolo componente viene visualizzata la classe dell'oggetto che potrà essere inserito, tipicamente dentro una finestra, alla pressione del tasto. Gli elementi possono essere inclusi anche scegliendo l'opportuna voce dal menù *New* della *Widget Browser*.

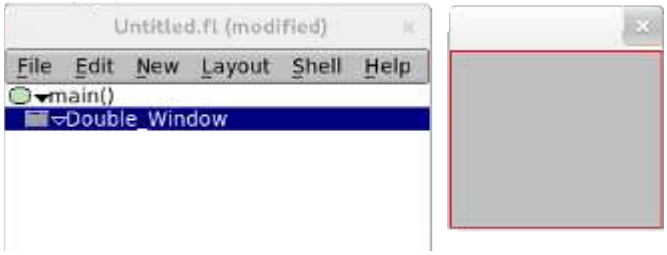
Per cominciare lo sviluppo di una nuova applicazione è necessario prima specificare la funzione che conterrà il codice degli oggetti grafici. La funzione si può selezionare dal primo pulsante in alto a sinistra della finestra *Widget Bin* oppure selezionandola dal menù *New, Code, Function/Method*.



Come suggerito all'interno della finestra che viene proposta in seguito alla scelta dello strumento *Function*, si può cancellare il nome proposto per fare in modo che la funzione dove verranno inseriti

i widget sia `main`.

Ora si può scegliere il widget *Window* dalla *Widget Bin* (quarto da sinistra nella riga in alto) oppure scegliendo dal menù *New, Group, Window*:



nell'elenco la finestra compare *dentro* il `main` e, inoltre essendo selezionata, la window compare circondata da un sottile bordo rosso ad indicare la selezione.

Un widget selezionato si può ridimensionare trascinando i bordi con il mouse, inoltre se si clicca due volte con il pulsante sinistro del mouse, viene visualizzata una finestra con tre schede da cui impostare le proprietà del widget. La finestra è sempre la stessa per tutti gli elementi grafici, perché riguarda le proprietà necessarie alla parte della costruzione dell'interfaccia grafica.

Nel tab *GUI* la proprietà più importante può essere *Label* che permette di attribuire una etichetta al widget: in questo caso l'etichetta `Prova` verrà visualizzata nella barra del titolo della finestra.

Le proprietà di posizionamento e larghezza/altezza possono essere impostate, oltre che da qui, trascinando con il mouse i bordi della selezione nel widget.

La *Tooltip* può essere utilizzata qualora si voglia far comparire, quando il mouse si sposta sul widget, un suggerimento sul significato o l'utilizzo del widget. Sono le tipiche finestrelle, generalmente in colore giallo, che compaiono quando il mouse si sposta sopra il controllo.

Con il tab *Style* si possono impostare i colori, il font e le caratteristiche grafiche del widget.

Le proprietà più importanti sicuramente sono raggruppate nel tab *C++*. Principalmente interessano *Name* (il nome che verrà utilizzato nel codice per riferirsi al widget) e *Callback* (il posto in cui si inserirà il nome della funzione che dovrà gestire gli eventi sul widget, per esempio la funzione che gestirà il clic del mouse su un pulsante). In questo primo approccio non verrà inserito niente.

Dal menù *File*, *Save As* si sceglie una directory dove salvare i file e il nome, per esempio: `prova.fl`. L'estensione `fl` deve essere specificata. Se inoltre si seleziona *File, Write Code* vengono salvati nella stessa directory scelta in precedenza, `prova.cxx` e `prova.h`. A questo punto si può compilare il programma:

```
fltk-config --compile prova.cxx
```

e si avrà l'eseguibile `prova`, generato usando soltanto i passaggi descritti, e che lanciato visualizza una finestra con titolo *Prova*. La finestra si chiude se si preme l'apposito pulsante di chiusura o se si preme il tasto *Esc*.

Sorgenti generati da FLUID

Mano a mano che si inseriscono oggetti grafici è possibile vedere i sorgenti generati. Selezionando dal menù *Edit, Show Source Code* viene mostrata una finestra con due tab che permettono di scegliere la visualizzazione del codice sorgente (tab *Source*) o quella del file di intestazione (tab *Header*). Il codice evidenziato è quello che traduce l'elemento grafico selezionato nella *Widget Browser* di FLUID.

L'esame del contenuto dei due più importanti, per il programmatore, file (`prova.h` e `prova.cxx`) permette di avere una idea generale, utile per quando si costruiranno i programmi, dell'uso di FLTK. Il terzo file (`prova.fl`), anche questo file di testo, è importante solo per FLUID perché gli permette di ricostruire, in un secondo tempo, la GUI che è stata creata ed è utilizzato anche per generare il codice.

```
// generated by Fast Light User Interface Designer (fluid) version 1.0302

#ifndef prova_h
#define prova_h
#include <FL/Fl.H>                                                  /*1*/
#include <FL/Fl_Double_Window.H>                                    /*2*/
#endif
```

Il file di intestazione `prova.h` contiene l'inclusione del file di intestazione principale (`1`) e del file dove è definita, in questo esempio, la classe cui appartiene l'unico oggetto utilizzato nel progetto della GUI (`2`).

```
// generated by Fast Light User Interface Designer (fluid) version 1.0302

#include "prova.h"

int main(int argc, char **argv) {
  Fl_Double_Window* w;                                              /*1*/
  { Fl_Double_Window* o = new Fl_Double_Window(150, 125, "Prova");  /*2*/
    w = o;
    o->end();                                                       /*3*/
  } // Fl_Double_Window* o
  w->show(argc, argv);                                              /*4*/
  return Fl::run();                                                 /*5*/
}
```

Nella `1` del file `prova.cxx` viene dichiarato il **puntatore ad un oggetto** della classe `Fl_Double_Window` (il nome di qualsiasi classe definita in FLTK comincia con il prefisso `Fl_`).

Dalla `2` vengono specificate le *caratteristiche* dell'oggetto che fa parte della GUI. Qui vengono inseriti anche gli altri elementi della GUI, assenti in questo caso, che fanno parte della finestra. Il richiamo del metodo della `3` chiude la definizione della finestra e del suo contenuto.

Le ultime due righe sono fondamentali: il metodo `show` richiamato nella `4` permette la visualizzazione della GUI (la window con tutto il suo contenuto). La `5` richiama il loop di attesa (`run`): il programma si pone in attesa di eventi da parte dell'utente cui risponderà con gli opportuni comportamenti definiti nelle funzioni callback associate ai vari oggetti. Dal loop si esce se la finestra della GUI viene chiusa.

Prima di passare all'esame di programmi più complessi è opportuna qualche precisazione:

- Quando si vuole definire un nuovo elemento dentro la finestra principale, questa deve essere selezionata: l'ordine degli elementi rispecchia l'ordine del codice generato. Gli elementi possono essere spostati, dopo la selezione e nella *Widget Browser*, utilizzando i tasti *F2* ed *F3*. Un elemento esistente si può cancellare selezionandolo e premendo la combinazione di tasti *Ctrl-x*.
- Per avere disponibile e in un unico posto tutto quello che può servire per sviluppare propri programmi, indipendentemente dagli esempi proposti, si possono trovare in appendice alla fine di questi appunti, le classi e i metodi utilizzati dagli oggetti dichiarati negli esempi.

Progetto 1: raddoppio di un numero

Come primo esempio di applicazione si propone la costruzione di un programma in cui se si inserisce un valore numerico e si clicca su un pulsante viene visualizzato il valore raddoppiato.

1. Avviare FLUID e selezionare, come visto in precedenza, *Function* (primo pulsante a sinistra in alto nella *Widget Bin* o scegliere dal menu *New, Code, Function/Method*. Cancellare il nome proposto della funzione in modo che sia impostato a `main`.
2. Scegliere *Window* dalla *Widget Bin* (quarto pulsante in alto da sinistra o, dal menù *New, Group, Window*). Fare visualizzare, se non lo è già, la finestra delle proprietà della *Window* cliccando due volte (tasto sinistro del mouse) sulla finestra selezionata. Nel tab *GUI* inserire nel campo *Label* la scritta che verrà visualizzata nella barra del titolo: nell'esempio `Raddoppia Numero`.
3. Inserire *Value Input* (*New, Validators, Value_Input* o il pulsante relativo nella riga in basso della zona centrale della *Widget Bin*). Dalla finestra delle proprietà, nel tab *GUI*, si può impostare la *Label* `Valore` (verrà visualizzata a sinistra della casella di input). Si può decidere di non impostare una *Label* e inserire un controllo *Box* (*New, Other, Box* o penultimo pulsante in alto a destra della *Widget Bin*) in modo da sistemare una scritta nel posto che si vuole e non necessariamente accanto alla casella. La cosa più importante è

impostare *Name* nel tab *C++*. Per esempio si potrebbe scrivere `inp`. Questo sarà il nome associato al controllo e che permetterà di manipolare il valore in esso inserito.

4. Inserire *Value Output* specificando anche in questo caso come label `Doppio` e come nome, nella tab *C++*, `out`.
5. Ora bisogna inserire un controllo di tipo *Button* (primo pulsante, terzo gruppo da sinistra della *Widget Bin* o *New*, *Buttons*, *Button*). Nella finestra delle proprietà nel tab *GUI* si inserirà come *Label* la stringa `Raddoppia`. La cosa più importante è impostare nel tab *C++* la funzione di callback: nella casella *Callback* del tab inserire il nome della funzione, per esempio, `cbRaddoppia`. Per questioni di leggibilità in questi appunti, per le callback, viene utilizzato il nome del pulsante cui la funzione è associata preceduto da `cb`.

A questo punto la GUI è terminata. Per completare il progetto è necessario solo aggiungere il codice della funzione di callback.

Uno sguardo al file di intestazione prodotto da FLUID può essere di aiuto per comprendere la logica della generazione del codice da parte di FLUID stesso e le istruzioni che dovranno essere inserite nella callback del pulsante. Si può esaminare il codice direttamente da FLUID selezionando *Edit, Show Source Code*.

```
// generated by Fast Light User Interface Designer (fluid) version 1.0302

#ifndef source_view_tmp_h
#define source_view_tmp_h
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Value_Input.H>
extern Fl_Value_Input inp;                                      /*1*/
#include <FL/Fl_Value_Output.H>
extern Fl_Value_Output out;                                     /*1*/
#include <FL/Fl_Button.H>
void cbRaddoppia(Fl_Button* ,void*);                            /*2*/
#endif
```

Intanto si può notare che le dichiarazioni dei controlli inseriti (`1`) hanno visibilità globale, come d'altra parte specificato di default, e non modificato, nel menù di scelta a destra del *Name* del tab *C++* delle proprietà dei widgets. I controlli sono quindi accessibili da tutte le parti di codice che si ha necessità di aggiungere.

```
...
Fl_Value_Input *inp=(Fl_Value_Input *)0;

Fl_Value_Output *out=(Fl_Value_Output *)0;
...
```

Nel file del sorgente si nota, come osservato anche in precedenza, che `inp` e `out` sono puntatori ad oggetti delle rispettive classi.

La `2` dichiara il prototipo della funzione callback associata al pulsante. Tutte le callback seguono lo schema: `void` *`nomeCallback`*`(`*`ClasseWidget`*`*, `void*`)`. Per completare il programma è necessario definire tale funzione. Poiché in questo caso il codice è composto da pochissime righe, si può agevolmente inserire direttamente utilizzando gli strumenti di FLUID:

1. Selezionare il widget *Function* e, stavolta, invece di lasciare il campo *Name* vuoto per generare la funzione `main`, è necessario inserire in accordo con il prototipo (`2`) il nome della funzione di cui si vuole scrivere il codice:

   ```
   cbRaddoppia(Fl_Button* o,void*)
   ```

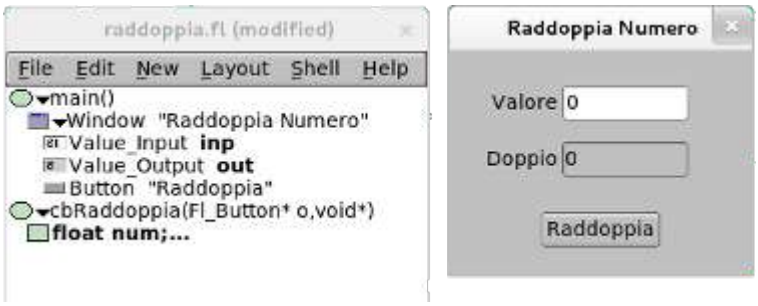
   inserire, sempre in accordo con il prototipo, in *Return Type*: `void` e confermare.

2. Selezionare la funzione appena creata nella *Widget Browser* e selezionare, in *Widget Bin*, il widget *Code* (primo da sinistra nella fila di mezzo della *Widget Bin* oppure inserire da *New, Code, Code*) e inserire, nella finestra *Code Properties*, le righe:

   ```
   float num;

   num = inp->value();                                          /*1*/
   num *= 2;                                                    /*2*/

   out->value(num);                                             /*3*/
   ```

Il codice semplicemente preleva il valore dalla casella di input (`1`), lo raddoppia (`2`) e inserisce il risultato nella casella di ouput (`3`) (si vedano le tabelle con i metodi delle classi interessate e il loro funzionamento nell'Appendice). Si noti l'uso dell'operatore `->` necessario per richiamare i metodi, essendo `inp` e `out` puntatori.

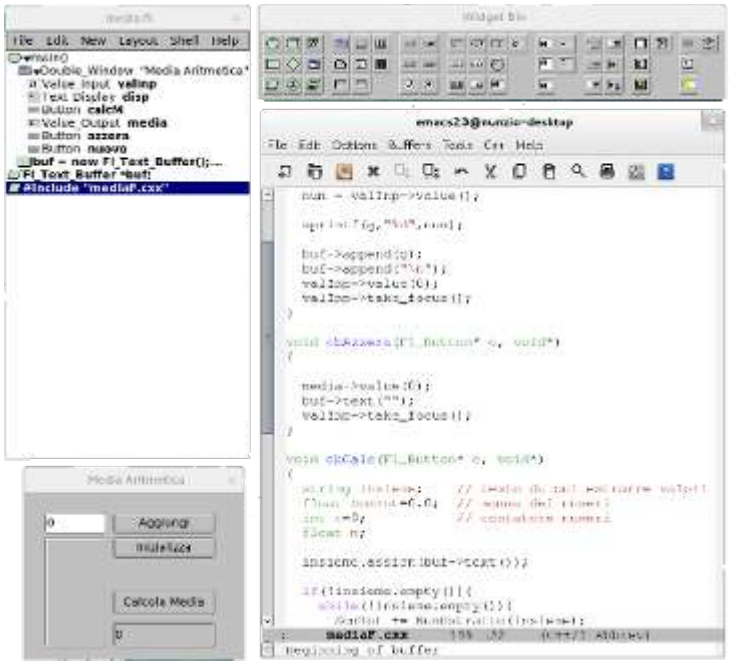
Ora il progetto dovrebbe apparire come nella figura sottostante:



Resta solo da salvare il file con la GUI, fare generare i file sorgenti (prima *File, Save,* se il nome è già stato scelto in precedenza, o *Save As* e dopo *File, Write Code*) e compilare, nella directory dove sono stati salvati i file:

```
fltk-config --compile raddoppia.cxx
```

Per progetti più complessi, dove necessita la stesura di una maggiore quantità di righe di codice, è opportuno e comodo, utilizzare un approccio diverso: distinguere la parte del disegno della GUI dalla parte di gestione e, utilizzare per questa ultima, un editor specializzato (per esempio Emacs) che consenta scrittura facilitata del codice e maggiore comodità di gestione.



Progetto 2: media di una serie di numeri

Si propone adesso la costruzione di un programma che permette l'input di una serie di numeri e ne calcola la media aritmetica.

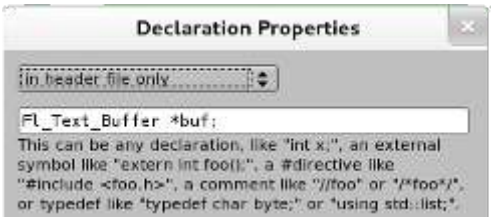
Quando si inserisce un nuovo valore si può premere *Invio* o il pulsante *Aggiungi* e il numero si accoda alla lista visualizzata nello spazio sottostante la casella di input. Premendo il pulsante *Calcola Media* viene calcolata la media aritmetica dei valori inseriti fino a quel momento. Il pulsante *Inizializza* permette di azzerare la lista dei numeri su cui fare la media e si può ripartire con una nuova lista.

