CS246E—Assignment 5, Project (Fall 2023)

B. Lushman

Due Dates: See project guidelines

DO NOT EVER SUBMIT TO MARMOSET WITHOUT COMPILING AND TESTING FIRST. If your final submission doesn't compile, or otherwise doesn't work, you will have nothing to show during your demo. Resist the temptation to make last-minute changes. They probably aren't worth it.

This project is intended to be doable in the alloted time. Because the breadth of students' ablilities is quite wide, exactly what constitutes the amount of work that a student can do is difficult to nail down. Some students will finish quickly; others won't finish at all. We will attempt to grade this assignment in a way that addresses both ends of the spectrum. You should be able to pass the assignment with only a modest portion of your program working. Then, if you want a higher mark, it will take more than a proportionally higher effort to achieve, the higher you go. A perfect score will require a complete implementation. If you finish the entire program early, you can add extra features for a few extra marks.

Above all, MAKE SURE YOUR SUBMITTED PROGRAM RUNS. The markers do not have time to examine source code and give partial correctness marks for non-working programs. So, no matter what, even if your program doesn't work properly, make sure it at least does something.

Note: This assignment requires you to manipulate the terminal screen in a non-sequential way. Before you start programming, you will need to learn to use the Ncurses library (or equivalent functionality). You may also want to spend some time thinking about how you might make this library (which was written for C) more C++-friendly (e.g., build some abstractions around the core functionality).

The vm editor

In this project, you and a partner will work together to produce a vim-like text editor. If you are not familiar with the vim editor, it would be worthwhile to spend some time working with it before your begin your project. You might even consider using vim to write vm! Once you get far enough into the project, you might consider using vm to write vm!

The vm text editor has functionality very similar to vim: there is an insert mode for entering text and a command mode for manipulating text. Pressing Escape switches from insert mode to command mode. The editor starts in command mode. In command mode, most keys on the keyboard carry out some useful function. Your submission is to support at least the following commands:

```
a b cc c[any motion] dd d[any motion] f h i j k l n o p q r s u w x yy y[any motion]
A F I J N O P R S X ^ $ 0 . ; / ? % @
```

`^b ^d ^f ^g ^u`

The "undo" command, `u`, must support an unbounded number of undos. Note that not all commands are considered undoable (for example, the cursor movement commands, `h j k l` are not undone when you press `u`).

Many commands can be prefixed with a numeric "multiplier" that executes the command a given number times (for example, `3j` moves the cursor down three lines; `4dw` deletes four words). Your solution will support multipliers for those commands that can be multiplied.

Some commands are accessed by pressing a colon (`:`). These commands show up visually at the bottom of the window, and are not executed until the user presses Enter. You are to support the following colon commands:

`:w :q :wq :q! :r :0 :$ :line-number`

The last of these, a colon followed by a number, brings the cursor to that line number in the document.

The `/` and `?` commands (mentioned above) search the document for a match to a given string. These are also displayed at the bottom of the screen, and are not executed until the user presses Enter. Note that although vim permits regular expression searches, you only have to support searches for exact strings.

Your program is not required to maintain persistent state (e.g., most recent search) between sessions.

Display

The display of the text file should occupy the entire terminal window (no matter how large it is; reasonable limitations may be placed on the minimum size of the window). The bottom line of the display functions as a "status bar". On the left, it indicates insert mode, or the the name of the file after a load or save. On the right, it displays row and column information.

If the document is not large enough to fill the screen, the remaining lines contain a single `~`. Lines that are too long for the width of the window must wrap.

You are expected to support ONE of the following two features; if you do both, you may claim the other as an enhancement for three marks.

Syntax highlighting

If the file being loaded ends with `.h` or `.cc`, your editor must assume that the file contains C++ code, and must colour the text, such that a distinct colour is used for each of the following:

1. keywords (how will you handle contextual keywords?)
2. numeric literals
3. string literals
4. identifiers
5. comments

6. preprocessor directives
7. everything else

Your syntax highlighter must also indicate, via a distinctive colouring, any mismatched braces, brackets, or parentheses. For questions of what constitutes a mismatch, use vim as your guide.

Macros

You will notice above that you are asked to support the `q` and `@` commands, that permit a user to record and play back macros. Note that not all commands are considered recordable (e.g., `^g`).

Question: Although this project does not require you to support having more than one file open at once, and to be able to switch back and forth between them, what would it take to support this feature? If you had to support it, how would this requirement impact your design?

Question: If a document's write permission bit is not set, a program cannot modify it. If you open a read-only file in vim, we could imagine two options: either edits to the file are not allowed, or they are allowed (with a warning), but saves are not allowed unless you save with a new filename. What would it take to support either or both of these behaviours?

Interpretation of Commands

For a given command issued in a given situation, if you are unsure what the behaviour of your editor should be, and the situation is not addressed in this document, use `vim` as your guide. If for some reason, it is impossible or infeasible to do what `vim` does, document your choice in your final submission.

If the user presses a key combination that is not recognized by your editor, it should not crash; rather, it should ignore the keypress. If the user presses the first key of a multi-key command (e.g., the user presses `c` or `d`), then Escape should abort the command.

Architecture

Your implementation of this project must employ the MVC design architecture (loosely defined). Information on MVC will be given in a tutorial. As part of this requirement, you must separate the classes that manipulate the core editor data structures from those that communicate with the user.

Grading

Your project will be graded as follows:

Correctness and Completeness	60%	Does it work? Does it implement all of the requirements?
Documentation	20%	Plan of attack; final design document.
Design	20%	UML; good use of separate compilation, good object-oriented practice; is it well-stuctured, or is it one giant function?

Even if your program doesn't work at all, you can still earn a lot of marks through good documentation and design, (in the latter case, there needs to be enough code present to make a reasonable assessment).

If Things Go Wrong

If you run into trouble and find yourself unable to complete the entire assignment, please do your best to submit something that works, even if it doesn't solve the entire assignment. For example:

- can't handle command multipliers
- no (or limited) macros
- no (or limited) syntax highlighting

You will get a higher mark for fully implementing some of the requirements than for a program that attempts all of the requirements, but doesn't run.

A well-documented, well-designed program that has all of the deficiencies listed above, but still runs, can still potentially earn a passing grade.

Plan for the Worst

Even the best programmers have bad days, and the simplest pointer error can take hours to track down. So be sure to have a plan of attack in place that maximizes your chances of always at least having a working program to submit. Prioritize your goals to maximize what you can demonstrate at any given time. We suggest: start with getting your program to display a file correctly. Then add basic cursor movement commands. Then implement insert mode. Then continue to add commands, one-by-one. At the same time, you should take all of the commands into consideration when you are designing your program, because some of them could affect your design considerably. Take the time to work on a set of test documents at the same time as you are writing your project. Although we are not asking you to submit a test suite, having one on hand will speed up the process of verifying your implementation.

You will be asked to submit a plan, with projected completion dates and divided responsibilities, as part of your documentation for Due Date 1.

If Things Go Well

If you complete the entire project, you can earn up to 10% extra credit for implementing extra features. These should be outlined in your design document, and markers will judge the value of your extra features.

To earn significant credit, enhancements must be algorithmically difficult, or solve an interesting problem in object-oriented design. Trivial enhancements will not earn a significant fraction of the available marks.

Submission Instructions

See **project_guidelines.pdf** for instructions about what should be included in your plan of attack and final design document.