Name: ______________________________

Computer Science 330

Spring 2018 Midterm

Doodle here.

This is an individual-effort exam. You may read notes, textbooks, or the internet, but you may not post questions to online forums or share answers with classmates. Meaningful attempts must be made when answering questions. Write things that are both true and relevant. Exams with any blank or perfunctory answers will not be awarded partial credit.

1. *Matchmaker*

Consider this terse summary of regular expression parts and pieces:

match what?	how many times?	where?
`(pat)` - group and capture	`pat?` - 0 or 1, greedy	`^` - at beginning of line
`pat1|pat2` - `pat1` or `pat2`	`pat*` - 0 or more, greedy	`$` - at end of line
`[abc]` - `a`, `b`, or `c` (complement: `[^abc]`)	`pat*?` - 0 or more, lazy	`\b` - at word boundary
`\d` - any digit (complement: `\D`)	`pat+` - 1 or more, greedy	`(?<=pat)` - after `pat`
`\w` - any alphanumeric (complement: `\W`)	`pat+?` - 1 or more, lazy	`(?=pat)` - before `pat`
`\s` - any whitespace (complement: `\S`)	`pat{n}` - `n`	`(?!pat)` - not before `pat`
`.` - any non-newline character	`pat{n,}` - `n` or more, greedy	`(?<!pat)` - not after `pat`

Write regular expressions to match the following criteria and only the following criteria. Write regular expressions only, not Ruby code. Be sure to test your solutions on a computer. You can do so at the command-line. Run `irb` and enter `"str" =~ /regex/`.

(a) Any string that contains an integer comprised of $3n + 1$ digits, for $n \in \mathbb{N}$. For example, these strings match: `"ur 8"`, `"4002"`, `"Ticket #3435947"`. These do not: `"44th ain't bad."`, `"div-921"`, `"[10230]"`.

(b) Any string whose entire contents are a septendecimal (base-17) literal. Allow for either upper- or lowercase letters.

(c) Any string that contains a block of text that begins with `$$` and ends with `$$`. Match only a single block.

(d) Any string whose entire contents are of the form `CITY, STATE ZIP`. `STATE` is a two-letter state abbreviation. `ZIP` is a either a 5-digit or ZIP+4 ZIP code. Capture the three pieces information and only those three. For example, matching `"Eau Claire, WI 54702-4004"` should capture `"Eau Claire"`, `"WI"`, and `"54702-4004"`.

(e) Any string that contains a sequence of space or tab characters at the end of a line. Match the whole sequence spaces and tabs. Such trailing spaces are a scourge.

2. *Eight Days a Week*

   C's `enum`s are not typesafe. For instance, consider the function `is_weekday` below. Any `int` may be passed to this function—not just the seven members of the enumerated set:

   ```
   typedef enum {M, T, W, R, F, SA, SU} weekday_t;
   int is_weekday(weekday_t wday) {
     ...
   }
   ```

   The developers of C almost certainly were aware of this danger. Investigate `enum`s in C. What are some reasons they may have allowed `enum`s to not be typesafe?

3. *Underloaded*

   Function overloading is supported in Java, C++, and C#. But overloading is not supported in Python, Ruby, or Javascript. What accounts for this difference?

4. *Fencepost*

   You are writing a programming language that directly supports fencepost (or loop-and-a-half) algorithms with a `repeat-around` loop. Such a loop is built out of two blocks separated by the keyword `around` and a count of repetitions for the outer block. Consider this loop:

```
repeat n
  print '|'
around
  print '-'
end
```

   If `n` is 3 at the time of execution, this code prints `|-|-|`. In Javascript, implement the class `StatementRepeatAround` similar to the AST models we wrote together in class. Provide a constructor and an `evaluate` method, whose only parameter is the current environment. Write only the code for the class—not any lexer or parser code.

5. *Crashing It*

   A pointer that points to a freed block of memory is called a *stale pointer*. Stale pointers are insidious in that dereferencing them does not always fail, even though it should. If the pointer was just magically set to `NULL` upon being freed, then you'd at least get a segmentation fault, which is far preferable. Your friend writes this wrapper around `free` to do just this for `int`s:

```
void freeint(int *p) {
  free(p);
  p = NULL;
}
```

   Try this code out with our own `main` that allocates and frees an `int`—and then dereferences the freed `int`. See if it crashes as your friend expects. Spoiler: it almost certainly doesn't. First, describe what's wrong. Second, rewrite the code to work as intended.

   Extra challenge: after writing `freeint`, write a more general `stalefree` that works for all types but still compiles without warnings.