Per prima cosa si procede alla costruzione della GUI:

1. Avviato FLUID si comincia selezionando *Function*. Il *Name* sarà lasciato vuoto per costruire la `main`.
2. Si aggiunge alla funzione una *Window* inserendo nella *Label* della finestra delle proprietà `Media Aritmetica`.
3. Si aggiunge un elemento di tipo *Value Input* come quello utilizzato nel progetto precedente. Il *Name* del tab *C++* delle proprietà si può impostare a `valInp`.
4. Un elemento di tipo *Text Display* (nella fila centrale del quinto gruppo da sinistra della

Widget Bin o da *New, Text, Text Display*) verrà utilizzato per contenere i numeri introdotti. Nel *Name* del tab *C++* della finestra delle proprietà, inserire `disp`.

5. Aggiungere un controllo di tipo *Return Button* (selezione del pulsante a destra di *Button* nella *Widget Bin* o da *New, Buttons, Return_Button*). Nel tab *GUI* delle proprietà inserire `Aggiungi` e nel tab *C++* inserire: nel *Name* `aggiungi` e in *Callback* `cbAggiungi`. La differenza fra un pulsante di tipo *Return Button* e uno di tipo *Button* consiste nel fatto che il primo è attivabile anche direttamente premendo il tasto *Invio*: il che è comodo se si sta usando la tastiera. Può essere la scelta migliore per l'opzione di default quando si effettuano inserimenti.
6. Aggiungere due elementi di tipo *Button*. Le *Label* del tab *GUI* delle proprietà dei pulsanti si imposteranno a: `Inizializza` e `Calcola Media` e i *Name* del tab *C++*, rispettivamente, a `inizializza` e `calcM`. Le *Callback* associate ai pulsanti saranno: `cbInizializza`, `cbCalcM`.
7. Aggiungere un elemento di tipo *Value Output* dove verrà visualizzato il valore medio con la proprietà *Name* nel tab *C++* impostata a `media`.
8. Per completare la GUI è necessario aggiungere un nuovo elemento non visuale (non è disponibile nella *Widget Bin* e non si vede nella GUI) necessario a contenere i dati che verranno visualizzati nel *Text Display* che è solo un contenitore. Si tratta di un elemento di tipo *Text Buffer* e affinché si comporti come tutti gli altri widget è necessario fare una dichiarazione globale nel file di intestazione. Selezionare *Declaration* (primo pulsante a sinistra in basso nella *Widget bin* oppure selezionare da *New, Code, Declaration*)



   La dichiarazione va inserita nel file di intestazioni come si selezionerà nel menù di scelta che precede, nella finestra, la casella per l'inserimento della riga della dichiarazione stessa. La dichiarazione da inserire sarà:

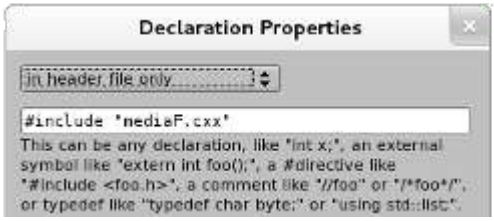
```
Fl_Text_Buffer *buf;
```

9. È necessario ora inserire il codice per associare il buffer al campo *Text Display* affinché quest'ultimo ne possa visualizzare il contenuto al suo interno. Selezionare *Code* e inserire le linee:

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

La GUI è completata. Resta da aggiungere l'`include` del file contenente il codice delle funzioni, comprese le callback, che saranno contenute in un file a parte. A tale scopo come parte finale scegliere ancora una volta *Declaration* con la stessa scelta nel menù a discesa, come il caso precedente, per l'inserimento nel solo file di intestazione:

```
#include “mediaF.cxx”
```



È importante inserire questa dichiarazione per ultima perché in questo modo le dichiarazioni degli elementi grafici, nel file `media.h`, si troveranno in righe precedenti e, dalle funzioni del file `mediaF.cxx`, si potrà accedere a tutti gli elementi della GUI.

La comodità dell'utilizzo dell'include consiste nel fatto che, alla fine, l'unico file che dovrà essere compilato sarà `media.cxx` generato da FLUID, che conterrà l'inclusione di `mediaF.cxx` assieme a `media.h`. Anche in questo caso, per ragioni di leggibilità, il nome del file che contiene il codice delle funzioni è scelto in modo da essere lo stesso del progetto ma con, aggiunta, la lettera `F` finale.

A lavoro concluso la GUI dovrebbe risultare definita in questo modo:

Per la scrittura del codice inserito in `mediaF.cxx` si può utilizzare Emacs e, alla fine, si dovrà salvare il file nella stessa directory che contiene `media.fl`, `media.cxx` e `media.h`.

Esame delle funzioni da definire nel file `mediaF.cxx`:

```
// callback associata al pulsante Aggiungi
// aggiunge un numero alla lista di cui calcolare la media

void cbAggiungi(Fl_Return_Button* o, void*)
{
  int num;
  char g[10];

  num = valInp->value();                                    /*1*/

  sprintf(g,"%d",num);                                      /*2*/

  buf->append(g);                                           /*3*/
  buf->append("\n");                                        /*3*/
  valInp->value(0);                                         /*4*/
  valInp->take_focus();                                     /*5*/
}
```

Si acquisisce (`1`) il valore contenuto nella casella di input, lo si trasforma (`2`) in formato stringa C type e lo si inserisce (`3`) assieme ad un carattere di *new line* nel buffer.

