Project 2 - `1730ed` - The 1730 Text Editor

Pair Programming Project

Team Application: `https://goo.gl/forms/YpUWA858zKmj5V7i2`

CSCI 1730 – Spring 2017

DUE MON 2016-03-27 @ 11:55 PM

1 Problem / Exercise

Your goal is to implement a basic text editor, `1730ed`, in C++ using system calls (APUE Ch. 3) for low-level file I/O and the `ncurses` library for the Text User Interface (TUI). This will require you to lookup things related to `ncurses` and apply your knowledge of object oriented programming.

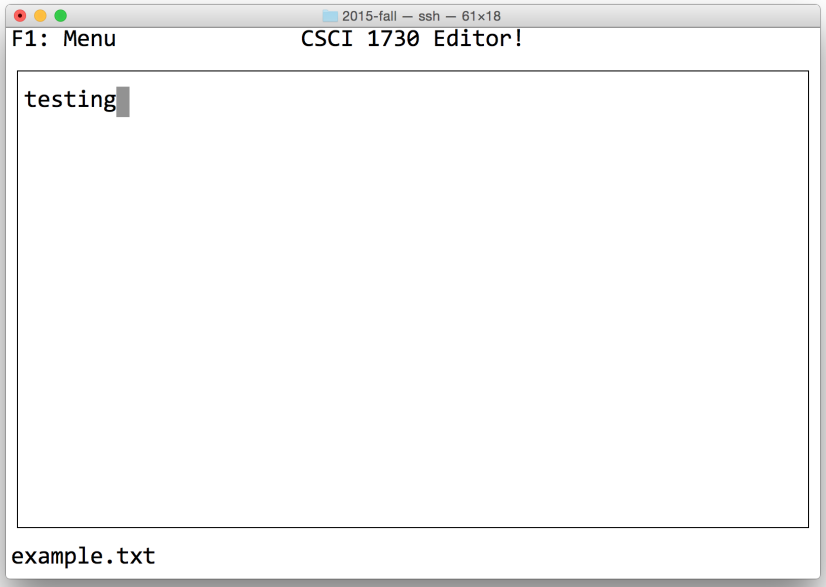
Part of software development is being given a goal but not necessarily being given instruction on all of the details needed to accomplish that goal. For example, even though the `ncurses` library has not been covered in class, in order to complete this project you are going to need to lookup how to create TUIs using `ncurses`. To help make things a little easier, you and your partner may pick and choose the best parts of each other's labs 5 and 7 to serve as a foundation for this project. Furthermore, since `ncurses` is a C library (and not a C++ library), you are going to need to take special care that you are still using C++ best practices and exercising concepts related to object oriented programming. For example, you may wish to create multiple classes in order to better organize your code base.

By submitting this project, you agree to the Academic Honesty policy as outlined in the course syllabus.

2 Functional Requirements (80 points)

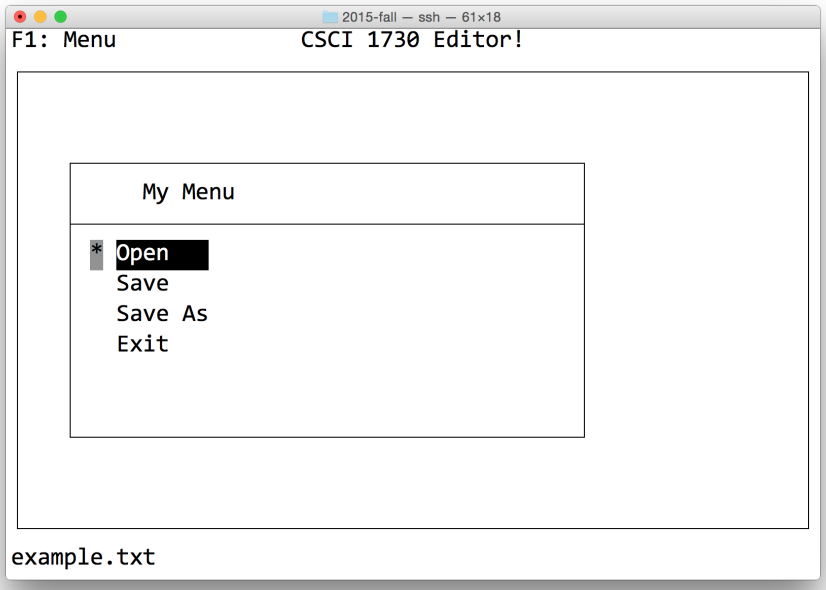
Your submission needs to satisfy the following functional requirements:

- **Editor TUI (40 points):** In addition to the editable text area, the main editor window needs to include text for the menu, the title, and the name of the file currently open in the editor. Here is an example of what the user interface might look like:



- **F1 Menu (20 points):** If the user presses the `F1` key (or `fn-F1` on a Mac), then your editor needs to create a window somewhere in the terminal screen that allows the user to select from the following options:
  - *Open:* This option should prompt the user to enter in a filename. After the user presses return/enter, your editor should attempt to open the file for editing. If an unsaved file is open in the editor when the user chooses this option, then your editor needs to ask the user whether or not they want to save their changes before opening the other file.
  - *Save:* This option should attempt to save the file currently open in the editor. The mode of the file should not be changed by your editor.
  - *Save As:* This option should prompt the user to enter in a new filename and attempt to save the file currently open in the editor to that new filename. If the file already exists, then your editor should ask the user if they want to overwrite the existing file.
  - *Exit:* This option should exit your editor. If an unsaved file is open in the editor when the user chooses this option, then your editor needs to ask the user whether or not they want to save their changes before exiting.

  Feel free to add and implement other options, if you wish. Here is an example of what the menu might look like:



- **Display Errors (20 points):** If a system call issues an error, then your editor needs to create a window in the center of the terminal screen that displays that error. Whenever it makes sense to do so, your editor should prompt the user in an attempt to fix the problem. For example, if the open system call issues an error about a file not existing, then you might prompt the user to re-enter the filename.

3 Extra Credit Requirements (10 points)

You may gain some extra credit points if your submission includes the following:

- **Padded Line Numbers (5 points):** Your editor should include line numbers on the left side of the TUI. These should be padded so that the text from the file that is open does not directly touch the numbers. If the user scrolls up or down, the line numbers should be adjusted accordingly.
- **Add Multiple Colors to your TUI (5 points):** Make your editor colorful for users who are using a terminal emulator that supports colors. Try not to use colors that unfriendly to users who are color blind or who suffer from epilepsy.

4 Nonfunctional Requirements (20 points)

Your submission needs to satisfy the following functional requirements:

- **Directory Setup:** Make sure that all of your files are in a directory called `LastNameA-LastNameB-p2`, where `LastNameA` and `LastNameB` are replaced with the actual last names of you and your pair programming partner.
- **Libraries:** You are allowed to use any of the C or C++ standard libraries. The only third-party libraries you are allowed to use are `ncurses-6.0` and any libraries that come bundled with `ncurses-6.0`. Failure to adhere to this non-functional requirement will result in an automatic 50 point deduction.
- **Documentation (5 points):** All classes, structs, and functions must be documented using Javadoc (or Doxygen) style comments. Use inline documentation, as needed, to explain ambiguous or tricky parts of your code.
- `Makefile` **File (5 points):** You need to include a `Makefile`. Your `Makefile` needs to compile and link separately. That is, make sure that your `Makefile` is setup so that your `.cpp` files each compile to individual `.o` files. This is very important. The resulting executable should be called `1730ed`.
- **Standards & Flags (5 points):** Make sure that when you compile, you pass the following options to `g++` in addition to the `-c` option:

  ```
  -Wall -std=c++14 -g -O0 -pedantic-errors
  ```

  Other compiler/linker options may be needed in addition to the ones mentioned above. The expectation is that the grader should be able to type in the following to clean, compile, link, and run your submission:

  ```
  $ make clean
  $ make
  $ ./1730ed somefile
  ```

- `README` **File (5 points):** Make sure to include a `README` file that includes the following information presented in a reasonably formatted way:
  - Your Name and 810/811#
  - Instructions on how to compile and run your program.

  Make sure that each line in your `README` file does not exceed 80 characters. Do not assume line-wrapping. Please manually insert a line break if a line exceeds 80 characters.
- **Compiler Warnings:** Since you should be compiling with both the `-Wall` and `-pedantic-errors` options, your code is expected to compile without `g++` issuing any warnings. Failure to adhere to this non-functional requirement will result in an automatic 5 point deduction.
- **Memory Leaks:** Since this project may make use of dynamic memory allocation, you are expected to ensure that your project implementation does not result in any memory leaks. We will test for memory leaks using the `valgrind` utility. Failure to adhere to this non-functional requirement will result in an automatic 5 point deduction.

5 Submission

Make sure your work is on `nike.cs.uga.edu` in a directory called `LastNameA-LastNameB-p2`. From within the parent directory, execute the following command:

```
$ submit LastNameA-LastNameB-p2 cs1730a
```

It is also a good idea to email a copy to yourself. To do this, simply execute the following command, replacing the email address with your email address:

```
$ tar zcvf LastNameA-LastNameB-p2.tar.gz LastNameA-LastNameB-p2
$ mutt -s "p1" -a LastNameA-LastNameB-p2.tar.gz -- your@email.com < /dev/null
```

6 Questions

If you have any questions, please post them on Piazza.