Select the right page in Gradescope or we will not grade the question!

Homework 4
Editors

Due: Wednesday, October 10th, 11:59PM (Hard Deadline)

Submission Instructions

Submit this assignment on Gradescope. You may find the free online tool PDFescape helpful to edit and fill out this PDF. You may also print, handwrite, and scan this assignment.

Optional Readings

Interrupting Developers and Makers vs Managers

In lecture we spent a little time talking about the mental model of programmers. As computer science grows, there is a growing body of research and wisdom in how to maximize the efficacy of individual programmers. One of the key revelations is that programming is a creative process. It requires time to "page in" what you are working on. As a result, even small interruptions (read: text messages) can be far more costly than you may realize.

http://thetomorrowlab.com/2015/01/why-developers-hate-being-interrupted/
Why developers hate being interrupted, by Derek Johnson at The Tomorrow Lab.

http://www.paulgraham.com/makersschedule.html
Maker's Schedule, Manager's Schedule, by Paul Graham, co-founder of Y Combinator.

Once you are in a job environment and meetings become a regular thing, one of the best tricks I learned is to schedule a few 4-hour meetings with yourself throughout the week. It both prevents others from interrupting you (your calendar shows you as busy) and lets you mentally prepare for a good, reliable work session. (A similar alternative, some find pomodoro helpful)

Software Engineering in the Real World

http://blogs.msdn.com/b/peterhal/archive/2006/01/04/509302.aspx
What Do Programmers Really Do Anyway?, by Peter Hallam, then-Microsoft Developer

> *Why is 5 times more time spent modifying code than writing new code? The answer is that new code becomes old code almost instantly. Write some new code. Go for coffee. All of sudden you've got old code.*

This post gives a nice perspective on what enterprise software engineering is really like. School projects are misleading. Rarely in life will you be faced with a spec and a blank text file. Far more often you are building on or improving something that came before you and will exist long after you.

http://www.joelonsoftware.com/articles/fog0000000069.html
Things You Should Never Do, Part I, by Joel Spolsky

> *It's harder to read code than to write it.*

Building on the previous, rarely in life are you handed a spec and a blank text file, often in life you are handed a spec and a pile of (seemingly) spaghetti code that does at least some of it. This article discusses the hidden value of the built-up institutional knowledge and the high value of working code.

When was the last time you saw a hunt-and-peck pianist? -Jeff Atwood, co-founder of Stack Overflow and Discourse.
Some extras on the value of being a good typist and ditching the mouse for the keyboard.

1 Trying out some of the fancy features

People often complain about how difficult the simple things are using text-based text editors. Indeed, Notepad (or TextEdit) make changing some text and saving a file very easy. In this question we explore some of the things that Notepad (and nano, and gedit, and Microsoft Word...) cannot do.

I encourage you to play around with things as you work through this question. The intent is to expose you to features you may not have been aware of along with some context for why they are useful.

There are many things to try here. **For credit on this question, you only need to choose any 5.** That said, I encourage you to try all of these.

For this question, you may choose `vim` or `Emacs`, whichever you are more comfortable with.

To start, grab copies of two files full of code:

```
# Examples from http://web.mst.edu/~price/cs53/code_example.html
wget http://web.mst.edu/~price/cs53/fs11/workDecider.cpp
wget http://web.mst.edu/~price/cs53/KatsBadcode.cpp
```

---

1. Sometimes you will come across some ugly code. Your editor can help make it better. Open `KatsBadcode.cpp`. Among other issues, the really bad tabbing makes this hard to read.
   Describe how to automatically fix the indentation for the whole file.

   Now imagine if you are editing code and you determine you can remove an if block, for example removing the `if` on line 28 of `KatsBadcode` and running its body unconditionally. You need to fix the indentation level of all 20 lines of the body.
   Describe how to change the indentation level of a block of code.

2. Editing one file is nice, but often it's really useful to compare files side-by-side (source and headers, spec and implementation).
   Describe the command sequence to create a window split (within your text editor) side-by-side with your current window, switch to it, and then open a file in that window.

   Try playing around with these two views, copy/paste code between them, bind them so they scroll together, resize them, add more splits.

3. Editors also have pretty nice integration with compilation tools. With `KatsBadcode.cpp` open, try typing `:make KatsBadcode` in `vim` or `M-x compile<enter>make KatsBadcode` in `Emacs`.[1]

   First cool thing that happened: Yes, you can run `make` *without* any Makefile anywhere. We'll cover how that happened during build-system week.

   Building `KatsBadcode.cpp` will fail with several errors.
   Describe how to navigate between compilation errors automatically.

4. While C-style languages support `/* block comments */`, others such as Python have no block comments and require you to put a `#` at the beginning of every line.
   Describe how to efficiently comment out a large block of Python code.