La `4` reinizializza la casella per l'input e la `5` porta il focus (cursore di inserimento) nel controllo.

```
// callback associata al pulsante Inizializza

void cbInizializza(Fl_Button* o, void*)
{
  media->value(0);                                                    /*1*/
  buf->text("");                                                      /*2*/
  valInp->take_focus();                                               /*3*/
}
```

La callback associata al pulsante di inizializzazione è abbastanza semplice: si azzera (1) il valore della media, il contenuto del buffer (2) e si restituisce il focus (3) alla casella di input.

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

// prototipo di funzione richiamata nella callback

float NumEstratto(string &);

// callback associata al pulsante Calcola Media

void cbCalcM(Fl_Button* o, void*)
{
  string insieme;    // testo da cui estrarre valori
  float SomTot=0.0;  // somma dei numeri
  int c=0;           // contatore numeri
  float m;

  insieme.assign(buf->text());                                        /*2*/

  if(!insieme.empty()){
    while(!insieme.empty()){
      SomTot += NumEstratto(insieme);                                 /*3*/
      c++;
    }

    m = SomTot/c;
    media->value(m);                                                  /*4*/
  }

  valInp->take_focus();                                               /*5*/
}

// estrae un numero da una stringa

float NumEstratto(string &s)
{
  float numero;
  string sNum;
  int pos;

  pos = s.find('\n');                                                 /*6*/
  sNum = s.substr(0,pos);                                             /*7*/

  numero = atof(sNum.c_str());                                        /*8*/
  s = s.erase(0,pos+1);                                               /*9*/

  return numero;
}
```

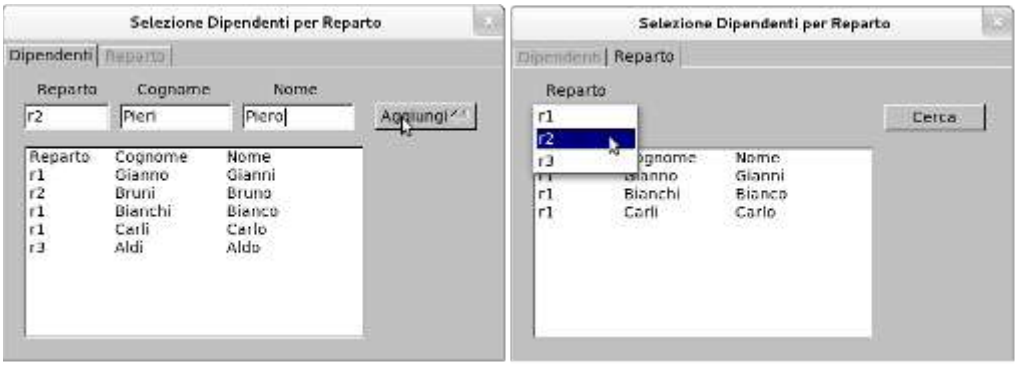
Poiché si dovranno svolgere elaborazioni sulle stringhe da trasformare in numeri ecc.. conviene (1) includere la classe `string` e trasformare (2) in stringa C++ type il contenuto del buffer.

Finché nella stringa ci sono caratteri, si estrae (3) un numero e si aggiunge al totale. Dopo aver effettuato il calcolo della media il risultato (4) viene inviato alla casella di output e si restituisce il focus alla casella di input (5).

Per l'estrazione di un numero dalla stringa contenuta nel buffer, prima si cerca il carattere di new line (6), poi si estraggono i caratteri che lo precedono (7) e si trasformano (8) in numero. Alla fine si eliminano tutti i caratteri fino al *new line* (9). In questo modo la stringa sarà vuota quando tutti i numeri saranno stati estratti.

Progetto 3: dipendenti di un reparto

Il programma ora proposto permette di inserire i dati dei dipendenti di una ditta e di selezionare i dipendenti che lavorano in un reparto scelto fra quelli in cui sono registrati dipendenti.



la GUI è composta da due schede: nella prima si può inserire un dipendente. Se non si specifica il reparto, essendo questo un dato importante per la selezione, viene mostrata una finestra di dialogo con un messaggio di avviso. Si può passare da un campo all'altro premendo il pulsante *Tab* e dopo l'inserimento dell'ultimo campo (*Nome*) si può premere *Invio* (o premere il tasto *Aggiungi*) per accodare il dipendente alla lista. Nella seconda scheda è presente un menù a discesa dal quale si può selezionare uno dei reparti e, premendo il pulsante *Cerca,* si possono vedere tutti i dipendenti inseriti che lavorano nel reparto selezionato.

1. Avviato FLUID si comincia selezionando *Function*. Il *Name* sarà, al solito, lasciato vuoto per costruire la `main`.
2. Si aggiunge alla funzione una *Window* inserendo nella *Label* della finestra delle proprietà `Selezione Dipendenti per Reparto`.
3. Inserire un elemento *Tabs* (secondo gruppo da sinistra, icona riga centrale a sinistra, nella *Widget Bin* o *New, Group, Tabs*). Non è necessario specificare alcuna proprietà.
4. Inserire *Group* (accanto alla *Window* nella *Widget Bin*) e immettere nella finestra delle proprietà nel tab *GUI* la *Label* `Dipendenti` (sarà visualizzata nell'etichetta della scheda). L'elemento raggrupperà tutti i controlli che verranno visualizzati nella scheda. Nel tab *Style* della finestra delle proprietà, si può selezionare (scegliendo da *Label Color*) un colore più chiaro in modo che, quando la scheda non risulterà selezionata, il colore dell'etichetta sia più chiaro e sarà ancora più evidente la scheda attiva.

5. Selezionare il gruppo e inserire dentro tre elementi di tipo *Input* (primo pulsante in alto, quinto gruppo della *Widget Bin*). Nelle proprietà *Name* del tab *C++* inserire, rispettivamente `rep`, `cognome`, `nome`.
6. Inserire, sempre all'interno del gruppo (basta accertarsi che sia selezionato il gruppo) tre elementi *Box* in modo da sistemare le etichette sopra le caselle di input. Le *Label* del tab *GUI* delle proprietà dei box saranno, rispettivamente, `Reparto`, `Cognome`, `Nome`.
7. Sempre all'interno del gruppo, inserire un elemento di tipo *Browser* (pulsante in alto penultimo gruppo da sinistra nella *Widget Bin*). Nel *Name* del tab *C++* della finestra delle proprietà, inserire `elDip`.
8. Per completare il gruppo inserire un *Return Button*. Specificare nella *Label* del tab *GUI* della finestra delle proprietà, `Aggiungi`. Nel tab *C++* bisogna inserire nel *Name* `aggiungi` e in *Callback,* `cbAggiungi`.
9. Selezionare *Tabs* e inserire un nuovo *Group*. Nelle proprietà inserire `Reparto` nella *Label*. Anche qui si può scegliere, nel tab *Style* della finestra delle proprietà, lo stesso colore più chiaro scelto nel gruppo precedente.
10. Selezionare il nuovo gruppo e inserire un *Box* con *Label* `Reparto`.
11. All'interno del gruppo inserire un elemento *Choice* per il menù a discesa (sesto gruppo da sinistra della *Widget Bin*, icona a sinistra in basso). Il *Name* sarà impostato a `listaRep`.
12. Inserire nel gruppo un elemento *Browser*. La proprietà *Name* potrà essere `elDipCerca`.
13. Completare il gruppo con un *Button* con *Label* `Cerca`, *Name* `cerca` e *Callback* `cbCerca`.
14. Per completare la GUI è necessario inserire un *Code* per specificare le caratteristiche dei Browser utilizzati. Selezionare *Code* e inserire le linee:

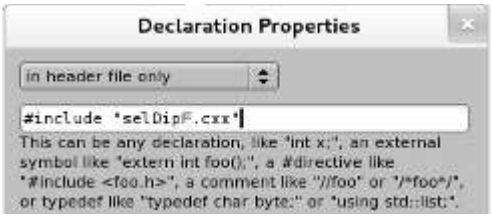
```
int colonne[]={80,105,105};                                    /*1*/

