Fondamenti di Informatica A
Laboratorio 11
Applicazione completa

Danilo Pianini
danilo.pianini@unibo.it
Mirko Viroli
mirko.viroli@unibo.it

Alma Mater Studiorum—Università di Bologna

29 maggio 2013

Nota prima di cominciare

Vi ricordo che gli esercizi svolti in aula ieri dal prof. Viroli sono disponibili online. Si tratta di un pacchetto molto utile per chi sta preparando l'esame, ed il suo download è caldamente consigliato. Lo trovate all'indirizzo
`http://bit.ly/finfa2013-lastexercises`
Non sono esercizi da svolgere in questo laboratorio, ma ho voluto comunque ricordarveli.

Operazioni preliminari

- **AVVISO:** per svolgere questo laboratorio è necessiario utilizzare la macchina virtuale del corso. Si faccia boot sulla **macchina fisica**, dove è disponibile una macchina virtuale Linux installata in `D:\finfa`
- Slides: `http://bit.ly/finfa2013-e11`
- Codice: `http://bit.ly/finfa2013-e11-code`
- Svolgete i punti contrassegnati come **OPZ.** solo quando avete completato tutti gli altri punti: sono versioni leggermente più difficili degli esercizi precedenti.

Applicazione completa

In questo laboratorio metterete mano dentro un'applicazione grafica C scritta da me.

- Tale applicazione è in grado di mostrare a video delle funzioni, disegnandole all'interno di un'area di disegno.
- Inoltre, a ciascuna funzione può applicare un filtro che la modifica, e disegnare a video la funzione filtrata.
- Il codice è piuttosto commentato: aprite i sorgenti e cominciate a prendere dimestichezza con il sorgente

Componenti:

- La libreria `colorfactory.h` che abbiamo visto a lezione è utilizzata anche qui
- La libreria `functions.h` descrive una struttura dati che associa un puntatore a funzione ad un nome
- La libreria `filters.h`, è molto simile, ma la struttura (parametri di ingresso e uscita) del puntatore a funzione è differente
- La libreria `points.h`, contiene una semplice struttura dati per la descrizione di un punto in 2D con coordinate cartesiane.

Compilazione

Per prima cosa, bisogna compilare il programma. Vi ricordo che per compilare librerie che si appoggiano a GTK+3 è necessario utilizzare gcc come segue:

```
gcc ‘pkg-config --libs --cflags gtk+-3.0‘ -c -Wall libreria.c
```

Il particolare apice da utilizzare può essere ottenuto in ambiente Linux con la combinazione di tasti `AltGr+'`
Per compilare invece un programma che usa GTK+3 ed altre librerie (in questo caso `lib1` e `lib2`), il comando è:

```
gcc ‘pkg-config --libs --cflags gtk+-3.0‘ lib1.o lib2.o -Wall programma.c
```

- Non è necessario compilare `points.h`, dato che non ha alcun file c di implementazione.
- Compilare `colorfactory.c` come libreria
- Compilare `functions.c` come libreria
- Compilare `filters.c` come libreria
- Compilare `FunctionDrawer.c` come programma che usa le tre librerie precedenti

Esecuzione

Una volta compilato, si esegua l'applicazione tramite doppio click sull'eseguibile oppure lanciando il comando

```
./a.out
```

Eventualmente sostituendo ad `a.out` il nome che avete dato all'eseguibile da produrre.
Provate a graficare diverse funzioni, applicare diversi filtri e a giocare con le sliders.

Cambiare colore al disegno della funzione filtrata

La funzione filtrata è attualmente colorata in nero, il che la rende poco evidente. Si modifichi il programma affinché sia colorata in rosso.

1. In `colorfactory.h`, si definisca un nuovo colore, ossia una variabile di tipo `const GdkRGBA*`, dandogli un nome appropriato (ad esempio, `color_red`)
2. In `colorfactory.c`, si definisca una nuova funzione `static GdkRGBA* gen_color_red(void)` che inizializzi la variabile dichiarata nell'header. Si prendano come riferimento le tre funzioni fornite. Si ricorda, inoltre, che per creare il colore rosso è necessario mettere ad `1.0` i campi di colore che descrivono rosso e trasparenza (alpha) e a `0.0` i restanti campi.
3. In `colorfactory.c`, si modifichi la funzione `void init_color_factory(void)` in maniera tale che inizializzi anche il nuovo colore. Si prenda spunto dal modo in cui vengono inizializzati gli altri colori.

Cambiare colore al disegno della funzione filtrata

4. Si ricompili colorfactory.c
5. In `FunctionDrawer.c`, nella funzione
   `static void draw_function(cairo_t *)`
   si modifichi l'argomento passato alla funzione `gdk_cairo_set_source_rgba` quando viene chiamata per definire il colore della funzione filtrata, in maniera tale che il colore non sia più `color_black` ma il nuovo colore che avete definito.
6. Si ricompili `FunctionDrawer.c` e si verifichi che la funzione filtrata appaia ora di colore rosso.

Implementazione di nuove funzioni

Si implementino nel programma le seguenti nuove funzioni:

- "`cos(x)`" — $\cos(x)$
- "`log(x)`" — $\log(x)$
- **OPZ.** "`wing(x)`" — $\begin{cases} \cos(x) + 5\log(-x+1) & x \leq 0 \\ \cos(x) + 5\log(x+1) & x > 0 \end{cases}$

Per implementare una qualunque nuova funzione `f`, si seguano i seguenti passi:

1. In `functions.c`, prendendo spunto dalle implementazioni esistenti, si implementi
   `static double f(double)`
2. In `functions.c`, si modifichi la funzione
   `void init_functions(void)`
   in maniera che la variabile `functions_number` contenga il corretto numero di funzioni definite, e che la nuova funzione `f` abbia una propria entry nell'array `functions`. Si osservi come vengono gestite le funzioni esistenti.

