You can turn in handwritten solutions to this assignment. Please write clearly and use standard-sized (8.5 by 11in) paper. If you choose to typeset your solutions using LaTeX, you may find the mathpartir.sty package useful.

1. **Warmup** (10 pts.)

   Consider the following command and store in the IMP language we covered in class:

$$\begin{array}{rcl} c_0 & = & \textsf{if}\ ((9+4) > 10)\ \textsf{then}\ x := 42\ \textsf{else}\ x := 2 \\ \sigma_0 & = & \{x \mapsto 0\} \end{array}$$

   (a) Draw the complete derivation tree showing *one* reduction step (small-step reduction) for $\langle c_0, \sigma_0 \rangle$.

   (b) Draw the complete derivation tree showing the big-step evaluation of $\langle c_0, \sigma_0 \rangle$.

2. **For Loop** (25 pts.)

   Consider IMP_{FOR}, a version of IMP that has for loops instead of while loops. We redefine commands c as follows:

$$c \ ::= \ \textsf{skip} \ \mid \ x := a \ \mid \ c_0; c_1 \ \mid \ \textsf{if}\ b\ \textsf{then}\ c_1\ \textsf{else}\ c_2 \ \mid \ \textsf{for}\ x = a_0\ \textsf{to}\ a_1\ \textsf{do}\ c$$

   Informally, the for loop works as follows. When entering the loop $\textsf{for}\ x = a_0\ \textsf{to}\ a_1\ \textsf{do}\ c$, the expression a_0 is evaluated to an integer n_0 and the expression a_1 is evaluated to an integer n_1. If $n_0 > n_1$, the command just behaves like skip. If $n_0 \leq n_1$, the body c is executed $n_1 - n_0 + 1$ times, with x assigned the value $n_0 + i - 1$ at the beginning of the ith loop iteration. (For instance, if $n_0 = 3$ and $n_1 = 5$, the body c will be executed 3 times, with x assigned the values 3, 4, and 5 at the beginning of the first, second, and third iteration, respectively.) Note that the loop bounds are computed once at the beginning of the loop, and no computation in the body of the loop can change the number of times the loop is executed. That is, although the loop index variable x can be assigned within the body c of the loop, these assignments do not affect the value of x at the beginning of the next loop iteration.

   (a) Write a big-step operational semantics for the $\textsf{for}\ x = a_0\ \textsf{to}\ a_1\ \textsf{do}\ c$ construct.

   (b) Write an IMP_{FOR} program that, given an input value in the variable n, computes the nth Fibonacci number $F(n)$ (where $F(0) = 0$, $F(1) = 1$, and $F(n) = F(n-1) + F(n-2)$), and returns the result in variable r. You may assume that you have multiplication, addition, and subtraction as built-in arithmetic operators.

3. **Dangling references** (50 pts.)

   In class we claimed that during evaluation, uML! programs never generate dangling references. Let's prove it. Consider the fragment of uML! consisting of the following expressions and values:

$$\begin{array}{rcl} e & ::= & n \mid x \mid \textsf{ref}\ e \mid\ !e \mid e_1 := e_2 \mid \textsf{null} \mid \lambda x.\, e \mid e_1\ e_2 \mid \\ & & \textsf{let}\ x = e_1\ \textsf{in}\ e_2 \mid (e_1,\, e_2) \mid \textsf{let}\ (x, y) = e_1\ \textsf{in}\ e_2 \\ v & ::= & n \mid (v_1,\, v_2) \mid \textsf{null} \mid \lambda x.\, e \ \ (\text{where } \lambda x.\, e \text{ is closed}) \end{array}$$

   To define the small-step semantics of uML!, we augment the grammar of expressions and values with a set of locations $\ell \in \textsf{Loc}$.

$$\begin{array}{rcl} e & ::= & \ldots \mid \ell \\ v & ::= & \ldots \mid \ell \end{array}$$

   A *store* σ is a partial map from locations to values (which could be other locations). The small-step semantics of uML! programs was defined in terms of *configurations* $\langle e,\, \sigma \rangle$, where e is an augmented expression and σ is a store. (For your reference, the small-step operational semantics of uML! is given at the end of this document.)

We define $loc(e)$ to be the set of locations that occur in the expression e. Thus, for example, $loc((!\,\ell_2)\ (\lambda x.\,(!\,\ell_1)+!\,(\mathsf{ref}\ 4))) = \{\ell_1, \ell_2\}$.

A uML! *program* is a closed expression that does not contain any locations. Thus, if e is a program then $loc(e) = \emptyset$.

(a) Consider the following uML! configuration:

$$\langle\ (\lambda x.\,(!\,\ell_1)\ 2)\ (\mathsf{ref}\ 1)\ ,\ \ \{\ell_1 \mapsto \lambda y.\,\mathsf{ref}\ y\}\ \rangle$$

Show the evaluation of this configuration. For each configuration $\langle e', \sigma'\rangle$ in the evaluation, give $loc(e')$.

(b) Give an inductive definition of the set $loc(e)$ of locations occurring in e.

(c) Prove that if e is a uML! program and $\langle e, \emptyset\rangle \longrightarrow^* \langle e', \sigma\rangle$, then $loc(e') \subseteq \mathrm{dom}(\sigma)$. If you use induction, identify what you are doing induction on.

Small-Step Operational Semantics of uML!

Evaluation contexts

$$\begin{array}{rcl} E & ::= & [\cdot]\ \mid\ \mathsf{ref}\ E\ \mid\ !\,E\ \mid\ E := e_2\ \mid\ v_1 := E\ \mid\ E\ e_2\ \mid\ v_1\ E\ \mid \\ & & \mathsf{let}\ x = E\ \mathsf{in}\ e_2\ \mid\ (E,\, e_2)\ \mid\ (v_1,\, E)\ \mid\ \mathsf{let}\ (x,y) = E\ \mathsf{in}\ e_2 \end{array}$$

Reductions

$$\begin{array}{rcll} \langle \mathsf{ref}\ v,\ \sigma\rangle & \longrightarrow & \langle \ell,\ \sigma[\ell \mapsto v]\rangle & (\text{where } \ell \notin \mathrm{dom}(\sigma)) \\ \langle !\,\ell,\ \sigma\rangle & \longrightarrow & \langle \sigma(\ell),\ \sigma\rangle & (\text{where } \ell \in \mathrm{dom}(\sigma)) \\ \langle \ell := v,\ \sigma\rangle & \longrightarrow & \langle \mathsf{null},\ \sigma[\ell \mapsto v]\rangle & (\text{where } \ell \in \mathrm{dom}(\sigma)) \\ \langle (\lambda x.\,e)\ v,\ \sigma\rangle & \longrightarrow & \langle e[v/x],\ \sigma\rangle & \\ \langle \mathsf{let}\ x = v\ \mathsf{in}\ e,\ \sigma\rangle & \longrightarrow & \langle e[v/x],\ \sigma\rangle & \\ \langle \mathsf{let}\ (x,y) = (v_1, v_2)\ \mathsf{in}\ e,\ \sigma\rangle & \longrightarrow & \langle e[v_1/x][v_2/y],\ \sigma\rangle & \end{array}$$

Context rule

$$\frac{\langle e,\ \sigma\rangle\ \longrightarrow\ \langle e',\ \sigma'\rangle}{\langle E[e],\ \sigma\rangle\ \longrightarrow\ \langle E[e'],\ \sigma'\rangle}$$