All subsequent answers should appear after the first page of your submission and may be shared publicly during peer review.

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.

Answer 1F-2. Hoare argues that fast translation is one of the criteria for good language desing, which is reasonable. I agree with that, because fast translation saves time in programming development and debugging. Few programmers could know the compilation result for sure because of the length and complexity of the code. Most of programmers need compile the code first and then debug. The waiting time while the code is being compiled is most probably wasted.

But from Hoare's point of view, independent compilation is a poor substitute for fast translation. I cannot agree with this point. First, independent compilation is not a substitute for fast compilation. They can exist in the same time for different purposes. Independent compilation not only enables programmers to only recompile parts of their program but also encourages compiled libraries to be reused. Moreover, fast translation is not achievable without independent compilation nowadays. Translating 27.8 million lines of code in the linux kernel requires an enormous effort. It is not enough only using the three techniques Hoare suggested, prescaning, precompilation and dum. For example, the technique of precompilation is effective only on single-pass compilers. However, multi-pass compilers is widely used today. Since the multiple passes include a modular structure, and the code generation decoupled from the other steps of the compiler, the passes can be reused for different hardware/machines.

In conclusion, I agree that fast translation is an important creterion for programming language but I doubt Hoare's argument in fast translation. Independent compilation is also a good practice for fast translation.

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.

Answer 1F-3. I added two rules for the division operator, because division by zero is undefined.
When $n_2 \neq 0$,

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \quad \langle e_2, \sigma\rangle \Downarrow n_2}{\langle e_1/e_2, \sigma\rangle \Downarrow n_1/n_2}$$

When the divisor evaluates to zero,

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \quad \langle e_2, \sigma\rangle \Downarrow 0}{\langle e_1/e_2, \sigma\rangle \Downarrow \mathit{ZeroDivisionError}}$$

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.

Answer 1F-4. `let` command assigns a value to a local variable, which is effective during the command c. After the command c, the local variable will not hold the value any more. The value of the variable name will restore to the state before this `let` statement but other effects of the command c will still remain.

The added rule is

$$\frac{\langle e, \sigma\rangle \Downarrow n \quad \langle c, \sigma[x := n]\rangle \Downarrow \sigma'}{\langle \texttt{let}\ x = e\ \texttt{in}\ c, \sigma\rangle \Downarrow \sigma'[x := \sigma(x)]}$$

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.

Answer 1F-5. An expression is added to the set of redexes.

$$r ::= ...|\texttt{let}\ x := n\ \texttt{in}\ c$$

Nothing is added to the set of contexts. Because firstly `let` statement is reduced to three statements, which can be further reduced.

Added reduction rule:

$$\langle \texttt{let}\ x = n\ \texttt{in}\ c, \sigma\rangle \rightarrow \langle x := n; c; x := \sigma(x), \sigma\rangle,$$

where σ is the state before `let` command. Then, we can use other reduction rules to process the expressions.

Exercise 1C. Language Feature Design, Coding. Download the Homework 1 code pack from the course web page. Modify `hw1.ml` so that it implements a complete interpreter for IMP (including `let` and `print`). Base your interpreter on IMP's large-step operational semantics. The `Makefile` includes a "`make test`" target that you should use (at least) to test your work.

Modify the file `example-imp-command` so that it contains a "tricky" terminating IMP command that can be parsed by our IMP test harness (e.g., "`imp < example-imp-command`" should not yield a parse error).

Submission. Turn in the formal component of the assignment (1F-1 through 1F-5) as a single PDF document via the `gradescope` website. Your name and Michigan email address must appear on the first page of your PDF submission but may not appear anywhere else. Turn in the coding component of the assignment (1C) via the `autograder.io` website.

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

- **0 pts** Correct