Lab 9
Booleans and helpers

24 October 2019

More functions this week, this time with a focus on writing and using Boolean functions and expressions.

First you'll be writing a function with three parameters that determines whether any two of the given numbers could be added to get the third.

1. Before you go further in the design process, think carefully about how many test cases you'll need at a minimum, and write out (or type in) some good ones.
2. Follow the process from last week's lab and class to set up `.h`, `.u`, and `.cpp` files that have (for now) just the stuff for this function, and a readme for the directory. (There will be more functions later in the lab.)
3. If you haven't already, type your test cases into the `.u` file.
4. Write the function in the `.cpp` file, test it, and debug it.

I'll be circulating around the lab to answer questions. If you're stuck on some part of the drill, ask me about that (and while you're waiting for me to get to you, look at the next sections about searching). If you're not stuck but haven't finished the drill, work on that now. If you're done with the drill, continue on to the next section.

Command line FOTD: `grep`

The `grep` command is a general search tool that lets you find occurrences of some pattern in a whole batch of files. For instance, if you go to your old Lab 7 directory and type

```
grep numbers labfunc*.*
```

you'll get a listing of every time the `numbers` variable shows up (in either file); what it's doing is going through each source file and printing out every line that contains the string "`numbers`".

But `grep` is more powerful than that. Its first argument is what's called a "regular expression" or "regex", and lets you search for some pretty complicated things. You can get more information on this on your own, but a few quick tricks:

- By enclosing the pattern in (single) quotes, you can search for strings with spaces in them:

  ```
  grep 'int count' lab*.cpp
  ```

- If you want to match any single character, use a period:

  ```
  grep 'num . ' lab*.cpp
  ```

  matches any line that uses the `num` variable followed by an operator.

- To match any amount of any text, use the period wildcard with an asterisk:

  ```
  grep 'if.*num' lab*.cpp
  ```

- To match the beginning or end of the line, use caret and dollar-sign respectively.

  ```
  grep '^int' lab*.cpp
  ```

  will give just those lines that *start* with `int`.

Vim FOTD: searching

From command mode, if you hit the forward slash key, it's a little bit like colon mode: the cursor moves to the bottom of the screen and awaits further input. But what it's waiting for now is a regular expression to search for.

Having just explained regexes in the context of `grep`, there's not much more to explain here; they work essentially the same way. After the initial slash, you type a regex and hit enter, and Vim will find the next place in the file that matches that regex, or if there are none it will tell you that.

Also inside command mode, the `n` command will repeat the previous search. So pressing `n` repeatedly will cycle through all matches in a file. Using `N` instead goes through matches in reverse order.

The `n` command together with the period command (which repeats the previous command) is a workhorse combination: first, search for a pattern and do something; then alternate `n.n.n.` until you've done your action every place that pattern occurs.

More boolean stuff

Even more than last week, this one is a bunch of disjointed tasks that are loosely grouped as "involving booleans". Don't get overly wrapped up in the silly cover story for each one as long as the function performs correctly.[1]

For each function, write good test cases. Since we've been talking more about this in class, I'm stepping up a little on the test case requirements: to be good test cases with good coverage, you should make sure to cover things like edge cases (boundary conditions) where relevant, and all meaningfully different outcomes and logic paths.

- Write a function that computes a slot machine jackpot based on the three given digits: if all three are different, the payoff is $10. If all are the same, the payoff is $100 for 7s or $50 for 1s, and $25 for other digits. Otherwise (if there's one pair and a mismatch), the payoff is $0.

- Write a function that decides whether the children in a daycare will be allowed to play outside based on the given temperature and whether it's currently summertime. Most of the year, they'll be permitted to play if the temperature is between 65 and 85 degrees (inclusive); when it's not summertime they've brought coats and can play outside if temperatures are as low as 50.

  Note that in this case, the second parameter is itself a `bool` value!

- Write a function that chooses whether to answer a phone based on three given conditions: whether it's before 9am, whether it's your mom calling, and whether you're asleep. In the early morning, you only answer if it's your mom calling, but other times you'll answer no matter who's calling. If you're asleep, though, you can't answer, regardless of anything else.

[1] Several of these are adapted from problems at javabat.com.

- Write a function that evaluates words for appropriateness for a certain kind of themed puzzle: a given word is appropriate if it starts or ends with the given letter, but not both.

Handing in and rubric

Hand in as `lab9`. Due 4pm next Wednesday. Rubric TBA.