elDip->column_widths(colonne);                                 /*2*/
elDip->column_char('\t');                                      /*3*/
elDip->type(FL_MULTI_BROWSER);                                 /*4*/
elDip->add("Reparto\tCognome\tNome");                          /*5*/

elDipCerca->column_widths(colonne);                            /*2*/
elDipCerca->column_char('\t');                                 /*3*/
elDipCerca->type(FL_MULTI_BROWSER);                            /*4*/
```

Nella `1` si dichiara un array di tipo `int` contenente la larghezza delle colonne. Negli elementi dell'array sono inseriti numeri ricopiati dalle larghezze delle tre caselle di input dei dati del dipendente (*Width* della riga *Position* del tab *GUI* della finestra delle proprietà).

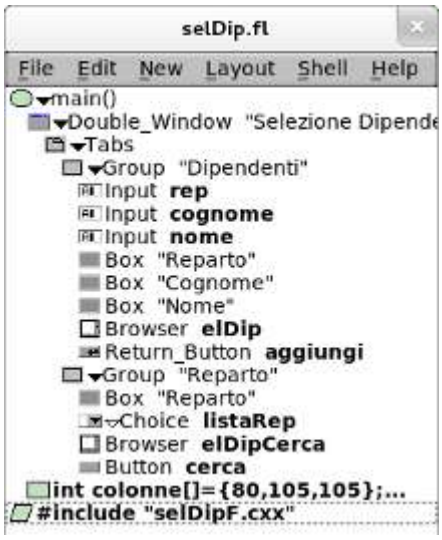
Nelle `2` si passa al browser la larghezza delle colonne e nelle `3` il carattere che separerà una colonna dall'altra: il carattere di tabulazione. Nelle `4` si specifica che si tratta di browser multicolonna. La `5` aggiunge la riga delle intestazioni delle colonne nel browser presente nella prima scheda dell'applicazione.

La GUI è completata. Resta da inserire, alla fine, l'include del file contenente il codice delle funzioni. Inserire una *Declaration:*

```
#include "selDipF.cxx"
```



A conclusione di tutti i passaggi specificati la GUI dovrebbe essere:



Esame delle funzioni da definire in `selDipF.cxx`:

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

// Prototipi

string estraiRep(string);
void modListaRep(const char *);

// Callback associata al pulsante Aggiungi
// Aggiunge un dipendente alla lista

void cbAggiungi(Fl_Return_Button* o, void*)
{
  string r,cog, nom;
  string linea;
  char lin[100];

  // verifica se inserito reparto

  if(!strcmp(rep->value(),"")){                                 /*2*/
    fl_alert("E\' necessario inserire il reparto");             /*3*/
    rep->take_focus();
    return;
  }

  // verifica se aggiungere reparto alla lista di scelta

  modListaRep(rep->value());                                    /*4*/
```

```
  // aggiunge dati del dipendente all'elenco

  r.assign(rep->value());                                          /*5*/
  cog.assign(cognome->value());                                    /*5*/
  nom.assign(nome->value());                                       /*5*/

  linea = r + "\t" + cog + "\t" + nom;                             /*6*/
  strcpy(lin,linea.c_str());                                       /*7*/

  elDip->add(lin);                                                 /*8*/
  rep->value("");                                                  /*9*/
  cognome->value("");                                              /*9*/
  nome->value("");                                                 /*9*/

  rep->take_focus();                                               /*10*/
}
```

L'inclusione della `1` definisce un insieme di funzioni della FLTK per l'utilizzo delle comuni finestre di dialogo degli ambienti grafici.

