This assigment consists of a written part and a programming part. Your solutions to the written part (Problem 1) should be typeset using LaTeX or Word. If you use LaTeX, there is a template available for your use at the course website.

You should submit two files by 3:00pm on the due date:

(i) a pdf file containing the solutions to the written part (Problem 1), and

(ii) a tar file containing all the files that make up the solutions to the programming part (Problems 2,3,4). Specifically, the tar file must include the following files:

- `a.ml`
- `b.ml`
- `lam.ml`
- `logic.ml`

1. **For Loop** (20 pts.)

   Consider $\mathrm{IMP}_{\mathrm{FOR}}$, a version of IMP that has **for** loops instead of **while** loops. We redefine commands c as follows:

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

   Informally, the **for** loop works as follows. When entering the loop $\mathbf{for}\, x = a_0\, \mathbf{to}\, a_1\, \mathbf{do}\, c$, 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 $\mathbf{for}\, x = a_0\, \mathbf{to}\, a_1\, \mathbf{do}\, c$ construct.

   (b) Write an $\mathrm{IMP}_{\mathrm{FOR}}$ 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.

2. **OCaml Warmup** (15 pts.)

   In order to program effectively in OCaml, you need to be able to understand the type inference error messages that the compiler will tell you when you make a mistake in a program. These messages can be quite cryptic!

   The goal of this problem is make as few changes as possible to the files `a.ml` and `b.ml` so that they type check. In other words, don't change 2 lines when you could change 1 line in the file. Don't change 2 operators when you could change 1 operator. (I'm not going to be too strict about this but basically I want to see if you can exactly pinpoint where the type error is in the file and change it.) Hand in your changed versions of `a.ml` and `b.ml`.

   To solve this problem, start the `ocaml` interactive system. At the prompt, type

   #use "a.ml";;

then look at the error message you get. Then use the information in that error message (or warning) to make a change to the file so that it type checks. Try doing `#use ''a.ml''` again to see if there are any more errors. Repeat until there are no errors *or warnings*. Do this for both `a.ml` and `b.ml`. The files are here:

`http://www.cs.indiana.edu/classes/b522/hw2-code.html`

3. **Lambda Calculus** (40 pts.)

   Implement and test the following functions in ML by downloading and modifying the file

   `http://www.cs.indiana.edu/classes/b522/hw2-code/lam.ml`

   - `fv e`, which computes the set of free variables of a lambda-calculus term `e`;
   - `print_varset s`, which prints out a set of variables `s`;
   - `subst e e1 x`, which implements capture-avoiding substitution `e{e1/x}`;
   - `isval e`, which checks if a term `e` is a value; and
   - `eval e`, which evaluates the term `e` using the big-step, call-by-value operational semantics.

   The file `lam.ml` contains further details. Make sure you test all your functions!

   Once you have implemented these functions, you can take a look at how they work together using the function `evalsto` that is defined for you in the `Lam` module (i.e., in `lam.ml`).

4. **Simple Logic** (25 pts.)

   Here is the definition of a simple logic with **true**, **false**, inequalities on natural numbers, conjunction and disjunction. Below n represents natural numbers. The formulas F are as follows:

$$\begin{array}{rcl} n & ::= & \mathbf{Z} \mid \mathbf{S}\ n \\ F & ::= & \mathbf{true} \mid \mathbf{false} \mid n_1 \leq n_2 \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \end{array}$$

   The truth of a logical formula is determined by judgements with the form $\mathsf{Leq}\ n_1\ n_2$ and $F\ \mathsf{Valid}$.

   Here are the rules for $\mathsf{Leq}\ n_1\ n_2$:

$$\frac{}{\mathsf{Leq}\ \mathbf{Z}\ n_2} \qquad \frac{\mathsf{Leq}\ n_1\ n_2}{\mathsf{Leq}\ (\mathbf{S}\ n_1)\ (\mathbf{S}\ n_2)}$$

   Here are the rules for proving $F\ \mathsf{Valid}$:

$$\frac{}{\mathbf{true}\ \mathsf{Valid}} \qquad \frac{\mathsf{Leq}\ n_1\ n_2}{n_1 \leq n_2\ \mathsf{Valid}}$$

$$\frac{F_1\ \mathsf{Valid} \qquad F_2\ \mathsf{Valid}}{F_1 \wedge F_2\ \mathsf{Valid}} \qquad \frac{F_1\ \mathsf{Valid}}{F_1 \vee F_2\ \mathsf{Valid}} \qquad \frac{F_2\ \mathsf{Valid}}{F_1 \vee F_2\ \mathsf{Valid}}$$

   Now, download the file `logic.ml` from here:

   `http://www.cs.indiana.edu/classes/b522/hw2-code/logic.ml`

   and fill in the missing definitions. It is possible to get bonus points on this problem. See the file `logic.ml` for additional details and directions.