The purpose of this exercise is to gain familiarity with \textit{constraint-based analyses}, particularly 0CFA (zeroth-order control-flow analysis) from Lecture 11.

1. (a) What is a higher-order function?
   (b) How do higher-order functions make it harder to predict control flow within a program?
   (c) How does the 0CFA help to predict control flow?
   (d) Do object-oriented programs have analysis issues related to higher-order functions?

Consider the following simple λ-calculus like language, call it $\mathcal{L}$:

$$e ::= v \mid c \mid \lambda v.e \mid e_1 e_2 \mid \text{let } v = e_1 \text{ in } e_2 \mid \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \mid e_1 \oplus e_2$$

where $v$ ranges over variables, $c$ ranges over integer constants, and $\oplus$ ranges over binary operations.

0CFA computes information about control flow in a program by computing a subset of a program’s data flow: the flow of functions (or function pointers). In the following, the data flow of integer constants will also be tracked to aid understanding.

2. (a) Define informally the notion of a \textit{binding site} and \textit{use site} and indicate the binding and use sites in the syntax of $\mathcal{L}$.
   (b) The following expression has a single \textit{program point} labelling the formal parameter $x$ of $f$:

   $$\text{let } f = (\lambda x^0.x + x) \text{ in } f \, 2 + f \, 3$$

   Label the remaining program points (it may help to write the expression as a tree).
   (c) Given \textit{flow variables} $\alpha_i$ associating sets to each program point, what is the value of set $\alpha_0$ following a 0CFA? What integer values flow out of the body of the $\lambda$?
   (d) Write down and explain the rule for generating constraints for \textit{let}-bindings and variables $v$.
   (e) Consider the following expression with a partial labelling of program points:

   $$\text{let } f = (\lambda x.x^1 \, 0) \text{ in } (\text{let } g = (\lambda y^0.y + 1) \text{ in } (f \, g) + (g \, 1))$$

   Compute the flow sets for $\alpha_1$ and $\alpha_0$.

3. (a) Calculate a full 0CFA (tracking just function values, not integer values) for the following expression:

   $$\text{let } f = (\lambda x.x \, 0) \text{ in } (f \, (\lambda y.y \, * \, 3)) + (f \, (\lambda z.z + 1))$$
(b) Write down and explain the rule for generating constraints for functions and function application.

4. Answer the following past paper questions:
   - 2004 Paper 9 Question 3
   - 2007 Paper 9 Question 16 (using the constraint-based analysis approach for part (b))

In question 2007, by escaping we mean that some part of a list passed as an input may be returned as part of the result. For example, given $f(x) = \text{tl}(x)$, the argument $x$ may escape (even though it will be just some cons cell and whenever the list $x$ is non-empty).

Think for example:

```plaintext
L = \text{cons}(\ldots, []);
x = f(L);
\ldots
use(x)
\ldots
```

If we know that argument of $f$ does not escape and the list $L$ is not used after the function call, then we can free all memory allocated for L (because we know that $x$ cannot point there). We have to be more careful if $[]$ above is some pre-existing list.

Past exam questions can be found at:
http://www.cl.cam.ac.uk/teaching/exams/pastpapers/t-OptimisingCompilers.html