Implementazione di nuove funzioni

3. Si ricompili `functions.c`
4. Si ricompili `FunctionDrawer.c` e si verifichi che la funzione aggiunta appare ora nell'elenco di quelle disponibili. Si verifichi che venga graficato quanto atteso.

Implementazione di nuovi filtri

Si implementino nel programma i seguenti nuovi filtri:

- `absolute` — calcola la funzione modulo della funzione data ($\sqrt{(f(x))^2}$), ossia trasforma in positivi i valori negativi. Si prenda spunto dal filtro `onlypos` già implementato, che azzera i valori negativi di una funzione.
- **OPZ.** `derivate` — calcola la funzione derivata della funzione data $f'(x)$. Si ricordi la definizione di derivata come limite del rapporto incrementale: il valore della derivata è approssimabile come $f'(x) \approx \frac{f(x+\varepsilon)-f(x)}{\varepsilon}$. Si prenda spunto dal filtro `integrate` già implementato, che calcola (a meno di una costante) il valore della funzione integrale della funzione data ($\int f(x)dx$), approssimando l'integrale come somma dell'area di rettangoli.

Implementazione di nuovi filtri

Per implementare un qualunque nuovo filtro `f`, si seguano i seguenti passi:

1. In `filters.c`, prendendo spunto dalle implementazioni esistenti, si implementi
   `void f(GArray *, GArray *)`
2. In `filters.c`, si modifichi la funzione
   `void init_filters(void)`
   in maniera che la variabile `filters_number` contenga il corretto numero di filtri definiti, e che il nuovo filtro `f` abbia una propria entry nell'array `filters`. Si osservi come vengono gestiti i filtri esistenti.
3. Si ricompili `filters.c`
4. Si ricompili `FunctionDrawer.c` e si verifichi che il filtro aggiunto appare ora nell'elenco di quelli disponibili. Si verifichi che venga graficato quanto atteso.

Creazione di un pulsante

Si crei un pulsante nell'interfaccia la cui funzione sia quella di salvare su disco i punti calcolati dalla funzione filtrata. Per farlo, si seguano i seguenti passi:

1. In `FunctionDrawer.c`, all'interno della funzione `static void activate(GtkApplication *, gpointer)` si definisca un nuovo `GtkWidget *`. Lo si chiami, ad esempio, `save_button`. Tale definizione può avvenire assieme a quelle degli altri `GtkWidget *` già presenti.
2. In `FunctionDrawer.c`, all'interno della funzione `static void activate(GtkApplication *, gpointer)`, si istanzi il nuovo pulsante utilizzando la funzione `GtkWidget *gtk_button_new_with_label(char *)` Una corretta invocazione può essere ad esempio: `save_button = gtk_button_new_with_label("Save");`

Aggiunta del pulsante, collegamento della callback

3. Si aggiunga il pulsante al layout verticale di destra, in fondo. Per farlo, si invochi la funzione seguente:
   `gtk_box_pack_start(GTK_BOX(rlayout), save_button, TRUE, FALSE, 0);`
   Se si vuole posizionare il pulsante in basso, tale chiamata deve essere posta dopo le altre invocazioni della stessa funzione.
4. Si compili `FunctionDrawer.c` e si verifichi che il pulsante (ancora privo di funzionalità collegate ai propri signals) sia presente nell'interfaccia dove desiderato.
5. Si colleghi la funzione callback
   `void save(GtkApplication *, gpointer)`
   già implementata al signal "clicked" del pulsante. Per farlo, si utilizzi la seguente linea di codice:
   `g_signal_connect(save_button, "clicked", G_CALLBACK(save), NULL);`
6. Si compili `FunctionDrawer.c` e si verifichi che il pulsante quando premuto ora apre una finestra di salvataggio.

Completamento della callback, salvataggio dati

7. La callback
   `void save(GtkApplication *, gpointer)`
   sfortunatamente non è completa: manca la parte di salvataggio dei dati.
8. Si utilizzi `fprinf` per salvare tutti i dati contenuti nel `GArray` `filtered`. Per farlo, si scorra filtered e per ciascun punto in esso contenuto si stampi su file la coordinata x, uno spazio, la coordinata y, e un "a capo".
9. Come riferimento, si consideri che la funzione
   `static void load(void)`
   già implementata deve essere in grado di caricare correttamente i punti salvati salla `save`.
10. Si compili `FunctionDrawer.c` e si verifichi che il pulsante quando premuto ora apre una finestra di salvataggio, e che a fronte della selezione di un file esso viene riempito con i dati come atteso.
11. Si verifichi che il programma è in grado di caricare i punti salvati in precedenza.

Fine dei giochi

Al termine dei laboratori di questo corso dovreste essere in grado di:

- Scrivere programmi giocattolo in Java
- Scrivere semplici programmi in C
- Scrivere nuove librerie C
- Utilizzare librerie C scritte da altri
- Realizzare programmi C con grafica e accesso al file system (conseguenza della precedente)
- Last but not least, **passare l'esame**!
- Chi ha trovato il corso interessante e volesse imparare a scrivere software "sul serio", ha le basi necessarie per affrontare il corso di Programmazione a Oggetti. Dovrebbe essere fra i vostri opzionali.