Project 1

cs1730-editor

This repository contains the skeleton code for the Editor project assigned to the students in the Fall 2015 CSCI 1730 class at the University of Georgia.

Please read the entirety of this file before beginning your project.

Updates

- Slightly changed the due date for this project under certain circumstances.
- Added another extra credit opportunity.

Due Date

This project is due on Friday 2015-10-16 @ 11:55 PM. However, if you attended lecture on TUE 2015-10-13 or signed the board during your breakout lab on WED 2015-10-14, then this project is due on Monday 2015-10-19 @ 11:55 PM.

Academic Honesty

You implicitly agree to Academic Honesty policy as outlined in the course syllabus and course website.

In accordance with the notice above, I must caution you **not** to fork this repository on GitHub if you have an account. Doing so will more than likely make your copy of the project publicly visible. Please follow the instructions contained in the Resources section below in order to do your development on `nike`.

Project Description

Your goal is to implement a basic text editor in C++ using system calls 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

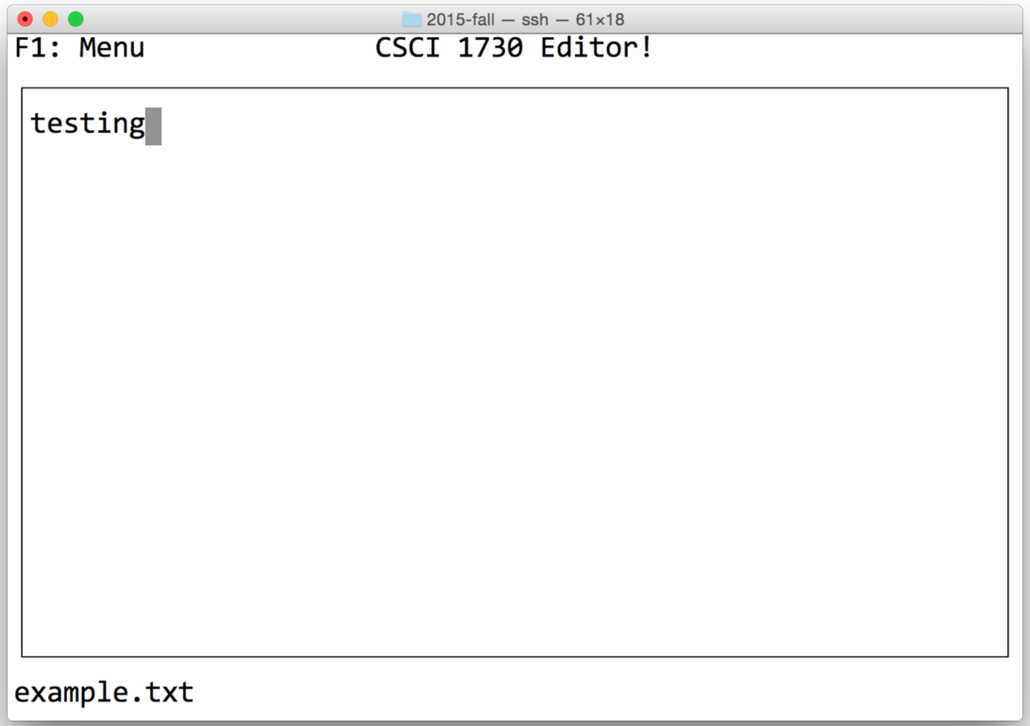
Interface (TUI). This will require you to lookup things related to `ncurses` and apply your knowledge of object oriented programming.

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 hasn't been covered in class, in order to complete this project you are going to need to lookup how to create TUIs using `ncurses`. Furthermore, since `ncurses` is a C library (and not a C++ library), you're 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.

Functional Requirements (80 points)

Your submission needs to satisfy the following functional requirements:

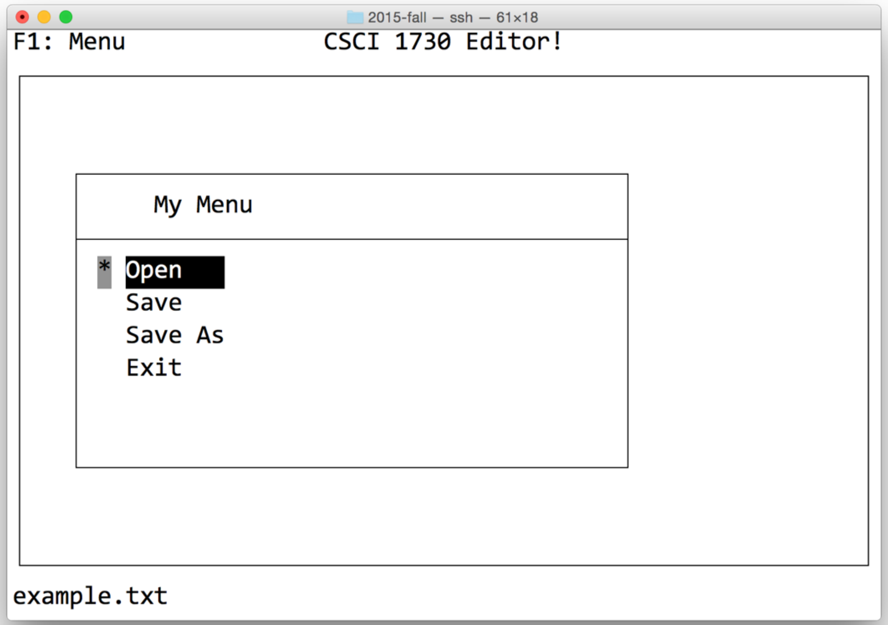
- **(40 points) Editor TUI:** The main editor window needs to include text for the menu, the title, and the name of the file currently open in the editor. Here is an example of what the



```
F1: Menu                 CSCI 1730 Editor!

 testing




example.txt
```

