To Do

First (load "load-simple-generics") to work with generic procedures. This defines what is needed and defines (generic-dispatcher) to be (simple-generic-dispatcher).

This will be needed for calling make-generic-arithmetic when making a generic arithmetic to install, as in:

(define full-generic-arithmetic
  (let ((g (make-generic-arithmetic generic-dispatcher)))
    (add-to-generic-arithmetic! g numeric-arithmetic)
    (extend-generic-arithmetic! g function-extender)
    (add-to-generic-arithmetic! g
      (symbolic-extender numeric-arithmetic))
    ;; YOU MAY ADD STUFF HERE
    g))

;;; To allow use of assign-handler! do the following:
(define *current-arithmetic* full-generic-arithmetic)

;;; This makes the default top-level arithmetic
;;; the full-generic-arithmetic

;;; You can use assign-handler! to add handlers here.
;;; For example, to enable + to do foo on objects of type
;;; bar and bletch you may write:
;;; (assign-handler! '+ foo bar? bletch?)

(install-arithmetic! full-generic-arithmetic)

;;; You can use your arithmetic here
Exercise 3.5: Infrastructure for Generics (pp. 85–86)

Exercise 3.6: Vectors by Extending a Generic Arithmetic (pp. 86–87)

Next we have to (load "load-trie-generics") to get a new system with the trie definition of generic-dispatcher.

Exercise 3.16: Trie Rules (pp. 111–112)

Exercise 3.17: Gotcha! (pp. 112–113)

Now we want the cached version so we have to (load "load-cached-generics").

Exercise 3.18: Cache Performance (p. 115)