Poiché il codice del reparto è importante per l'elaborazione, visto che seleziona i dipendenti per reparto, è necessario che l'utente inserisca tale codice. La funzione controlla (`2`) se è stata lasciata vuota la casella di input del codice e, in tal caso, visualizza (`3`) una finestra di dialogo con un messaggio di avviso. Si tratta delle classiche finestre degli ambienti grafici, che non permettono di andare avanti se prima non vengono chiuse, che mostrano un avviso accompagnato da un icona con un simbolo di attenzione e che presentano un pulsante *Ok* che permette di chiuderle e riprendere il controllo del programma. L'inclusione della `1` serve per disporre, fra l'altro, di `fl_alert()`.

Se non è stato inserito alcun valore nel campo del reparto la funzione termina altrimenti si passa il reparto (`4`) ad una funzione che provvede ad aggiungerlo, se non già esistente, alla lista dei reparti da cui scegliere per la selezione.

I valori inseriti nelle caselle di input vengono convertiti (`5`) in stringhe C++ type in modo da creare facilmente la riga da inserire nel browser (`6`). Nella riga fra un campo e l'altro è stato inserito un carattere di tabulazione in modo da incolonnare i dati, così come previsto dalle impostazioni effettuate sui componenti *Browser* della GUI. Infine la riga è riconvertita in formato C type (`7`). Era possibile generare la stringa da inserire anche senza effettuare la doppia conversione utilizzando più volte la funzione `strcat()`, ma qui si è fatta questa scelta per motivi di maggiore leggibilità in riferimento soprattutto alla `6`.

la riga viene aggiunta al browser (`8`), si reinizializzano i valori delle caselle di input (`9`) e si porta il focus sulla casella dell'inserimento del reparto (`10`).

```
// Verifica se aggiornare lista reparti

void modListaRep(const char *r)
{
  int i;
  bool trovato=false;

  if(!listaRep->size())                                            /*1*/
    listaRep->add(r);                                              /*2*/
  else{
    for(i=0;i<listaRep->size()-1;i++){                             /*3*/
      if(!(strcmp(r,listaRep->text(i))))                           /*4*/
        trovato = true;
```

```
    }
    if(!trovato)
      listaRep->add(r);                                          /*5*/
  }
}
```

Se la lista dei reparti è vuota (1) si inserisce il reparto (2) nelle voci del menù a discesa. In questo caso il metodo `size()` restituisce un valore nullo.

Il ciclo 3 effettua la scansione delle righe contenute nel menù a discesa. Il metodo `size()` restituisce la quantità di righe +1 (la riga con il terminatore), ma il metodo `text()` richiede come parametro il numero di riga (4) e le righe sono contate a cominciare dal valore `0`. Quindi se, per esempio, ci sono 5 righe, `size()` restituisce 6 e il ciclo deve esaminare le righe dalla 0 alla 4.

Se il reparto passato come parametro alla funzione non coincide con alcuna delle righe esistenti nella *Choise* (4) si aggiunge in coda alle righe esistenti già (5).

```
// Callback associata al pulsante Cerca
// Seleziona i dipendenti del reparto cercato

void cbCerca(Fl_Button* o, void*)
{
  string rInp, rDip, linOld;
  int i;

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

  rInp.assign(listaRep->text());                                 /*2*/

  // inizializza browser

  elDipCerca->clear();                                           /*3*/
  elDipCerca->add("Reparto\tCognome\tNome");                     /*4*/

  // le righe cominciano da 1 ma
  // nella 1 ci sono le intestazioni

  for(i=2;i<=elDip->size();i++){                                 /*5*/

    // estrae e verifica reparto

    linOld.assign(elDip->text(i));
    rDip = estraiRep(linOld);                                    /*6*/

    if(rDip==rInp)
      elDipCerca->add(elDip->text(i));                           /*7*/
  }
}
```

Se non è stato selezionato alcun reparto (1), o perché l'utente non ha effettuato una scelta o perché non è stato possibile fare una scelta poiché non sono stati inseriti dipendenti, viene visualizzata una finestra di dialogo di avvertimento e si abbandona la funzione. La non selezione infatti fa ritornare al metodo il valore `-1`. Se invece una qualche selezione è stata effettuata, il metodo ritorna il numero di selezione effettuata contata dal valore `0` (la prima selezione).

Se c'è una selezione si trasforma in stringa C++ type il reparto selezionato (2), si pulisce l'area che conterrà i dati (3) e si scrive la riga delle intestazioni (4).

Si esaminano tutte le righe che rappresentano tutti i dipendenti inseriti (5). Si estrae dalla riga il codice del reparto (6) e se coincide con quello scelto su cui si vuole effettuare la selezione si aggiunge la riga del dipendente al contenitore della selezione (7).

L'inizializzazione del ciclo `5` al valore `2` dipende dal fatto che la prima riga del browser `elDip` contiene l'intestazione che non è utile nell'elaborazione e che deve essere scartata.

```
// Estrae reparto dalla linea

string estraiRep(string ri)
{
  string ro;
  int pos;

  pos = ri.find("\t");                                          /*1*/
  ro = ri.substr(0,pos);                                        /*2*/

  return ro;
}
```

L'ultima funzione da commentare è abbastanza intuitiva: si cerca la posizione del carattere (il tabulatore) che separa la prima colonna, contenete il codice del reparto, dalla successiva (1) e si estrae la sottostringa (2) del codice.

