Erik Kriz
CSE 3150
Project Part 5 Code Write-Up

Report of Object orientation: Throughout this project, I have tried to follow the object orientation shown in the class slides. I will list all of the classes I have created and describe what their job is and some key functions of those classes.

List of Classes:

1. ECTextEditor
    - The ECTextEditor Class is really just meant to be constructed and deconstructed. It both creates the main classes of the editor (View, Controller, Document) and it interconnects each of those objects, giving each a pointer to each of the other ones.

2. ECTextEditorView
    - This class is where most of the work gets done. It takes parameters from the controller, and data form the document, and uses them to create what the user sees. It creates the ECTextViewImp on creation, and is the only object to invoke the ECTextViewImp, essentially it is the more complex ECTextViewImp. This class has many functions: to start it has a function which is called every "frame" of the program which builds the page the user sees. It does this by reading data from the ECTextEditorDocument object. More specifically, it iterates through the text stored in that document, and checks where each page would begin and end, then it displays the page the user wants by adding all related lines to the ECTextViewImp.

    - There is a function which builds the status line. Usually the status line just has my name and debug information (cursor coordinates, page number, document position of cursor), but in find mode and replace mode the status bar changes to show which mode it is in and the find/replace phrase.

    - There is a function for finding where in the document you are by virtue of the X,Y coordinates of the cursor as well as which page you're on; which helps to insert characters into the correct position of the Document's text vector.

    - The rest of the functions are either for manipulating the cursor, or changing the text within the status bar.
3. ECTextEditorController
    - This class inherits from ECObserver, meaning it handles key presses. The only functions within it are ones which set the View/Document to pointers, one which

let's it know if it is in find/replace mode, and Update. Update is called whenever a user presses a button, and is what is inherited from ECObserver. It creates a Ticket class object (which records the code of the pressed button) and sends that ticket to a chain of responsibility structure to be handled.

- This Controller has a vector of objects within it that act as the first node of a chain a responsibility. Depending on what state the machine is in (normal, find, replace), the controller will send each ticket down a different chain.

4. ECTextEditorDocument
    - This class holds a vector of chars, which represents the entire text file. It holds functions to add/remove chars from this vector. Very basic
5. Ticket
    - Objects of this class are created by the Controller class. Each time the user presses a button, an object of this class gets created, and is handled by a chain of responsibility. It holds a few functions to access the View/Controller/Document, as well a function to access the key of the button pressed.
6. KeyEventManager
    - This class is meant to be the first stop of the chain of responsibility. The Controller class contains a vector of these objects, depending on the state of the machine, a ticket will be handled by a different manager/chain.
7. KeyEventHandler
    a. Many handler sub-classes
    - KeyEventHandler is a base class which all handlers inherit from. On its own it does nothing, but it defines destructors/handling for all of the classes which inherit from it.
    - All "Handler" classes inherit from this class, each handler class becomes an object within a chain of responsibility, with a function for keys it can handle and a function which handles those keys.
8. ECCommand
    a. Commands Sub-classes
    - Similar to what we did in class, this is just a base class which all "Command" classes inherit from. Basically, if there was an action which I wanted to be "Undo-able", I created a Command class which defined how to do so.
9. ECCommandHistory
    - Unchanged to what we did in a previous code. This holds a vector of commands and defines how they should Execute/Undo/Redo.

Known issues:

1. The highlighting function of find/replace does not work. The code is there to set the color, but nothing happens. After testing it, it seems that I cannot change the color at all after the first “update” is called. Even if the set color is tied to the update/refresh functions, it only ever works before the user presses their first button of the session.
2. When a user quits out (with CTRL-q), the code seg-faults. This occurs because of an error in the final “delete” functions at the end of my ECEditorTest.cpp file. I know that the code probably tries to delete the same object twice, which would result in said seg fault, but I haven’t found it. Since all functionality is retained in spite of this fault (including file saving), I fixed other major errors, without having time to fix this.