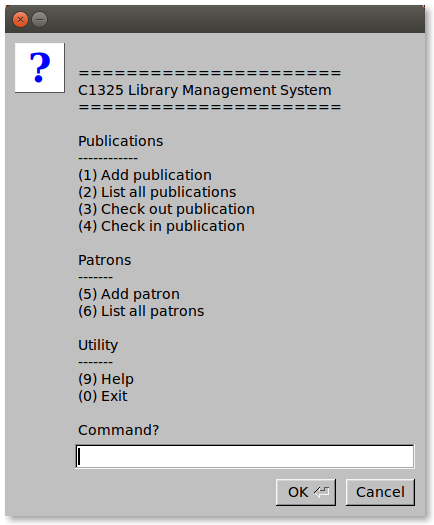
Getting Organized–Video Store Model

Sprint #2

CSE 1325 – Fall 2017 – Homework #5

Due Sunday, October 8th at Noon

Command Line Interfaces (CLI) can be very productive for technical people, but are not acceptable to normal users. They prefer rich, graphical interfaces, and our job as professional software developers is to provide interfaces that delight our users. But not this week. Instead, we'll start by implementing a literal replacement of the console menus and inputs with pre-defined FLTK dialogs, mainly fl\_input (for commands and data prompts) and fl\_message (for reports). Below are some examples using a library model.



# 

# Requirements

Port your solution (or the suggested solution provided with this assignment, if you prefer), to an all-GUI, dialog-only implementation. For this sprint, you need use only the pre-defined FLTK dialogs discussed in class and documented at <http://www.fltk.org/doc-1.3/group__group__comdlg.html>.

(**You will receive partial credit if you only implement a portion of the requirements.** You will be graded on the extent to which you cover the requirements and the quality of your code. **For FLTK programs only,** you are permitted – but not required – to put all of your code into a single source code file. Use Scrum to manage this second sprint. If you don't understand the *intent* of a requirement, feel free to ask – although reasonable assumptions without asking are also fine if you've used a library.)

# Suggested Approach

FLTK can be... temperamental. Therefore, I recommend the following approach.

1. If you wish to modify your own solution to the previous assignment, assemble all of the code into a single file **named abc1234\_video\_store\_gui.cpp**. Place your header files (.h) first (stripping the preprocessor directives), following each with its implementation (.cpp). Migrate all of the #includes to the top of your single source file, removing duplicates.
2. FLTK 1.3.3 (install with command from the slides) appears to have a stability bug when displaying only pre-built dialogs, without an Fl\_Window visible. The symptom is that the program immediately generates a “segmentation fault (core dumped)” message to STDERR and exits.  
     
   To work around this, **add the following two lines of code immediately following “int main() {“**, *exactly as shown.* This will create a 1 pixel window on the screen to avoid the bug. (Google “web beacon” if you're curious as to the variable name.)  
     
    **Fl\_Window beacon(1,1);  
    beacon.show();**  
     
   You do **NOT** need to end your main() with “return Fl::run();” for this assignment, because you are supplying your own main loop to control the dialogs.  
     
   (Your other option to avoid this bug, if you're really brave, is to download and build FLTK 1.3.4-1 from source. This is left as an exercise for the ambitious student, though I note that it built perfectly on my machine using a single-word command – wait for it... wait for it... “make”. ;-) )
3. Then **build and test the resulting CLI application** using “fltk-config --compile abc1234\_video\_store\_gui.cpp” or (better) **using “make gui” with a Makefile.** The Makefile handles adding the “-std=c++11” to the g++ command line.  
     
   You may use “make clean” to start the build from scratch, and “make debug” to create a version compatible with the ddd or gdb debuggers.  
     
   If you wish to use the suggested solution, use the single file solution provided with this assignment. Build and test this CLI application as above to ensure it works with your system.  
     
   **You must start with a solid, working CLI application before making any FLTK changes.**
4. Add the necessary FLTK headers to the top of the single source file. This is the standard FLTK header plus fl\_ask.H. Verify that the program still compiles and runs. It should behave identically to the pre-modification version.
5. Now, make the necessary change for *just one menu* to work as a dialog box. Remember – **baby steps!** (Hint: If you followed the Model – View – Controller pattern, then **the ONLY classes you should need to modify are Controller and View.** If using the provided code as a starting you, then the ONLY class you should need to modify is Controller.)  
     
   Note in the FLTK documentation (e.g., <http://www.fltk.org/doc-1.3/group__group__comdlg.html#ga4daf4c0bc85342a5794b9c2ad9e20b99>) that FLTK only works with const char\* text, not C++ string objects. The way to convert between the two is with the C++ string method c\_str(), which returns a const char\* - see <http://www.cplusplus.com/reference/string/string/c_str/>.  
     
   So, if you have two C++ strings, string\_one and string\_two, and you want to pass them to fl\_input, you would do something like fl\_input(string\_one.c\_str(), string\_two.c\_str()).  
     
   Also note that fl\_input returns a const char\*, so to convert that to an int, you would use atoi – see <http://www.cplusplus.com/reference/cstdlib/atoi/?kw=atoi> – and to a string using a string constructor (one of them accepts a const char\* as a parameter, look it up).  
     
   Build and interactively test your modification. The debugger is your friend. Don't proceed until it works well, and you understand *why* it works well.
6. Now, successively make the necessary changes for each remaining console prompt or report, and test carefully. Remember – **baby steps!** Once every console prompt and report has been converted, you'll have a really simplistic, but working, GUI!

# Grading

* **Full Credit** – For the second sprint, update the Product Backlog tab of the Scrum spreadsheet with the new features (you can **manually merge** rows from the included Scrum\_P5 spreadsheet – do not use it directly!). Then, duplicate the Sprint\_01\_Backlog tab and rename it Sprint\_02\_Backlog, **leaving Sprint\_01\_Backlog unmodified** (this will be kept permanently as a record of the first sprint). Then update cells B1:B3 with the updated info. Delete all of the task rows completed in sprint #1, then add rows for the tasks needed to complete the new features for sprint #2. Use this tab to track your progress during sprint #2.  
    
  You will deliver your .h and .cpp class implementations, or a single file named abc1234\_video\_store\_gui.cpp, along with a Makefile that is competent to rebuild only files that have been modified, if applicable. Also include screenshots demonstrating each of the dialogs as images in PNG format, and your updated Scrum spreadsheet for the second sprint, including the Scrum results from your first sprint.
* **Bonus** – If you modify the GUI to display titles and custom icons on the dialog boxes instead of the system defaults.  
    
  You will deliver your updated class implementations or abc1234\_video\_store\_gui.cpp, a Makefile, screenshots of each dialog in PNG format, and your updated Scrum spreadsheet for the second sprint showing the additional tasks. This must be in a separate folder
* **Bonus** 2– If you replace the series of standard dialogs used for collecting data for a new Video Storewith a single, custom dialog – an fl\_window instance, with labeled Fl\_Input (for title, leading actor, director, copyright, genre, media type, age rating, and id) to accept the media data, and OK and Cancel buttons to accept the data or cancel the operation, respectively.

You will deliver your updated class implementations or abc1234\_video\_store\_gui.cpp, a Makefile, a screenshot of the custom dialog in PNG format **with a media's data entered and ready to submit**, and your updated Scrum spreadsheet for the second sprint showing the additional tasks. This must be in a separate folder.