Tutorial FLTK
CIT 1010 Programación Avanzada

Jorge Brito

4 de octubre de 2016

Resumen

Este documento contiene los primeros pasos para utilizar FLTK (pronunciada "fulltick"), una herramienta de interfaz gráfica de usuario para C++.

1. Instalación

Para instalar FLTK en un sistema operativo UNIX (Linux ó Mac OSX), debemos realizar los siguientes pasos:

- Descargar FLTK de `http://www.fltk.org`, la última versión estable (1.3.3).
- Descomprimir el archivo con el siguiente comando:
  $ tar zxf fltk-1.3.3.tar.gz
- Acceder a la carpeta fltk-1.3.3 descomprimida y correr los siguientes comandos para compilar e instalar fltk.
  $ ./configure
  $ make
  $ make install

Si ocurrió algún error al compilar el programa pruebe descargando la versión de desarrollador 1.3.x, disponible en la misma página.

2. Creando tu primer programa FLTK

Todos los programas deben incluir el archivo ¡FL/Fl.H¿. Además deben incluir un header por cada clase de FLTK que se use. Por ejemplo, el siguiente código despliega el típico "Hello, World!" en una ventana.

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
int main(int argc, char **argv) {
  Fl_Window *window = new Fl_Window(340,180);
  Fl_Box *box = new Fl_Box(20,40,300,100,"Hello, World!");
  box->box(FL_UP_BOX);
  box->labelfont(FL_BOLD+FL_ITALIC);
  box->labelsize(36);
  box->labeltype(FL_SHADOW_LABEL);
  window->end();
  window->show(argc, argv);
  return Fl::run();
}
```

Después de incluir los header, el programa crea una ventana (Window). Donde luego todos los siguientes Widgets serán automáticamente hijos de esta ventana.

```
Fl_Window *window = new Fl_Window(340,180);
```

Luego creamos una caja (Box) con el texto de "Hello, World!". FLTK agregará automáticamente esta caja a la ventana.

```
Fl_Box *box = new Fl_Box(20,40,300,100,"Hello, World!");
```

Siguiendo, asignamos el tipo de caja y el tamaño, fuente y estilo del texto o etiqueta.

```
box->box(FL_UP_BOX);
box->labelfont(FL_BOLD+FL_ITALIC);
box->labelsize(36);
box->labeltype(FL_SHADOW_LABEL);
```

Le decimos al programa que no agregaremos más widgets a la ventana.

```
window->end();
```

Y finalmente, mostramos la ventana y corremos el programa.

```
window->show(argc, argv);
return Fl::run();
```

El resultado de este programa será una ventana como se muestra en la figura 1.

Figura 1: Primer programa "Hello, World!"

2.1. Algunos widget básicos

Fl_Window , genera una ventana.
Fl_Window(int W, int H, const char* TITLE = 0)

Fl_Button , genera un callback al ser clickeado por el usuario.
Fl_Button(int X, int Y, int W, int H, const char * L = 0)

Fl_Box , dibuja una caja y puede llevar un texto.
Fl_Box (int X, int Y, int W, int H, const char *l=0)

Fl_Input , dibuja un cuadro de entrada de texto.
Fl_Input (int X, int Y, int W, int H, const char *l=0)

Fl_Output , dibuja un cuadro que contiene un texto.
Fl_Output (int X, int Y, int W, int H, const char *l=0)

3. Usando callbacks

Todo widget tiene un único callback, el cuál se acciona dependiendo del tipo de widget que estemos utilizando.
En el siguiente ejemplo, usaremos el callback del widget Button, el cual se acciona al ser presionado.

```
#include <iostream>
#include <FL/Fl.H>
#include <FL/Fl_window.H>
#include <FL/Fl_Button.H>
using namespace std;

void button_cb( Fl_Widget*, void* );

void make_window()
{
   Fl_Window* win= new Fl_Window( 100,100, "Button Toggle" );
   win->begin();
      Fl_Button* but = new Fl_Button( 30, 30, 40, 40, "ON" );
   win->end();
   but->callback( ( Fl_Callback* ) button_cb );
   win->show();
}

void button_cb( Fl_Widget* obj , void* )
{
   fprintf(stderr, "%s\n", obj->label()); // Debug
   if(strcmp( obj->label(),"ON") == 0 )
      obj->label( "OFF" );
   else
      obj->label( "ON" );
   obj->redraw();
}

