FLTK. Быстрое введение

Прууэл Э. Р.
pru@hydro.nsc.ru

13 февраля 2022 г.

Содержание

Аннотация

Приводится короткое описание кросс-платформенной библиотеки FLTK (www.fltk.org) на языке программирования c++ для построения графического интерфейса пользователя (GUI). FLTK представляет собой библиотеку виджетов и работает на ОС UNIX/Linux X11, Microsoft Windows и MacOS X.

Обсуждаются основные концепции применения библиотеки. Рассматривается несколько примеров использования FLTK.

1 Введение

Количество библиотек для создания графического пользовательского интерфейса (GUI) необозримо велико, и по-видимому, такая ситуация сохранится и в ближайшем будущем. Поэтому, вопрос, в изучение какой библиотеки вложиться представляется весьма не тривиальным.

В зависимости от требований решаемых задач можно предложить несколько подходов.

- Попытаться вообще избежать использования графической библиотеки. Изучение низкоуровневых библиотек это непозволительно дорого, и окупится только если этим специально заниматься и на больших масштабах работ.

  В большинстве случаев, использование командной строки и файлов с командами позволяет решить проблему управления программой. А графический анализ полученных результатов, можно осуществить и другими средствами.

- Использовать специализированную библиотеку, адаптированную для ваших специфических нужд. При хорошем выборе Вы сможете почти без затрат на изучение нового средства легко решать свои проблемы.

- Если ни одно из решений не подошло — значит Вы столкнулись с серьезной проблемой, и для ее решения все-же придется разобраться с одной или даже несколькими графическими библиотеками общего назначения.

Хорошим решением является использование комбинации из представленных подходов (рис. 1). Начинаете использовать специализированную библиотеку (My lib), если ее возможностей не хватает, опускаетесь глубже к более общей библиотеке или к api используемой графической системы. При хорошей архитектуре библиотеки Вам потребуются знания только об используемом уровне.

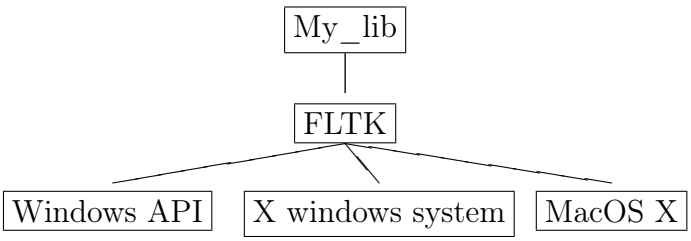

Рис. 1: Вертикальная структура библиотек.

Рассмотрим подробнее последнее предложение. Очень важный вопрос какую библиотеку выбрать. Приведем некоторую сравнительную таблицу.

Название	Функциональность	Простота освоения	Документированность и стройность	Тяжеловесность
FLTK	слабая	отличная	отличная	легкая
WxWidgets	хорошая	хорошая	хорошая	легкая
GTK	отличная	слабая	слабая	средняя
Qt	отличная	отличная	отличная	тяжелая

FLTK www.fltk.org Легкая в освоении и использовании, не требовательна к ресурсам. Идеально подходит для небольших проектов. Эти аргументы стали определяющими при выборе ее для ознакомления. Недостаток один, при интенсивном ее использовании очень скоро станет не хватать предоставляемых средств со сложным поведением.

Qt www.trolltech.com По-видимому, единственная библиотека в изучение которой стоит серьезно вкладываться. Очень популярна, хорошо развивается, из недостатков: очень требовательна к ресурсам на стадии сборки, и до недавнего времени имела значительные лицензионные ограничения[1].

[1] С 2005 года распространяется GPL версия библиотеки Qt.

WxWidgets www.wxwidgets.org Достаточно гибкая, функциональная и легковесная библиотека. В основном низкоуровневые ее средства удобны для конструирования высокоуровневых библиотек. Хорошо подойдет для тех кто хочет разобравшись с массой подробностей создать средство с необходимыми ему свойствами.

GTK www.gtk.org C библиотека со всеми вытекающими плюсами и минусами. Может все, но сложна в использовании. Имеет смысл использовать ее, если вы создаете долгосрочный проект.

2 Основные концепции

В отличии от консольных приложений, исполняющихся, в каком-то смысле, последовательно, строчка за строчкой, работа программы с графическим интерфейсом не последовательна, а больше похожа на набор разрозненных реакций приложения на действия пользователя. Каждый раз, когда Вы нажимает на кнопку, двигаем курсор или редактирует поле ввода, графическая система посылает соответствующие сообщения приложению, которое последовательно обрабатывает их, вызывая каждый раз соответствующую функцию. Поэтому и написание программы с графическим интерфейсом состоит в определении ряда функций, которые будут вызываться при соответствующих обстоятельствах.