  user interface might look like:
- **(20 points) F1 Menu:** If the user presses the `F1` key (or `fn-F1` on a Mac), then your editor needs to create a window in the

- **(20 points) F1 Menu:** If the user presses the `F1` key (or `fn-F1` on a Mac), then your editor needs to create a window in the center of 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 befire exiting.
- Feel free to add and implement other options, if you wish. Here is an example of what the menu might look like:



```
2015-fall — ssh — 61×18
F1: Menu                   CSCI 1730 Editor!

        My Menu

      * Open
        Save
        Save As
        Exit

example.txt
```

- **(20 points) Display Errors:** If a system call issues an error, then you your edtor 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 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.

Non-Functional Requirements (20 points)

Your submission needs to satisfy the following non-functional requirements:

- **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.
- **(5 points) Documentation:** Your source code should be documented using comments. Inline comments may be used inside of functions. Functions themselves should be documented using Javadoc-style comments. You don't need to go overboard when commenting your code. Simply include comments so that someone else could potentially figure out what is going on with your code.
- **(5 points) Compiling & Linking:** You need to make sure that your submission compiles each `.cpp` file into its own `.o` file. Furthermore, your `.o` files need to link into an executable called `editor`. This can be streamlined by a properly implemented `Makefile`. Furthermore, you need to make sure that you are passing the following options (in addition to any other options you may need) to `g++` when compiling and linking:`-Wall -std=c++11 -pedantic-errors`
- The only exception to this, is that you may also wish to use C++14 features. If that is the case, then you can change the `-std=c++11` option to `-std=c++14`.
- **(5 points) Makefile:** Your `Makefile` should include an `all` target and a `clean` target in addition to any other targets it might need. Furthermore, your `Makefile` should be written in such a way that is satisfies the "Compiling & Linking" requirement mentioned above.
- **(5 points) Compiler Warnings & Memory Leaks:** Your submission should not issue any compiler warnings or errors when compiled and linked. Furthermore, when your submission is run, the `valgrind` utility should indicate that

submission is run, the `valgrind` utility should indicate that there are NO memory leaks (in the first three leak reports).

NOTE: 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
$ ./editor somefile
```

Extra Credit Opportunity (10 points)

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

- **(10 points) Padded Line Numbers:** 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.
- **(5 points) Add Multiple Colors to your TUI:** Make your editor colorful for users that using a terminal emulator that supports colors.

Skeleton Code

The files for this project are hosted Github using `git`. They can be retrieved by cloning the repository found at `git://github.com/uga-csci-1730-cotterell/cs1730-editor.git`. For example, you can issue the following command to clone the repository:

```
$ git clone git://github.com/uga-csci-1730-cotterell/cs1730-editor.git LastName-FirstName-p1
```

If the above command issues you any errors or warnings, then you might try the following command instead:

```
$ git clone https://github.com/uga-csci-1730-cotterell/cs1730-editor.git LastName-FirstName-p1
```

NOTE: There really isn't any skeleton code for this project. Creating your files is completely up to you so long as you fullfill the project requirements. The above command simply creates a directory for this project that includes this `README.md` file as well as a stub for your `Makefile`.

NOTE: As always, I suggest developing directly on `nike.cs.uga.edu`because this is where your project will be run and tested. Since `git` is already installed on `nike`, you can clone the project directly into your `nike` home directory using the and tested. Since `git` is already installed on `nike`, you can clone the project directly into your `nike` home directory using the command provided above.

NOTE: If any changes are made to the project description or skeleton code, they will be announced in class. In order to incorporate such changes into your code, you need only do a `git pull`.

NOTE: Also, since `git` is a decentralized version control system, you will have your own local copy of the repository. This means that you can log your changes using commits and even revert to a previous revision if necessary.

Submission Instructions

You will be submitting your project via `nike`. Make sure your work is on `nike.cs.uga.edu` in a directory called `LastName-FirstName-p1`, and, from within the parent directory, execute the following command:

```
$ submit LastName-FirstName-p1 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 LastName-FirstName-p1.tar.gz LastName-
FirstName-p1
$ mutt -s "[cs1730] p1" -a LastName-FirstName-p1.tar.gz
-- your@email.com < /dev/null
```

Questions

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