Appendice: classi e metodi usati negli esempi

`Fl_Browser`

Componente che mostra un riquadro, con eventuali barre di scorrimento, per scorrere linee di testo. Le linee possono essere multicolonna in modo da consentire la visualizzazione di tabelle. Ogni colonna è separata dalla seguente da un carattere specifico. Una linea è separata dalla successiva dal *new line*.

Metodo	**Comportamento**
`add()`	Aggiunge una linea in coda a quelle esistenti nel browser. Il parametro può essere una stringa racchiusa tra doppi apici o una variabile di tipo `char[]`.
`clear()`	Inizializza a stringa vuota il testo contenuto nel browser.
`column_char()`	Specifica il carattere separatore fra le colonne di un browser che mostra i dati distribuiti in più colonne.
`column_widths()`	Specifica la larghezza delle colonne in cui verranno distribuiti i dati. Richiede come parametro un array C type di interi contenete la larghezza in pixel delle varie colonne.
`size()`	Ritorna il numero di righe presente nel browser. Se il browser è vuoto restituisce il valore `0`.
`text()`	Ritorna una stringa del tipo C type contenente la riga il cui numero è specificato come parametro del metodo. Il numero specificabile comincia dal valore `1` (la prima riga).
`type()`	Specifica il tipo di Browser. Negli esempi è sempre `FL_MULTI_BROWSER` per dati incolonnati.

`Fl_Choise`

Componente che permette la scelta di una delle voci contenute in un menù a discesa.

Metodo	**Comportamento**
`add()`	Aggiunge una voce in coda a quelle esistenti nel menù a discesa. Il parametro può essere una stringa racchiusa tra doppi apici o una variabile di tipo `char[]`.
`size()`	Restituisce un numero intero che rappresenta la quantità di elementi presenti nel menù a discesa. È opportuno tenere presente che la lista degli elementi è terminata dal valore `NULL` che viene conteggiato e quindi, per esempio, se nel menù sono presenti `5` voci il metodo restituirebbe `6`. Se la lista delle opzioni da cui scegliere è vuota, il metodo restituisce il valore `0`.

Metodo	Comportamento
`text()`	Senza parametro ritorna la stringa, in formato C type, selezionata. Può essere inserito un numero intero come parametro e, in questo caso, il metodo restituisce la stringa della riga. Le righe vengono contate da 0.
`value()`	Ritorna l'intero associato al numero di riga dell'opzione scelta contato dal valore `0` (selezione della prima voce del menù di scelta). Se non si è effettuata alcuna scelta, il metodo ritorna il valore `-1`.

`Fl_Text_Buffer`

Componente non visuale delle librerie. Contiene il testo multilinea da associare ad un oggetto della classe `Fl_Text_Display`.

Metodo	Comportamento
`append()`	Aggiunge un testo al contenuto del buffer. Come parametro si può specificare una stringa racchiusa da doppi apici o una variabile di tipo `char []`.
`text()`	Inserisce un testo nel buffer. Ciò che esisteva prima nel buffer viene sostituito dalla nuova stringa specificata come parametro. Il parametro può essere specificato anche come variabile di tipo `char[]`. Se non specificato alcun parametro restituisce in una stringa C type il contenuto del buffer.

`Fl_Text_Display`

Spazio disponibile per l'inserimento di testo multilinea. Lo spazio non è immediatamente disponibile: è necessario definire un oggetto della classe `Fl_Text_Buffer` e associarlo ad esso.

Metodo	Comportamento
`buffer()`	Definisce l'associazione con il buffer. Come parametro è necessario inserire un oggetto della classe `Fl_Text_Buffer`.

`Fl_Value_Input` e `Fl_Input`

`Fl_Value_Input` è una casella per l'inserimento di valori numerici. Si possono inserire sia interi che numeri con parte decimale. Se nel tab *GUI* delle proprietà dell'elemento si inserisce, nella casella *Step* della riga *Value*, il valore `1`, la casella permette l'inserimento di soli numeri interi.

`Fl_Input` è una casella per l'inserimento di testo.

Metodo	Comportamento
`take_focus()`	Porta il focus nel controllo. Il cursore di inserimento è pronto per l'inserimento di un valore.

Metodo	Comportamento
`value()`	Utilizzato senza specificare alcun parametro ritorna il valore contenuto nel controllo come `int` o `float` a seconda del tipo di variabile di destinazione. Se si tratta di oggetto della classe `Fl_Input` ritorna una stringa C type ovvero un `char[]`. Se si inserisce come parametro un valore, numerico o stringa a seconda il tipo di casella, il valore inizializza la casella.

`Fl_Value_Output` e **`Fl_Output`**

Casella per l'output di valori numerici un oggetto della prima classe, output stringhe per oggetti della seconda.

Metodo	Comportamento
`value()`	Se è un oggetto della classe `Fl_Value_Output` nel parametro bisogna inserire il valore numerico che verrà visualizzato nel controllo. Può essere di qualsiasi tipo numerico. Se si tratta di oggetti della classe `Fl_Output` il parametro è una stringa delimitata da doppi apici o una variabile di tipo `char[]`.

Riferimenti

- **http://www.fltk.org/** il sito di riferimento del toolkit FLTK da cui si possono scaricare le diverse versioni per varie piattaforme oltre alla documentazione in formato HTML e PDF
- **http://www.fltk.org/doc-1.3/** il manuale di programmazione che contiene anche il reference completo di tutte le classi definite nel toolkit
- **http://seriss.com/people/erco/fltk/** esempi *chiavi in mano* di applicazione di vari oggetti delle classi di FLTK
- **http://www.dmoz.org/Computers/Software/Operating_Systems/Graphic_Subsystems/Toolkits/FLTK/** altra pagina contenente diversi link ad applicazioni che utilizzano FLTK, tutorial.