This guide is meant to be shared between CSCE121 and TAMU students in assistance to the March 28th lab drill. This guide comes with no warranty and no guarantees of success. Please also be mindful of variable names, coding style, etc in your programs when referring to my file, as too similar of work or direct copies can be seen as cheating. Thanks & Gig 'Em

CONTACT
Oneal Abdulrahim
onealabdul@tamu.edu

SUMMARY

The light bulb lab is very simple, though it requires an understanding of what the heck is going on in Bjarne's graphics library. We're essentially making our own Window object, meaning we will give it different attributes than just a **`Simple_window`**, like just an ON/OFF button and a **`bool`** representing the light bulb's state.

Links to files you will need:

- My own completed lab (injected with comments everywhere so you're motivated to write your own instead)
- Bjarne's chapter 16.5 example code. Begin here and edit accordingly, just like Dr. Daugherity said in the description on Piazza.

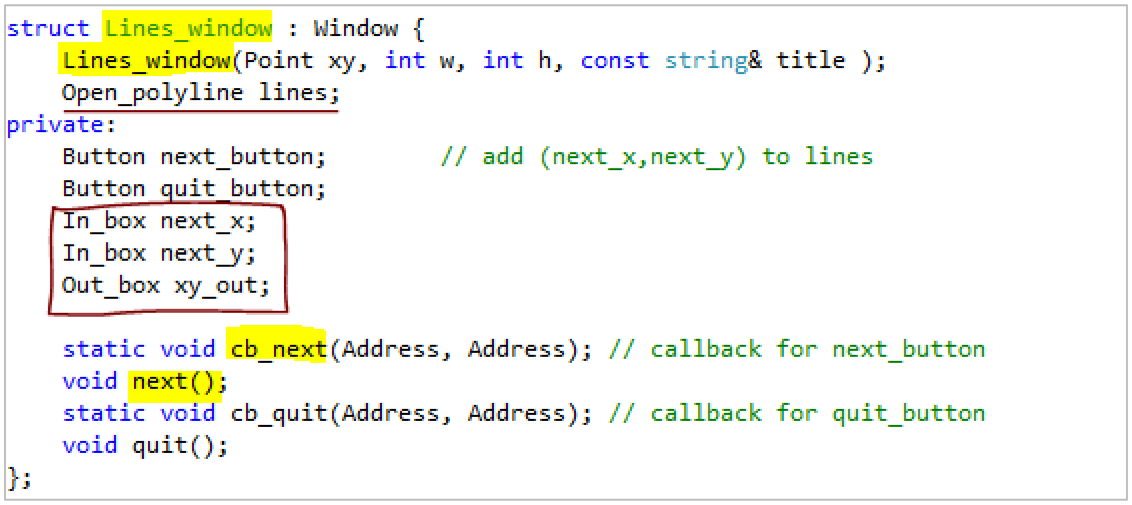
We shall go over Bjarne's version of the file, method by method, with highlighting where changes need to occur. You may refer to my file for further comments, though I encourage you to write your own to demonstrate the knowledge to yourself.

The highlighted components you just need to change a variable name to better fit your program. You can technically still call it a **`Lines_window`**, but I would suggest **`Bulb_window`** or something more meaningful. The components indicated in maroon are those that we will comment on below each screenshot. **Hopefully not all of it is giving you trouble, so just jump to the methods that are confusing and I hope it will work! (They are in same order as the file)**

Issues with references to Window

Ensure you are not using the **`std`** namespace, or that any references to objects in the **`Graph_lib`** namespace are preceded by **`Graph_lib::Object`**…