Из сказанного следует. Вы не можете в произвольном месте программы нарисовать что-либо на экране, а должны определить функцию, которая будет вызываться всякий раз, когда необходимо обновить область окна. Вы не можете написать цикл ожидания нажатия кнопки на клавиатуре или движения мыши, а должны определить соответствующий обработчик события.

Объектно–ориентированные библиотеки предоставляют набор готовых классов–виджетов, отражающих видимые на экране атрибуты интерфейса: кнопки, окошки, меню, диалоги и т. п. Пользователь, используя композицию, наследование и перегрузку виртуальных методов создает новые классы с требуемой функциональностью.

Часть иерархии классов предоставляемых FLTK, представлена на рисунке 2.

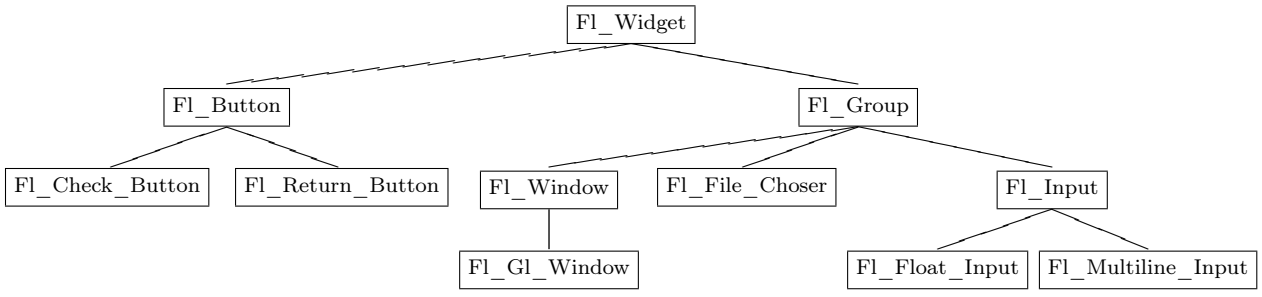

Рис. 2: Часть иерархии классов FLTK библиотеки.

3 Несколько примеров

Приведем несколько примеров демонстрирующих стиль использования FLTK.

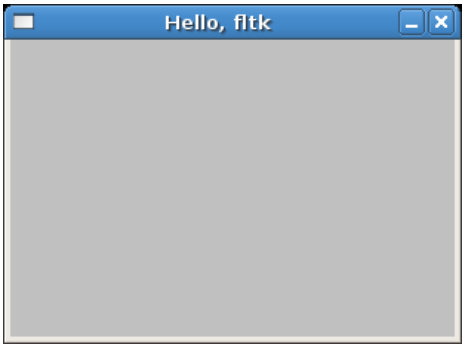

Рис. 3: Внешний вид минимального fltk приложения.

3.1 Простейшее приложение

Пример кода и внешний вид простейшего приложения (рис. 3).

```
1  // fl01.cpp
2  // Простейшее приложение с использованием FLTK
3  #include <FL/Fl.H>                          // базовые части fltk приложения
4  #include <FL/Fl_Window.H>                   // класс с простейшим окошком
5
6  int main(){
7      Fl_Window win(300, 200, "Hello, fltk"); // создаем окно
8      win.show();                             // отображаем окно
9      return Fl::run();                       // запускаем обработчик событий
10 }
```

```
int main(){
```

Как всегда, работа приложения начинается с исполнения функции `main`.

```
Fl_Window win(300, 200, "TEST");
```

Создаем окошко, которое является объектом класса `Fl_Window`. Аргументы конструктора определяют ширину, высоту и подпись окошка соответственно.

```
win.show();
```

Вызванный метод метит окошко как видимое. По умолчанию все виджеты создаются невидимыми и появляются на экране только после вызова метода `show()`.

```
Fl::run();
```

Внутри этой функции происходит вся жизнь приложения. Именно сюда попадают сообщения о происходящих событиях и вызываются соответствующие обработчики. Функция заканчивает свою работу как-только закрывается последнее видимое окно.

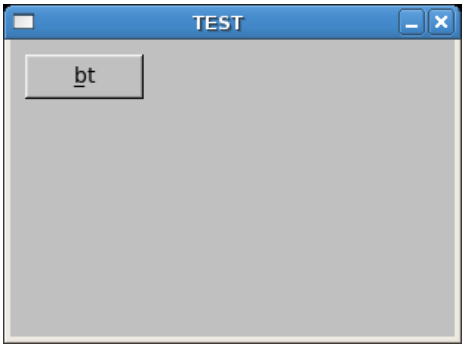

Рис. 4: Пример fltk приложения с кнопкой.