int main( int argc, char* argv[] )
{
   make_window();
   return Fl::run();
}
```

En este ejemplo agrupamos toda la parte de interfaz gráfica en la función *make_window()* y definimos la función *button_cb()* como nuestro callback, el cual se ejecutará cada vez que presionemos el botón.

4. Creando nuestro propio widget

Para poder crear y dibujar nuestros propios widget es necesario implementar un nuevo tipo de dato que herede de la clase Widget o sus hijos.
En este ejemplo crearemos la clase My_Box, en la cual sobre escribiremos la función draw, que es la que dibuja el widget en la ventana.

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Enumerations.H>
#include <FL/fl_draw.H>
#include <FL/x.H>

class My_Box : public Fl_Box {
  void draw() {
    fl_color(FL_BLACK);
    int x1 = x(),        y1 = y();
    int x2 = x()+w()-1, y2 = y()+h()-1;
    fl_line(x1, y1, x1, y2);
    fl_line(x1, y2, x2, y2);
    fl_rect(x1+10, y2-110, 100, 100);
    fl_color(FL_RED);
    fl_circle(x1+10,y2-110,3);
    fl_circle(x1+110,y2-10,3);
  }
public:
  My_Box(int x, int y, int w, int h, const char *l=0) : Fl_Box(x,y,w,h,l){}
};

int main(int argc, char **argv) {
  Fl_Window *window = new Fl_Window(400,400);
  My_Box *box = new My_Box(50,50,300,300);
  window->end();
  window->show(argc, argv);
  return Fl::run();
}
```

5. Compilando nuestros programas

Para compilar estos programas en UNIX, probablemente haya que decirle al compilador donde se encuentran los headers de FLTK. Esto se hace generalmente agregando la opción -I.

```
$ c++ hello.cxx -I/usr/local/include
```

Y lo mismo para utilizar las librerías de FLTK.

```
$ c++ hello.cxx -L/usr/local/lib -lfltk -lXext -lX11 -lm
```

Lo cual resulta muy engorroso y tedioso a la hora de compilar nuestros programas, por lo que utilizaremos una herramienta llamada CMake para poder compilar nuestros programas de forma automática con una serie de opciones previamente programadas.

5.1. Instalación de CMake

Para instalar CMake en Linux basta con utilizar el gestor de repositorios *apt-get*.

```
$ sudo apt-get install cmake
```

En caso homólogo, en Mac OSX podemos utilizar brew para el mismo efecto.

```
$ brew install cmake
```

Para otros métodos de instalación, consulta directamente en la página `http://www.cmake.org/install`.

5.2. Preparando el proyecto

Para entregarle todas las instrucciones necesarias a CMake para que pueda compilar nuestro programa sin problemas, debemos crear un archivo llamado *CMakeLists.txt*.
En este caso sólo utilizaremos las sentencias básicas para compilar el programa hello.cxx, nuestro primer programa.

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.6)

PROJECT(hello)

FIND_PACKAGE(FLTK REQUIRED)
FIND_PACKAGE(OpenGL REQUIRED)

ADD_EXECUTABLE(hello hello.cxx)
TARGET_LINK_LIBRARIES(hello FLTK_LIBRARIES)
TARGET_LINK_LIBRARIES(helloOPENGL_LIBRARIES)
```

5.3. Compilando

Una vez preparado el proyecto, corremos CMake para que haga la compilación del proyecto.

```
$ cmake .
```

Y finalmente compilamos nuestro programa.

```
$ make
```

Estos pasos nos generarán un ejecutable en la carpeta del proyecto, el cual podemos correr de la siguiente forma.

```
$ ./hello
```

6. Otros ejemplos útiles

6.1. Mover objetos

```
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Scroll.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Pixmap.H>
#include <stdio.h>
//
// Demonstrate user-movable boxes in a scroll region
// erco@netcom.com 08/06/02
// erco@3dsite.com 01/06/05 -- added call to Fl_Box::handle()
//
static char *cat_xpm[] = {                              // XPM
"50 34 4 1",
"  c black",
"o c #ff9900",
"@ c #ffffff",
"# c None",
"##################################################",
"###      ##################################      ###",
"### oooo  ##############################  oooo ###",
"### oo  oo  ############################  oo  oo ###",
"### oo    oo  ##########################  oo    oo ###",
"### oo     oo  ########################  oo     oo ###",
"### oo      oo  ######################  oo      oo ###",
"### oo       oo                      oo       oo ###",
"### oo        oo  oooooooooooooooooo  oo        oo ###",
"### oo         oooooooooooooooooooooo         oo ###",
"### oo      ooooooooooooooooooooooooooo      ooo ###",
"#### oo   ooooooo ooooooooooooo ooooooo   oo  ####",
"####  oo ooooooo ooooooooooooo ooooooo oo  ######",
"#####  o ooooooooooooooooooooooooooooo o  #######",
"######ooooooooooooooooooooooooooooooo########",
"#####ooooooooo      ooooooooo      ooooooooo######",
"#####ooooooooo  @@@  ooooooo  @@@  ooooooooo######",
"#####ooooooooo @@@@@ ooooooo @@@@@ ooooooooo######",
"#####ooooooooo @@@@@ ooooooo @@@@@ ooooooooo######",
"#####ooooooooo  @@@  ooooooo  @@@  ooooooooo######",
"#####ooooooooo      ooooooooo      ooooooooo######",
"######oooooooooooooo       ooooooooooooooo#######",
"######oooooooo@@@@@@@     @@@@@@@oooooooo#######",
"######ooooooo@@@@@@@@@   @@@@@@@@@ooooooo#######",
"#######ooooo@@@@@@@@@@@ @@@@@@@@@@@ooooo########",
"#########oo@@@@@@@@@@@@@ @@@@@@@@@@@@oo##########",
"##########o@@@@@@ @@@@@ @@@@@ @@@@@@o###########",
"###########@@@@@@@     @     @@@@@@@############",
"#############  @@@@@@@@@@@@@@@@@@@  ##############",
"################  @@@@@@@@@@@@@@@  ################",
"##################   @@@@@@@@@   ##################",
"####################       ####################",
"##################################################",
};

