This is a 24 hour, open notes, books, etc. exam.
ASK if anything is not clear.
WORK INDIVIDUALLY.

If there are any corrections to exam questions, they will be posted to Canvas.

Question	Points	Score
1	15	
2	15	
3	15	
4	15	
5	15	
6	15	
7	10	
Total:	100	

Notation:

$\sigma(x)$ – Get reference x from store σ.
$\sigma[x := v]$ – Set reference x to value v in store σ.

1. (15 points) Consider the following language:

$e ::=$		*Expressions*
	n	number (integer)
	`set` x e	set variable
	`read` x	read variable
	`inc` x	increment operator
	e `then` e	then expression
	`test` e e	test expression
	`loop` e e	loop
	`crash`	crash expression
$x ::=$		*Variables*

The big-step operational semantics relation evaluates an expression to a **number**.

$$e, \sigma \Downarrow n, \sigma'$$

The store σ is a mapping of references (x) to numbers (n). The evaluation rules are provided below. Note that `crash` has no evaluation rule, meaning that it causes evaluation to get stuck.

$$[\textsc{Num}] \quad \frac{}{n, \sigma \Downarrow n, \sigma}$$

$$[\textsc{SetVar}] \quad \frac{e, \sigma \Downarrow n, \sigma_1 \quad \sigma' = \sigma_1[x := n]}{\texttt{set}\ x\ e, \sigma \Downarrow n, \sigma'}$$

$$[\textsc{ReadVar}] \quad \frac{x \in \mathit{domain}(\sigma)}{\texttt{read}\ x, \sigma \Downarrow \sigma(x), \sigma}$$

$$[\textsc{Inc}] \quad \frac{e, \sigma \Downarrow n, \sigma' \quad n' = n + 1}{\texttt{inc}\ e, \sigma \Downarrow n', \sigma'}$$

$$[\textsc{Then}] \quad \frac{e_1, \sigma \Downarrow n_1, \sigma_1 \quad e_2, \sigma_1 \Downarrow n_2, \sigma'}{e_1\ \texttt{then}\ e_2, \sigma \Downarrow n_2, \sigma'}$$

$$[\textsc{TestEq}] \quad \frac{e_1, \sigma \Downarrow n_1, \sigma_1 \quad e_2, \sigma_1 \Downarrow n_2, \sigma' \quad n1 = n2}{\texttt{test}\ e_1\ e_2, \sigma \Downarrow 1, \sigma'}$$

$$[\textsc{TestNeq}] \quad \frac{e_1, \sigma \Downarrow n_1, \sigma_1 \quad e_2, \sigma_1 \Downarrow n_2, \sigma' \quad n1 \neq n2}{\texttt{test}\ e_1\ e_2, \sigma \Downarrow 0, \sigma'}$$

$$[\textsc{LoopTrue}] \quad \frac{e_1, \sigma \Downarrow 1, \sigma'}{\texttt{loop}\ e_1\ e_2, \sigma \Downarrow 1, \sigma'}$$

$$[\textsc{LoopFalse}] \quad \frac{e_1, \sigma \Downarrow 0, \sigma_1 \quad e_2, \sigma_1 \Downarrow n, \sigma_2 \quad \texttt{loop}\ e_1\ e_2, \sigma_2 \Downarrow n', \sigma'}{\texttt{loop}\ e_1\ e_2, \sigma \Downarrow n', \sigma'}$$

In the `prob1` directory of the exam zip file, you will find `interp.hs`. The `Expression` data type matching this language has been provided for you. Complete the `evaluate` function so that its behavior matches the evaluation rules defined above.

Any error cases should return `Nothing` rather than calling `error`.

The `test.hs` file has several cases for you to consider. The expected results are not specified, but you should be able to read the evaluation rules to see what the result of any program should be. Your may run the test cases from the command line by typing:

```
$ runhaskell test.hs
```

2. (15 points) Consider the following language and big-step operational semantics:

$e ::=$		*Expressions*
	v	value
	`if` (e) `then` (e) `else` (e)	if expression
	`rotate` e	rotate expression
$v ::=$		*Values*
	`red`	red
	`green`	green
	`blue`	blue

$$[\text{IF-CTXT}] \quad \frac{e_1 \to e_1'}{\texttt{if } (e_1) \texttt{ then } (e_2) \texttt{ else } (e_3) \to \texttt{if } (e_1') \texttt{ then } (e_2) \texttt{ else } (e_3)}$$

$$[\text{IF-RED}] \quad \frac{}{\texttt{if } (\texttt{red}) \texttt{ then } (e_1) \texttt{ else } (e_2) \to e_2}$$

$$[\text{IF-OTHER}] \quad \frac{v \neq \texttt{red}}{\texttt{if } (v) \texttt{ then } (e_1) \texttt{ else } (e_2) \to e_1}$$

$$[\text{ROT-CTXT}] \quad \frac{e \Downarrow e'}{\texttt{rotate } e \Downarrow \texttt{rotate } e'}$$

$$[\text{ROT-RED}] \quad \frac{}{\texttt{rotate red} \Downarrow \texttt{green}}$$

$$[\text{ROT-GREEN}] \quad \frac{}{\texttt{rotate green} \Downarrow \texttt{blue}}$$

$$[\text{ROT-BLUE}] \quad \frac{}{\texttt{rotate blue} \Downarrow \texttt{red}}$$

Write equivalent big-step operational semantics for this language. Submit a PDF of your semantics. (The LaTeX of this exam is included in the exam zip file. There are some latex commands in it that you may find useful.)

3. (15 points) From the exam zip file, modify `prob3/employees.hs` to implement the `totalManagerPayroll` and `empsWithPayLowerThan` functions.

   You must use higher-order functions for full credit, and you may not use recursion in either of your solutions.

4. (15 points) From the exam zip file, modify `prob4/linklist.lhs` to add support for linked lists to be used as functors, applicative functors, and monads.

   Support for functors is worth 8 points, support for applicative functors is worth 5 points, and support for monads is worth 2 points.

5. (15 points) From the exam zip file, modify `prob5/trie.lhs` to implement the 'contains' function.

6. (15 points) From the exam zip file, modify `prob6/schemeParser.lhs` to implement a parser for a (very minimal) Scheme program parser. The `prob6/test.scm` file has a sample Scheme file for you to consider.

   Test your parser by calling:

   ```
   $ runhaskell schemeParser.hs test.scm
   ```

7. (10 points) When submitting your exam, include a text file with the following text:

   > I have not worked with anyone else for these exam problems. I have not consulted with any outside parties. For any code that I have used from an external source, I have cited the original source within my code comments.