Class Declaration

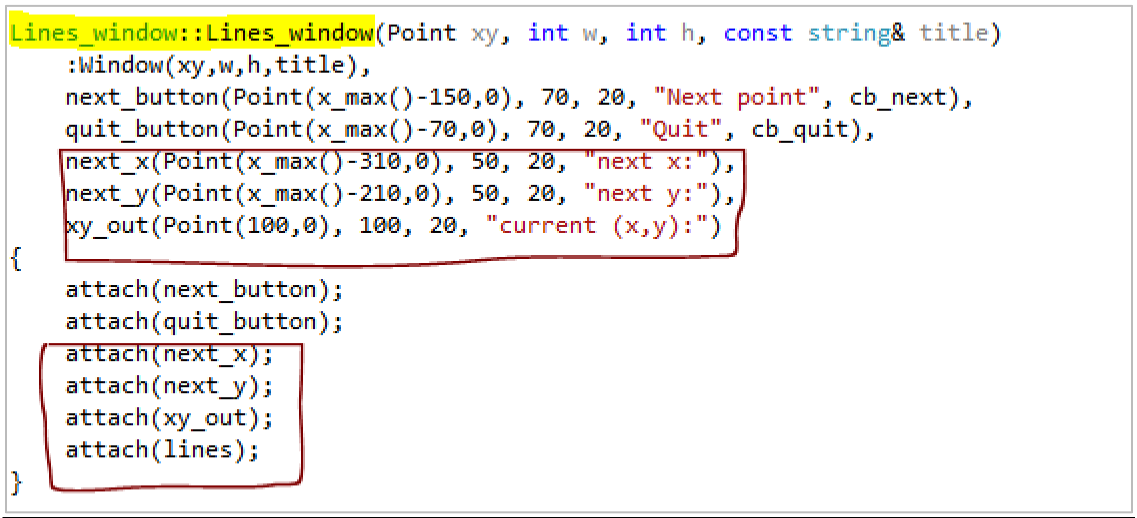


```
struct Lines_window : Window {
    Lines_window(Point xy, int w, int h, const string& title );
    Open_polyline lines;
private:
    Button next_button;        // add (next_x,next_y) to lines
    Button quit_button;
    In_box next_x;
    In_box next_y;
    Out_box xy_out;

    static void cb_next(Address, Address); // callback for next_button
    void next();
    static void cb_quit(Address, Address); // callback for quit_button
    void quit();
};
```

- Declare your variables here. You don't need **Open_polyline** lines or any of the extra **In_box** or **Out_box** textboxes.
- You may add any shapes or variables you feel would fit the description of every **Bulb_window** that's ever made. Consider: you will need a **Circle** object to represent the bulb, a switch **Button**, and a quit **Button**. You should declare a **bool** here as a flag for your switch state.
- The static methods signal to FLTK that when a button is pressed, a "callback", or reference to, a method needs to be made. If you do not understand this, just change **cb_next** and **next()** to **cb_switch** and **switch**. The quit method you can also leave, as it will give us a quit button to work with.