Fl_Double_Window *G_win    = NULL;
Fl_Scroll        *G_scroll = NULL;
static Fl_Pixmap  G_cat(cat_xpm);

#define BOXWIDTH  80
#define BOXHEIGHT 50
```

```
// A 'MOVABLE' BOX
class Box : public Fl_Box {
protected:
    int handle(int e) {
        static int offset[2] = { 0, 0 };
        int ret = Fl_Box::handle(e);
        switch ( e ) {
            case FL_PUSH:
                // save where user clicked for dragging
                offset[0] = x() - Fl::event_x();
                offset[1] = y() - Fl::event_y();
                return(1);
            case FL_RELEASE:
                return(1);
            case FL_DRAG:
                // handle dragging
                position(offset[0]+Fl::event_x(), offset[1]+Fl::event_y());
                G_win->redraw();
                return(1);
        }
        return(ret);
    }
public:
    Box(int X, int Y, int W, int H, const char *L=0) : Fl_Box(X,Y,W,H,L) {
        image(G_cat);
        box(FL_UP_BOX);
        color(FL_GRAY);
    }
    Box(int X, int Y) : Fl_Box(X,Y,BOXWIDTH,BOXHEIGHT,0) {
        image(G_cat);
        box(FL_UP_BOX);
        color(FL_GRAY);
    }
};

/// MAIN
int main() {
    G_win = new Fl_Double_Window(420,300);
    G_scroll = new Fl_Scroll(10,10,420-20,300-20);
    G_scroll->box(FL_FLAT_BOX);
    G_scroll->color(Fl_Color(46));
    G_scroll->begin();
    {
        // CREATE NEW BOXES ON THE SCROLLABLE 'DESK'
        for ( int x=20; x<=G_scroll->w()-BOXWIDTH; x+= BOXWIDTH+20)
            for ( int y=20; y<=G_scroll->h()-BOXHEIGHT; y+= BOXHEIGHT+20)
                new Box(x,y);
    }
    G_scroll->end();
    G_win->resizable(G_win);
    G_win->show();
    return(Fl::run());
}
```

6.2. Input y Output

```
#include <iostream>
#include <FL/Fl.H>
#include <FL/Fl_window.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Output.H>
#include <cstdlib>
using namespace std;

void copy_cb( Fl_Widget*, void* );
void close_cb( Fl_Widget*, void* );
void make_window();

int main( int argc, char* argv[] )
{
   make_window();
   return Fl::run();
}

void make_window()
{
   Fl_Window* win= new Fl_Window(600,250, "C++ FLTK Buttons");

   win->begin();
      Fl_Button*  copy = new Fl_Button( 350, 0,  70, 30, "C&opy" ); //child 0
      Fl_Button* close = new Fl_Button( 430, 0,  70, 30, "&Quit" );   //child 1
      Fl_Input*    inp = new Fl_Input( 20, 0, 140, 30, "In" );        //child 2
      Fl_Output*   out = new Fl_Output( 200, 0, 140, 30, "Out" );  //child 3
   win->end();

   copy->callback( (Fl_Callback*) copy_cb );
   close->callback( (Fl_Callback*) close_cb);
   win->show();
 }

void copy_cb( Fl_Widget* obj , void* )
{
   Fl_Button* button=(Fl_Button*)obj;
   const char* temp;
   temp = (  (Fl_Input*)(button->parent()->child(2))  )->value();
   (  (Fl_Output*)(button->parent()->child(3))  )->value(temp);
}

void close_cb( Fl_Widget* obj, void*)
{
   exit(0);
}
```