Homework Assignment 6

Any automatically graded answer may be manually graded by the instructor. Submissions are expected to only use functions taught in the course. If a submission uses a disallowed function, that exercise can get zero points. Excluding promises, *all functions that mutate values are disallowed* (mutable functions usually have a `!` in their name).

Memory management

1. (30 points) Implement function `frame-refs` that returns the set of all handles contained in the given frame, according to the notion of *contains* introduced in Lecture 16. *Hint:* Consider using function `frame-fold`. If your solution uses `frame-fold`, then the solution should **not** be recursive.

   ```
   (-> frame? set?)
   ```

2. (20 points) Implement function `(mem-mark contained mem ref)` that returns the set of all reachable handles from handle `ref`, and takes: `contained` a function that takes an element of the heap and returns a set of handles contained in that element, `mem` is a heap, and `ref` is the initial handle, according to the memory sweep algorithm discussed in Lecture 16. An example of a `contained` function is function `frames-refs` for a heap of frames. Notice that function `mem-mark` expects a heap of any data.

   ```
   (-> (-> any/c set?) heap? handle? set?)
   ```

3. (10 points) Implement function `(mem-sweep mem to-keep)` that given a heap `mem` and a set of handles to keep (parameter `to-keep`) returns a new heap of frames that only contains the handles in the given set. *Hint:* Peruse `hw6-util.rkt` for heap-related functions. The solution should be a single function call.

   ```
   (-> heap? set? heap?)
   ```

Monads

4. (20 points) Implement the monadic operation `(mlist bind pure args)` that takes a list `args` of monadic operations and binds each result in a new list. Do not hardcode the implementation of `mlist` to `eff-bind` or `eff-pure`. *Hint:* Check if your code works with `list-bind` and `list-pure`.

   For instance, `(mlist eff-bind eff-pure (list op1 op2 op3))` is equivalent to the code below

   ```
   (do
     x1 <- op1 ; using eff-bind
     x2 <- op2 ; using eff-bind
     x3 <- op3 ; using eff-bind
     (eff-pure (list x1 x2 x3)))
   ```

5. (10 points) Implement the monadic operation `(mapply bind pure f . args)` that takes a function `f`, a variable number of monadic operations `args`, and returns the pure result of applying function `f` to a list that is obtained by binding the result of each monadic operation in `args` into a list (by the same order). The solution can be solved by using function `mlist`. *Hint:* this exercise should `bind` the result of `mlist`, and feed it to function `apply`.

   For instance, `(mapply eff-bind eff-pure * op1 op2 op3)` is equivalent to the code below

   ```
   (do
     x1 <- op1 ; using eff-bind
     x2 <- op2 ; using eff-bind
     x3 <- op3 ; using eff-bind
     (eff-pure (* x1 x2 x3)))
   ```

Manually graded questions

6. (10 points) **Manually graded.** Consider memory management via reference counting and the increment count operation. Suppose that the reference count algorithm is faulty and the reference count overflows resetting back to zero. *Discuss if overflowing the reference count affects soundness or completeness of memory management.*