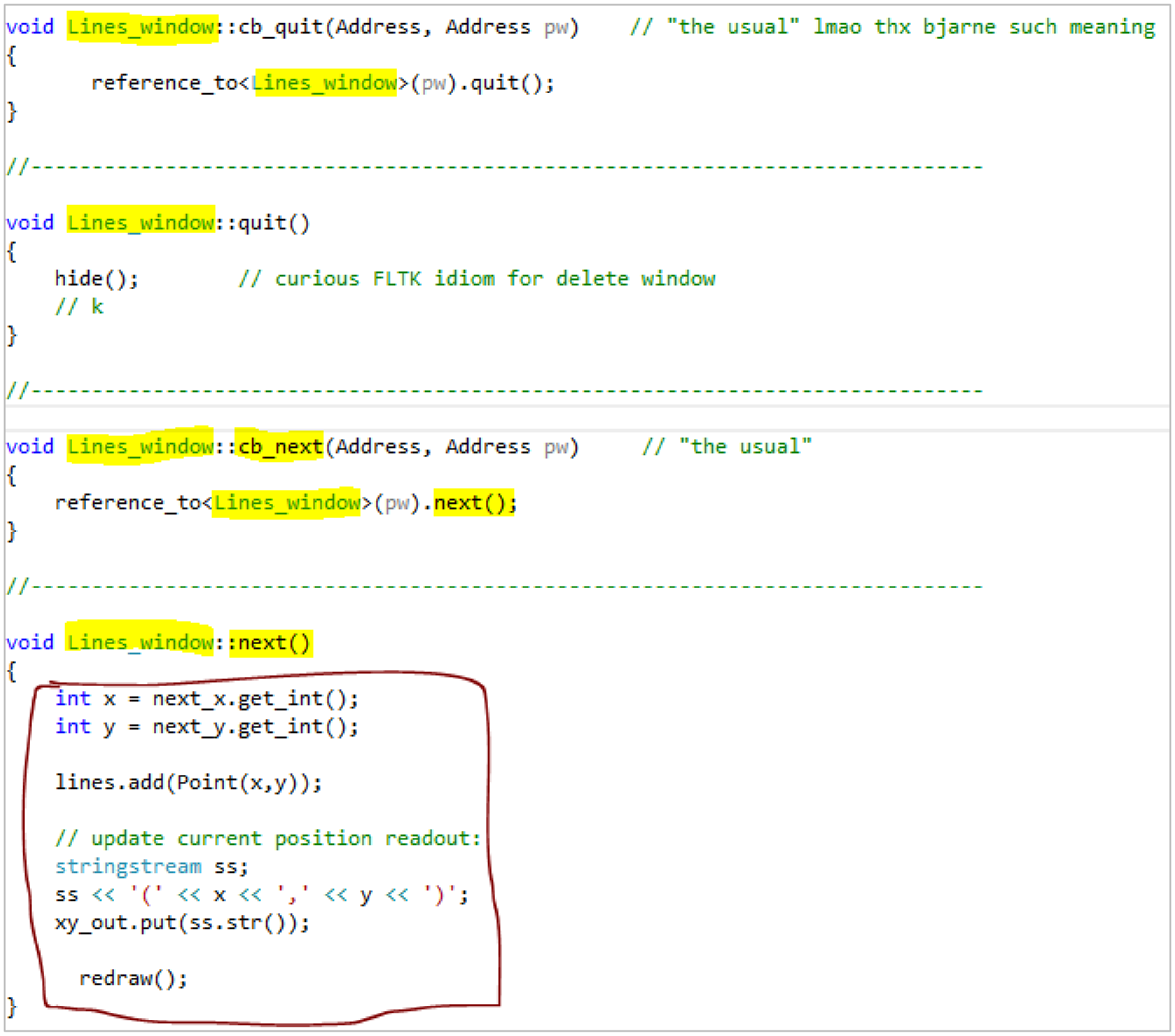
Constructor



```
Lines_window::Lines_window(Point xy, int w, int h, const string& title)
    :Window(xy,w,h,title),
    next_button(Point(x_max()-150,0), 70, 20, "Next point", cb_next),
    quit_button(Point(x_max()-70,0), 70, 20, "Quit", cb_quit),
    next_x(Point(x_max()-310,0), 50, 20, "next x:"),
    next_y(Point(x_max()-210,0), 50, 20, "next y:"),
    xy_out(Point(100,0), 100, 20, "current (x,y):")
{
    attach(next_button);
    attach(quit_button);
    attach(next_x);
    attach(next_y);
    attach(xy_out);
    attach(lines);
}
```

- Not much needs to change here. Think: "every time someone makes a **Bulb_window**, this method is called". It's the constructor for your window.
- If you come from Java, everything after the single colon **:** can be dumped into the method body and initialized using the **this** keyword. However, following the C++ syntax is clearer here.
- So, initialize your objects (**Buttons**, **Circle**, **Rectangles** etc...) and attach them in the method body

Helper Methods



```
void Lines_window::cb_quit(Address, Address pw)    // "the usual" lmao thx bjarne such meaning
{
       reference_to<Lines_window>(pw).quit();
}

//------------------------------------------------------------------------------

void Lines_window::quit()
{
    hide();        // curious FLTK idiom for delete window
    // k
}

//------------------------------------------------------------------------------

void Lines_window::cb_next(Address, Address pw)     // "the usual"
{
    reference_to<Lines_window>(pw).next();
}

//------------------------------------------------------------------------------

void Lines_window::next()
{
    int x = next_x.get_int();
    int y = next_y.get_int();

    lines.add(Point(x,y));

    // update current position readout:
    stringstream ss;
    ss << '(' << x << ',' << y << ')';
    xy_out.put(ss.str());

      redraw();
}
```

- Anything labelled **cb_** is a callback, and points to a method that the program should run when a button is clicked. So, **cb_next** points to the **next()** method, written just below it.
- These methods must include at least the switcher. In the maroon box, you should write your switcher implementing the usage of the class boolean variable. You can delete Bjarne’s code or just remove his method and write your own completely.
- For the quit method, you can leave it as is because it would do the same thing.