CSE 3150 — Essentials of C++ — Yufeng Wu — Spring 2020
Project Milestone 1: A Simple Text Editor: Basic Functionalities. Due: 04/10/20.

In this semester, you will practice C++ programming by writing a console application: a vi-like text editor. Before we start, please keep in mind that our objective is learning C++in the object-orientation approach. Text editor itself is not particular exciting to implement. We have all used text editors for various purposes. The point of this project is implementing a non-trivial program following the OO paradigm.

There are the main components for the first milestone of project:

1. Get to know basics about console based text editor.
2. Get familiar with ECTextViewImp: the provided text view implementation.
3. Implement a list of basic features for the project.

1 Basics of console based editor

Many of you have at least tried vi or vim. If you haven't, you should try to experience vi, which is almost universal in every console system. In particular, you should see what are the basic components needed in a console-based text editor; what the UI looks like?

2 ECTextViewImp: a C++ implementation of text view for console application

The first challenge for writing a vi-like text editor using C/C++ is that there is no user interface library provided in C/C++ standard library. C/C++ does provide basic support for basic things such as read input and output to the console. However, the standard library is just too limited in providing user interface (UI). One key needed feature is the ability to control the cursor. Standard C/C++ library has no notion of cursor. You can write to console and you can change to a new line easily with STL; however, you cannot easily move the cursor to different positions using only STL. To handle this difficulty, we could use some UI libraries such as Qt. However, these libraries, while powerful, are usually complex. This can let you spend too much learning a library which is not the main objective of this course.

In this project, I have decided to take a lightweight approach. I will provide you an implementation of basic console based text view, called ECTextViewImp. This ECTextViewImp has only a single class, which supports a limited set of functions. I wrote the class myself, which is essentially a C++ port from a simple editor code called Kilo. I have tested this class myself, and am confident it will work. However, please **note** that I didn't have a lot of error checking in the code. Thus, your code can easily crash if you invoke the library in the wrong way. So you need to carefully debug your code when using ECTextViewImp. That being said, ECTextViewImp is very simple (only have several hundreds lines of C++ code), and it should be cross-platform and has no dependencies. In contrast, if you use heavy weight UI libraries such as Qt, even installation can go wrong for some platforms.

Here are the basic features supported by ECTextViewImp. The basic design of ECTextViewImp is event driven. That is, after initialization, ECTextViewImp would enter a forever loop. You can change the content of view. If something occurs (in particular, a key is pressed), a notification is sent to the client. I encourage to read the source code of ECTextViewImp before using it.

1. Basic view management functions. These include: Init (must be called before using the class ECTextViewImp object), Show (this will render the view and enter a forever-loop), Refresh (refresh the current view), Quit (invoke to exit from the view).

2. View management. A view essentially contains a set of rows. You can clear the rows in the view by calling InitRows. This can be useful when say you want to display a new page of the document. You can add a row by calling AddRow.

3. Get/Set of the states of the view. These include getting/setting information about cursor (position) and getting dimension of the view.

4. Status rows. You can choose to add one or more status rows at the bottom of the view. For each status row, you can place two pieces of texts: one at the left end of the row, and one aligned to the right of the row. You can choose whether to use black background (i.e. darken the background).

5. Key stroke. ECTextViewImp implements the "Observer" design pattern. The client (which is a subclass of ECObserver) that is interested in getting key stroke of the text view must register with ECTextViewImp by using the Observer interface function, called Attach. Then when a key stroke occurs, the "Notify" function of the Observer will invoke "Update" function. Then, the client can invoke "GetPressedKey" of ECTextViewImp to find out which key is pressed. ECTextViewImp defines a list of key codes.

Note that ECTextViewImp is still under development. I may update the code and add new functions to the interface. But I will try not to change the existing (public) interface functions. I will provide a simple starter program to demonstrate the functionalities of ECTextViewImp.

Caution

The current ECTextViewImp implementation doesn't perform extensive error handling. That is, it will be easy to break the code if you are not careful. Based on my own programming experience, one frequent mistake a programmer can make while using the ECTextViewImp code is the cursor management. The cursor must be within the valid range. That is, suppose you have three rows, and each row has five characters. Then the cursor can be placed at the first three rows at positions 0 to 5 (cursor position is zero-based). If you are going to have an empty fourth row (by pressing Enter at the end of the third row), then the cursor can also be located at the position 0 of the fourth row. If you place a cursor outside the valid region, the code can easily crash.

3 Features to implement

3.1 Document functions

You should be able to support basic operations of text editors. These include:

1. Application: allow quit by typing ctrl-q.

2. Cursor movement. Initially the cursor should be positioned at the upper left corner of the view. You shouldn't allow a cursor to move past the end of the text. That is, if a line has 10 characters now. You can move the cursor past the end of the line by 1 at most.

3. Text insertion. The user should be able to add a new line (by pressing enter). You should break the current line into two lines when the user pressed enter. If you press "enter" at the end of a row, you will create a new (empty) row after the current row. The user can also add new text by typing some text at the cursor.

4. Text removal. Use backspace to remove the text in a line. If you press backspace at the first column (position 0) and there is some row before it, then you will merge the two rows.

5. Undo/redo: all the editing actions should be undoable/redoable. Here, ctrl-z for undo and ctrl-y is for redo.

For simplicity, you may assume the text to edit is small so that you won't need to have a row that is longer than the number of columns of the view. Also you won't need to have more rows than the number of rows in the view. The issue of larger text will be handled in the next milestone.

4 Design and implementation

I strongly suggest you to apply OO principles and design patterns in your code. Working on a real project is the best way to learn how to do OO. Try to make your code flexible and accommodate for changes. In particular, I recommend you to apply the following design patterns:

1. **Model-view-controller**: use MVC as the main structure of your document class.
2. **Commands**: use command pattern to support undo/redo.
3. **Chain of responsibility**: use chain of responsibility to handle the key stroke events. This way, your key stroke event handling will be easier to maintain to handle more keys in the future.
4. **Observer**: use Observer pattern to receive notification of key stroke. This is already implemented by ECTextViewImp. You need to register as an observer to the text view in ECTextViewImp to receive key stroke event notification.

5 What to submit?

At the end of this assignment, each student must submit the following:

1. A short report on the key features you have implemented. You must also provide a simple user manual, which teaches the user how to use your text editor. This should be submitted a single PDF file.
2. Source code. You must include a Makefile. Your code should be able to build by typing "make" at the main source code folder. You will lose a significant portion of grade if your code cannot compile this way. Note: we may or may not look at your code for grading.

Note: you are *not* allowed to change the provided ECTextViewImp files. I may update the ECTextViewImp code and any code change done by you on ECTextViewImp may be in conflict with the new code. We will definitely examine your code in more depth at the end of the semester. However, due to the large enrollment of the class, we may not be able to check your code at each milestone.