5. Many times you have a repetitive task that you need to do say 10 or 20 times. It's too short to justify writing a script or anything fancy, but it's annoying to type the same thing over and over again. As example, reformatting the staff list to a nice JSON object:

```
Darden,Marcus                       staff = [
Hariharan,Amrit                             {"first": "Marcus", "last": "Darden"},
Triesenberg,Stevie                          {"first": "Amrit", "last": "Hariharan"},
Khan,Samiur                                 {"first": "Stevie", "last": "Triesenberg"},
Liu,Christina                               {"first": "Samiur", "last": "Khan"},
Khubchandani,Tarun                          {"first": "Christina", "last": "Liu"},
                                            {"first": "Tarun", "last": "Khubchandani"},
                                    ]
```

   Editors support the record and replay of *macros*. With macros, you can start recording, puzzle out how to turn one line into the other, and then simply replay it for the rest. As a general rule, do not bother trying to be optimal or efficient, just find anything that works. Try starting with the column on the left and devising a macro to turn it into the column on the right.
   Describe how to record and replay a macro. *(you do not need to include the contents of your macro)*

[1] Descriptions of `Emacs` commands are usually written as `C-c` or `M-x`, which mean "Ctrl+C" or "Meta+x" respectively. Meta is often mapped to the escape and/or alt key, `M-x` means press Escape and then x or hold alt and press x.

6. Some final little things

   (a) Editors understand balanced ()'s and {}'s. **Describe how to jump from one token to its match, such as from the opening { on line 29 of `KatsBadcode` to its partner } on line 58.**

   (b) Sometimes it's useful to run quick little commands. For example, your code needs to read from a file, but you can't remember if it's called `sample_data.txt` or `sampleData.txt`.
   Describe how to run 'ls' directly from within your editor.

Personalising your Editor (Ungraded but recommended)

Similar to how ~/.bashrc configures your shell, a ~/.vimrc or ~/.emacs file[2] can configure how your editor behaves. Here are some excerpts from the staff's configuration files:

```
.vimrc:
set number     " double-quote means comment in vimscript; this turns on line numbers
map ; :        " this line and the next makes ; act like : so that you don't have to
noremap ;; ;   " hit shift all the time, typing ";;" will act as ";" used to

.emacs:
; line numbers - in Emacs, ; means a comment
(global-linum-mode t)
(setq linum-format "%d ")
; Changes all yes/no questions to y/n type
(fset 'yes-or-no-p 'y-or-n-p)
```

Add something useful not listed above to your editor's configuration file.

[2] These files do not exist by default, you have to create them.

2 Remote work

One thing that can make life very convienent is learning how to work on remote machines. CAEN has a large number of machines available for students to use remotely, at `login.engin.umich.edu`. Let's check that out.

`ssh` is the **s**ecure **sh**ell program, and is used to securely log in to remote machines.

Run `ssh login.engin.umich.edu` to log into a CAEN machine. Take a look around, this is the same login environment and home directory as if you had sat down at a physical CAEN machine.

Use `vim` or `emacs` to create a file called `test.txt` on CAEN. Put some text in it, save, and quit.

Now try to run `gedit test.txt`.

What error do you get? Why?

Close your ssh session (log out of the terminal, either by typing `exit` or pressing `Ctrl-d` at an empty prompt). Then log in again with the additional `-X` flag to ssh: `ssh -X login.engin.umich.edu`.

Now try running `gedit test.txt` – great! Only, now you cannot do anything else in that terminal.

How can you keep using this terminal without killing the gedit session that is already running?

How should you launch `gedit` so that you can still use the terminal?

(Hint: remember we talked about shell job control a few weeks ago)

Try running a more graphically intensive program, such as the `eclipse` IDE or `matlab`. How does it compare to sitting at a physical CAEN machine? How does it compare to using VNC to log into CAEN? (And how good is your Internet connection right now ;) ?)

You may also find the **s**ecure **co**p**y** program, `scp`, useful for moving files between your machine and the remote machine. Assuming you have the file 'test.txt' on your local machine,

```
scp test.txt login.engin.umich.edu:
```

will copy the file to CAEN. Note the trailing colon, this is what tells `scp` to copy to a remote machine.

You can also specify a path on one or both machines. Assuming that you have the folder 'inhomeoncaen' on your CAEN account and 'local' in the current directory of your machine,

```
scp login.engin.umich.edu:inhomeoncaen/test.txt local/
```

will the file `test.txt` from CAEN to your machine.

`scp` operates just like `cp`, that is, you have to pass the `-r` flag if you want to copy a directory, such as

```
scp -r project1 login.engin.umich.edu:classes/eecs280/
```

(There's no question on `scp`, just a tool for you to play with)