3.2 Окно с кнопкой

Чтобы сделать что-то полезное – надо вызвать функцию.

Пример кода и внешний вид приложения с кнопкой (рис. 4).

```
1  // fl02.cpp
2  // Окно с кнопкой и обработчиком событий
3  #include <FL/Fl.H>
4  #include <FL/Fl_Window.H>
5  #include <FL/Fl_Button.H>
6
7  // функция для вызова при нажитии на кнопку
8  void fbt(Fl_Widget *w, void *) {
9      if (w->color()!=FL_RED) {
10         w->color(FL_RED);
11         return ;
12     }
13     if (w->color()!=FL_GREEN) {
14         w->color(FL_GREEN);
15         return ;
16     }
17 }
18
19 int main(){
20     Fl_Window win(300, 200, "TEST");        // создаем окно
21     Fl_Button bt(10, 10, 80, 30, "&bt");    // добавляем кнопку
22     bt.callback(fbt,0);  // подключаем обработчик события нажатия на кнопку
23     win.show();                             // отображаем окно
24     return Fl::run();                       // запускаем обработчик событий
25 }
```

При вызове конструктора объекта класса `Fl_Window` и вообще `Fl_Group` открывается контекст, и все последующие виджеты создаются принадлежащими окну, в частности, первые два аргумента конструктора задают положение кнопки относительно верхнего левого угла окна. В общем, случае открытие и закрытие контекста регулируется вызовом методов `Fl_Gruop::begin()` и `Fl_Group::end()`.

```
bt.callback(fbt,0);
```

Устанавливаем функцию для вызова при нажатии кнопки.

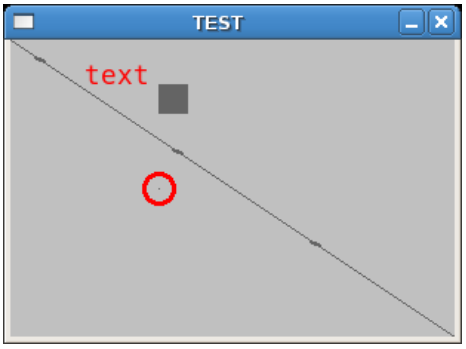

Рис. 5: Пример перегрузки метода draw

3.3 Рисование примитивов

Чтобы уточнить внешний вид виджета, надо перегрузить метод `draw()` у виджета. Пример кода и внешний вид приложения с отрисовкой примитивов (рис. 5).

```
1  // fl03.cpp
2  // M_Win окошко с переопределенным методом draw
3  #include <FL/Fl.H>
4  #include <FL/Fl_Window.H>
5  #include <FL/fl_draw.H>
6
7  class M_Win:public Fl_Window{
8   public:
9      M_Win(int w, int h, const char *name="M_Win"): Fl_Window(w, h, name){
10         resizable(this);
11         show();
12     }
13     void draw(){
14         Fl_Window::draw();
15         fl_color(100, 100, 100);
16         fl_point(100, 100);
17         fl_line(0, 0, w(), h());
18         fl_rectf(100, 30, 20, 20);
19         fl_color(FL_RED);
20         fl_line_style(FL_SOLID, 3);
21         fl_circle(100, 100, 10);
22         fl_font(FL_COURIER,18);
23         fl_drawРусский(" text", 50, 30);
24     }
25
26 };
27
28 int main(){
29    M_Win win(300, 200, "TEST");
30    return Fl::run();
31 }
```

3.4 Обработка событий

Пример кода обработки нажатия клавиш клавиатуры.

```
1  // fl04.cpp
2  // Обработка события нажатие кнопки
3  #include <FL/Fl.H>
4  #include <FL/Fl_Window.H>
5  #include <FL/Fl_Button.H>
6  #include <FL/fl_draw.H>
7
8  class M_Win:public Fl_Window{
9   public:
10      int x, y, dx;
11      M_Win(int w, int h, const char *name="M_Win"): Fl_Window(w, h, name){
12          show();
13          x=y=0;
14          dx=10;
15      }
16      void draw(){
17          Fl_Window::draw();
18          fl_color(100, 100, 100);
19          fl_rectf(x,y, 20, 20);
20      }
21      int handle(int event){
22           if (event==FL_KEYDOWN){ // FL_KEYUP
23               switch (Fl::event_key()){
24                    case FL_Up:    y-=dx; redraw(); return 1;
25                    case FL_Down:  y+=dx; redraw(); return 1;
26                    case FL_Right: x+=dx; redraw(); return 1;
27                    case FL_Left:  x-=dx; redraw(); return 1;
28
29                    default : Fl_Window::handle(event);
30               }
31           }
32           return Fl_Window::handle(event);
33      }
34 };
35
36 int main(){
37    M_Win win(300, 200, "TEST");
38    return Fl::run();
39 }
```

