CS112 Assignment 1

Instructions: You do not need to answer all questions. It is possible to get a perfect score without doing so. Look at the end of this file to see what we mean. Make a PDF or TXT file with your answers to the non-programming questions. Specify which question you are answering by writing the question number before your answer. Answer as clearly and as concisely as you can. For the programming questions, create individual Racket files with the specified names. Assume standard Racket unless otherwise specified. You may write as many definitions as you need in the file, as long as the file contains the required definition. **IMPORTANT: Make sure the required definition is correctly named, or the automatic checker may fail and you will earn no points for that question.** ZIP all the files together with a Certificate of Authorship, name it `<firstname>.<lastname>.1.zip`, and submit to Google Classroom.

For items 1-3, assume that y is a non-negative integer and always name the function `exp`. You should not use the built-in functions `exp` and `expt`.

1. Write a function that given x and y, computes x^y by y repeated multiplications. Write a recursive version first, and save it as `linear-exp-recursive.rkt`. Then write an iterative (tail recursive) version and save it as `linear-exp-iterative.rkt`.

Points for this item:

`linear-exp-recursive.rkt` (1 point)

`linear-exp-iterative.rkt` (2 points)

2. Solve the same problem, but this time cutting down the number of required multiplications to $O(\lg y)$, using the technique of *exponentiation by repeated squaring*. Exponentiation by repeated squaring works according to the following observation:

$$x^y = \begin{cases} \left(x^{\frac{y}{2}}\right)^2 & if\ y\ is\ even \\ x\left(x^{\left\lfloor\frac{y}{2}\right\rfloor}\right)^2 & if\ y\ is\ odd \end{cases}$$

The base case is trivial and is intentionally left out. Write a recursive version first, and save it as `log-exp-recursive.rkt`. Then write an iterative (tail recursive) version and save it as `log-exp-iterative.rkt`.

Points for this item:

`log-exp-recursive.rkt` (2 points)

`log-exp-iterative.rkt` (3 points)

3. Write an exponentiation function that requires $O(\lg y)$ multiplications, assuming that Racket has an call-by-value eager evaluation strategy; but $O(y)$ multiplications, assuming that Racket has a call-by-name lazy evaluation strategy (no memoization); and $O(\lg y)$ multiplications, assuming that Racket has a call-by-need lazy evaluation strategy (with memoization). Save it as `lazy-vs-eager-exp.rkt`. Hint: try to analyze the efficiency of your `exp` definitions above, assuming the three different evaluation strategies.

Points for this item:

`lazy-vs-eager-exp.rkt` (3 points)

4. Write a function named `curry`[1], which takes a function f (that takes two arguments) as an argument, and returns a curried version of f. For example `(((curry remainder) 7) 3)` should produce the same value as `(remainder 7 3)`. Save it as `curry.rkt`. You may refer to Van Roy and Haridi's book or other sources for information about currying. Give one example where currying might be useful.

Points for this item:

`curry.rkt` (2 points)

Usefulness of currying (1 point)

5. Assume lexical scope and a linked-list environment. The environment model of evaluation discussed in class does not specify what happens when the same identifier is used in multiple definitions within the same block. Here is one possible reasonable way to define the semantics of `define`:

1. To interpret a `define` statement in an environment e, simply prepend the new binding to e to create a new environment e' for the next expression to be evaluated in, without checking if a binding with the same identifier already exists in e.
2. Since the environment is scanned linearly when looking up the value associated with an identifier in a block b, definitions in b take priority over definitions in blocks enclosing b.

This is known as *name masking* or *shadowing*[2]. What are its advantages and disadvantages? Explain one alternative way to define the semantics and comment on its advantages and disadvantages.

Points for this item:

Name masking (3 points)

Alternative semantics (5 points)

[1] Fun fact: the DISCS website, curry.ateneo.net is a pun on the food (because other subdomains of ateneo.net also use food names) and this concept.

[2] Note that this also implies that the "most recently" interpreted `define` statement takes priority over "earlier" `define`s, but this is outside the scope (pun intended) of the question.

6. Look for the documentation for Racket's `let` construct and study it. Explain how to de-sugar `let` into `lambda`. You may demonstrate with pseudo-Racket (i.e. with ellipses and angle brackets as placeholders for actual values). What is the difference between `let` and `define`? Why would you use one over the other?

Points for this item:

De-sugaring (2 points)

`let` vs. `define` (3 points)

7. We want to make this course as good as possible and we'd appreciate your feedback. Answer the following questions to get a free point:

- How hard did you find this assignment? Did it seem unreasonably difficult or time-consuming for a 3-unit course?
- Did you do read the assigned readings? How helpful were they?
- How is the pace of the course so far? Is it too fast, too slow, or just right?
- What should we improve on for this course?
- What are you liking or disliking about the course so far?

Points for this item:

Course feedback (1 point)

Total points for this assignment: 28

To get the final percentage, points will be divided by: 20