Exercise 1F-2. Language Design [5 points]. Comment on some aspect from Hoare's *Hints On Programming Language Design* that relates to your programming experience. Provide additional evidence in favor of one his points and against one of his points. Do not exceed three paragraphs. Both your ideas and also the clarity with which they are expressed (i.e., your English prose) matter. Readers should be able to identify your main claim, the arguments you are bringing to bear, and your conclusion.

Hoare introduced many interesting points in *Hints On Programming Language*. First Hoare emphasised the importance on fast translation and shared some thoughts on how to make compilers faster. I took several compiler related courses, and I really agree that prescan can make compilers run much faster. Based on my experience, the scanner I wrote ran super slow compared to other parts of my compiler. However, I would also want to extend this point. I believe Hoare ignored the fact that there is a trade off between the efficiency of compiler and the speed of code. I wrote a compiler with some optimizations like Dead Code Elimination, Common Subexpression Elimination and Forward Copy Propagation. I also designed a Register Allocation scheme. These all made my compiler run slower, but my code ran faster later. Therefore, I believe we should not only focus on the fast translation itself, but also notice this trade off.

Hoare put emphasis on programming documentation. I strongly agree with that. During my several research experience, I found many codes are extremely long and without documentation. It wasted me much time to just make these codes run. Debugging on badly documented code was extremely painful.

Last interesting point from Hoare's article is that he claims it's almost impossible to persuade programmers to change their programming languages. I'm against this claim. After taking several compiler related courses and programming for years, I found programming is an way of thinking. Once you get the higher level view of programming for example the design principle of programming language. I think we will able to appreciate the strengths and weaknesses of different programming languages. In this way, it will make it easier for us to pick up a new programming language and make us more open-minded to different programming languages.

Exercise 1F-3. Simple Operational Semantics [3 points]. Consider the IMP language discussed in class, with the Aexp sub-language extended with a division operator. Explain what changes must be made to the operational semantics (big-step only). Write out formally any new rules of inference you introduce.

To add a division operator, we need to allow the expressions to be able to change the state. Because divided by 0 will lead to an error. Define the error state as σ_e.

$$\frac{< e_2, \sigma > \Downarrow 0}{< e_1/e_2, \sigma > \Downarrow \sigma_e}$$

$$\frac{< e_1, \sigma > \Downarrow n_1 \quad < e_2, \sigma > \Downarrow n_2}{< e_1/e_2, \sigma > \Downarrow \lfloor n_1/n_2 \rfloor}$$

Exercise 1F-4. Language Feature Design, Large Step [10 points]. Consider the IMP language with a new command construct "`let` $x = e$ `in` c". The informal semantics of this construct is that the Aexp e is evaluated and then a new local variable x is created with lexical scope c and initialized with the result of evaluating e. Then the command c is evaluated. We also extend IMP with a new command "`print` e" which evaluates the Aexp e and "displays the result" in some un-modeled manner but is otherwise similar to `skip`.

We expect (the curly braces are syntactic sugar):

```
x := 1 ;
y := 2 ;
{ let x = 3 in
  print x ;
  print y ;
  x := 4 ;
  y := 5
} ;
print x ;
print y
```

to display "3 2 1 5".

Extend the natural-style operational semantics judgment $\langle c, \sigma \rangle \Downarrow \sigma'$ with one new rule for dealing with the `let` command. Pay careful attention to the scope of the newly declared variable and to changes to other variables.

$$\frac{< e, \sigma > \Downarrow n \qquad < x := e, \sigma > \Downarrow \sigma'' \qquad < c;\ x := \sigma(x), \sigma'' > \Downarrow \sigma'}{< \text{let } x = e \text{ in } c, \sigma > \Downarrow \sigma'}$$

Exercise 1F-5. Language Feature Design, Small Step [10 points]. Extend the set of redexes, contexts and reduction rules for the contextual-style operational semantics that we discussed in class to account for the `let` command introduced above.

Context:

$$H ::== \cdots \\ |\text{let } H \text{ in } c$$

Redex:

$$r ::== \cdots \\ |\text{let } x = n \text{ in } c$$

Reduction rule:

$$< \text{let } x = n \text{ in } c, \sigma > \rightarrow < x := n; c; x := \sigma(x), \sigma >$$

For the reduction rule, $x := n; c; x := \sigma(x)$ can be regarded as c'

1 HW1 (select all pages: your first page has your name and bookkeeping, and all others are anonymous))

- **0 pts** Correct