3.5 Работа с таймером

Для одновременной работы интерфейса и параллельного проведения длительных расчетов, можно использовать таймер.

```
1  // fl05.cpp
2  // Работа с таймером
3  #include <FL/Fl.H>
4  #include <FL/Fl_Window.H>
5  #include <FL/fl_draw.H>
6
7  class M_Win:public Fl_Window{
8   public:
9      M_Win(int w, int h, const char *name="M_Win"):Fl_Window(w, h, name){
10         xx=0;
11         show();
12     }
13     int xx;
14     void draw(){
15         Fl_Window::draw();
16         fl_color(0, 0, 0);
17         fl_rectf(xx, 50, 20, 20);
18     }
19 };
20
21 void callback(void* winp) {
22    M_Win * win=(M_Win *)winp;
23    win->xx++;
24    win->redraw();
25    Fl::repeat_timeout(0.2,callback, win);
26 }
27
28 int main(){
29    M_Win win(300, 200, "TEST");
30    Fl::add_timeout(1.0, callback, &win); // создаем таймер
31    return Fl::run();
32 }
```

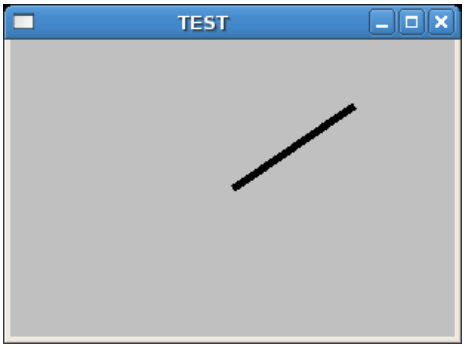

Рис. 6: Длительные вычисления и интерфейс

3.6 Длительные вычисления

Пример кода и внешний вид приложения с вычислениями и тнтерфейсом (рис. 6).

```
1  // fl06.cpp
2  // Тяжелые вычисления
3  #include <cmath>
4  #include <FL/Fl.H>
5  #include <FL/Fl_Double_Window.H>
6  #include <FL/fl_draw.H>
7  double phi;
8  class M_Win:public Fl_Double_Window{
9   public:
10     M_Win(int w, int h):Fl_Double_Window(w, h, "M_Win"){
11         resizable(this);
12         show();
13     }
14     void draw(){
15         Fl_Window::draw();
16         fl_color(0, 0, 0);
17         fl_line_style(0, 5);
18         fl_line(w()/2, h()/2,
19         w()/2+int(100*cos(phi)), h()/2-int(100*sin(phi)));
20     }
21 };
22 void callback(void* winp) {
23    M_Win * win=(M_Win *)winp;
24    phi+=1e-4; // квант необходимых вычислений
25    win->redraw();
26 }
27 int main(){
28    M_Win win(300, 200);
29    // запускать в промежутке между обработкой событий
30    Fl::add_idle(callback, &win);
31    return Fl::run();
32 }
```

4 Упражнения

Несколько простых упражнений для знакомства с библиотекой FLTK.

1. Посетите сайт https://www.fltk.org. Посмотрите закладки Screenshots и Documentation (FLTK 1.3.8). Найдите описание классов `Fl_Widget, Fl_Window, Fl_Button, Fl_Input`. Найдите описание функций `fl_color, fl_line, fl_point, fl_draw`.

2. В директории fltk-1.3.8/tests запустите несколько примеров, посмотрите файлы с реализацией.

3. Поработайте с примерами в директории examples. Рекомендуется последовательно познакомиться со всеми примерами.

Проекты для самостоятельной работы.

1. Напишите интерфейс для игры в крестики-нолики.

2. Напишите программу рисующую простые геометрические фигуры с задаваемыми свойствами. Обеспечьте ее необходимыми полями ввода и вывода.

3. Напишите программу моделирующую простое физическое явление.

4. Обсудите свое предложение с преподавателем.

5 Установка библиотеки и компиляция программы

5.1 Microsoft visual studio

```
## makefile_msvc
FLTKDIR=D:/tmp/src
CXXFLAGS=/MD /nologo /I$(FLTKDIR)
LIBS=/link $(FLTKDIR)\\lib\\fltk.lib \
   wsock32.lib comctl32.lib kernel32.lib user32.lib gdi32.lib \
   winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib \
   oleaut32.lib uuid.lib

.cxx.exe:
   cl $(CXXFLAGS) $*.cxx $(LIBS)

all: fl01
```

5.2 GNU утилиты

```
g++ main.cpp -lfltk
```