CS 163 Regular Expressions Lab

Fall 2019

October 32, 2071

1 Introduction

In this lab, we will be using a text editor called `vim` to modify some C files with regular expressions. All the cool kids use `vim`. `vim` (and some other Linux text editors) are able to process regular expressions using some special syntax that you're going to learn in this lab.

In class, we talked about the math notation for regular expressions which involves some Greek letters, superscripts, etc. that aren't typable on keyboards. Text editors use a different syntax with only typable characters. The table below shows some of the notation conversions:

Math Notation	`vim` **Notation**	**Description**
Σ	`.` (period)	Set of inputs that the machine recognizes
R^+	\+	A sequence of **one** or more of the superscripted symbol
R^*	*	A sequence of **zero** or more of the superscripted symbol
R^3	/R \{3}	A sequence of exactly three of the superscripted symbol
R^{2-4}	/R \{2,4}	A sequence of 2, 3, or 4 of the superscripted symbol

`vim` also has some text-file specific input characters

`vim` **Notation**	**Description**
$	End of a line
^	Beginning of a line
\n	New Line
\t	Tab
\< and \>	Word boundaries

2 Simple RegExes in `vim`

Get a Text File First, go to the course webpage and download `hermes.c` from the schedule for today. This is a C source file with almost 1000 lines of

code. I tried to use good coding style when I wrote this program, but there's always room for improvement. We'll use RegExes in `vim` to fix some of the ugliness.

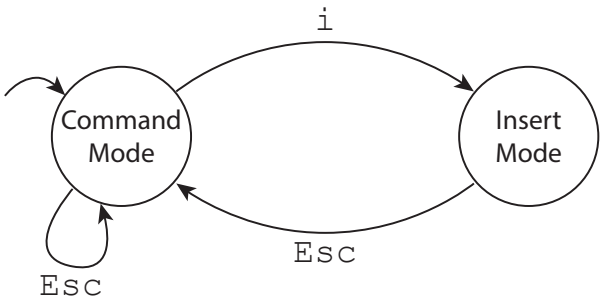
2.1 Intro to `vim`

Open this file with vim. On Mac/Linux:

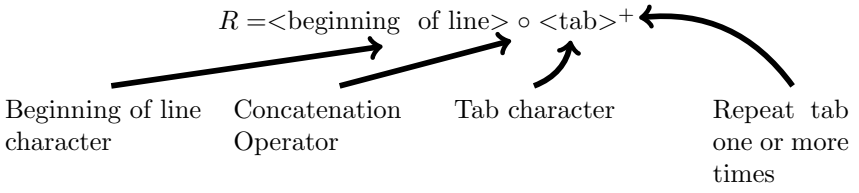
```
vim hermes.c
```

Once you open it, you can scroll around in the file using the arrow keys, but you won't be able to actually type any input text. `vim` has two modes: command mode and insert mode. In command mode, you can type in commands (like RegExes). Commands you type will not be directly written to the text file.

In insert mode, you can type in text that will actually get written to the file. To switch from command mode to insert mode, press the `i` key. When you are in insert mode, you will see `-- INSERT --` on the bottom left corner of your screen. To switch from insert mode back to command mode, press ESC.



Replace Tab Indents with Two Spaces There are a lot of deeply nested blocks of code in this file that are indented many times. For instance, check out line 702, which is indented eight times. In math notation, the regex to search for tabs at the beginning of lines is:



Here, we want the superscript + to apply only to the tab character. Draw a finite state machine that implements this regular expression in the box below. Make sure you include (1) the initial state, (2) intermediate states that parse the text file, (3) state transitions that depend on the input text file, and (4) a final (accepting) state.

Running a RegEx in `vim` To enter this into the text editor, we need to insert the beginning of line character and tab character from the table of `vim`-specific characters above:

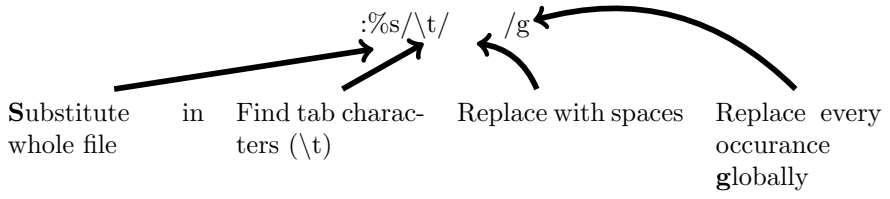
$$/\wedge\backslash t\backslash +$$

To run this regex in `vim`, you need to be in command mode—just press `ESC` a bunch of times to get into command mode. Let's go through this regex character-by-character:

`/`	Regular expressions have to start with a forward slash. This is not part of the regex, it's just a character that tells `vim` that what follows is a regex.
$\wedge$	Beginning of line character. This is the first part of the regex that tells `vim` to match strings starting with a beginning of line.
`\t`	Tab character. Since the tab character is right next to the caret, it's implied that we're searching for the concatenation of the two.
`\+`	This is the way we type in the superscript plus, which specifies that we're searching for sequences of one or more tabs.

Now get `vim` into command mode and type this in and press enter. In command mode, the regex should appear in the bottom left corner of the screen. After you press enter, `vim` should highlight whitespace at the beginning of most lines. If you have problems, get Neil to help.

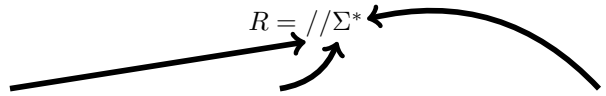
Ok this is great, but we have only searched for tabs at the beginning of lines. Now let's try replacing them with spaces



Type this in and replace all tabs to spaces in your file. The general form for search and replace in `vim` is:

$$:\%s/< regex >/< whatever\ to\ replace\ matches\ with >/g$$

Now suppose we want to remove all comments. We might want to do that in order to obfuscate our source code before making it public.



Two slashes indicate beginning of single line comment	Followed by any element of the input alphabet (typable characters)	repeated zero or more times

Draw a finite state machine that implements this regex in the box below:

To express this regex in `vim`, we will

1. Escape the forward slashes. Literal forward slashes need to be represented as \/
2. Replace the Σ with `.`

The `vim`ified version looks like:

$$/ \backslash/ \backslash/.*$$

Type this in to vim and verify that it highlights all single line comments. It shouldn't replace anything though. In order to replace, we need to:

1. Add `:%s` at the beginning of the regex, which tells vim to **s**ubstitute in the whole file.
2. After the regex, we need to tell `vim` what to substitute for the matched strings

The complete substitution command is:

$$/ \backslash/ \backslash/.*//g$$

This should delete all single-line comments that begin with a //. The problem is, we only deleted the comments. Most of the comments we deleted had spaces before the //. This will leave us with spaces at the end of the line. To search for spaces at the end of the line:

/ \+$

Type this in to highlight all spaces at the end of lines. Now: how would you use the replace feature to eliminate spaces at the end of lines?

Removing Multiline Comments Now that we've removed the single line comments that begin with //, write a regular expression that would remove multiline comments that begin with /* and end with */.

Draw a finite state machine that implements this regex: