Review of Lab 6
Part I: Parsing
1. What does the MiniScheme expression * parse to?
What does the MiniScheme expression * parse to?

Answer: (var-ref *)
2. What does (+ (* 2 3) 5) parse to?
What does (+ (* 2 3) 5) parse to?

Answer: (app-exp (var-ref +) (app-exp (var-ref *) ((lit-exp 2) (lit-exp 3))) (lit-exp 5))
3. What does (if (< x 10) (* x 2) x) parse to?
What does (if (< x 10) (* x 2) x) parse to?

Answer: (if-exp (app-exp (var-ref <) ((var-ref x) (lit-exp 10)))
               (app-exp (var-ref *) ((var-ref x) (lit-exp 2)))
               (var-ref x))
4. What does (let ([f +] [A 3] [B (* 4 5)]) (f A B)) parse to?
What does (let ([f +] [A 3] [B (* 4 5)]) (f A B)) parse to?

Answer:
(let-exp (f A B)
  ((var-ref +) (lit-exp 3) (app-exp (var-ref *) ((lit-exp 4) (lit-exp 5))))
  (app-exp (var-ref f) ((var-ref A) (var-ref B))))
Part II: Evaluation
So * parses to (var-ref *)

What does * evaluate to?
What does * evaluate to?

(prim-proc *)

How does this happen? * parses to (var-ref *) and we evaluate a var-ref by looking it up in the environment. In init-env all primitive procedures are bound to prim-proc versions of themselves.
So (+ (* 2 3) 5) parses to
(app-exp (var-ref +) ( (app-exp (var-ref *) ((lit-exp 2) (lit-exp 3)))
  (lit-exp 5)))

We know this evaluates to 11. But how does it get evaluated?
How does
(app-exp (var-ref +) ( (app-exp (var-ref *) ((lit-exp 2) (lit-exp 3))) (lit-exp 5)))
get evaluated?

It is an app-exp, so we call apply-proc with evaluated (var-ref +) as the procedure and the list of evaluated arguments.

First argument: we evaluate (app-exp (var-ref *) ((lit-exp 2) (lit-exp 3))) by calling (apply-proc (prim-proc *) (2 3)), which gives 6
Second argument: we evaluate (lit-exp 5) and get 5

So altogether we call (apply-proc (prim-proc +) (6 5)) and this gives 11.
So \((\text{if } (< x 10) (* x 2) x)\) parse to
\[
(\text{if-exp } (\text{app-exp } (\text{var-ref } <) \ ((\text{var-ref } x) \ (\text{lit-exp } 10)))
  (\text{app-exp } (\text{var-ref } *) \ ((\text{var-ref } x) \ (\text{lit-exp } 2)))
  (\text{var-ref } x))
\]

How does it get evaluated in an environment where \(x\) is bound to 12?
How is
(if-exp (app-exp (var-ref <) ((var-ref x) (lit-exp 10)))
  (app-exp (var-ref *) ((var-ref x) (lit-exp 2)))
  (var-ref x))
evaluated in an environment where x is bound to 12?

We first evaluate the condition. It is an app-exp so we call
(app-apply (prim-apply <) (12 10)) (using the fact that x is bound to 12)
This should evaluate to False, so we evaluate the second branch of the
expression, which is the third field of the if-exp, which is (var-ref x). We
evaluate this by looking up x in the environment, which gives 12.
Last question!
So (let ([f +] [A 3] [B (* 4 5)]) (f A B)) parses to

(let-exp (f A B)
  ((var-ref +) (lit-exp 3) (app-exp (var-ref *) ((lit-exp 4) (lit-exp 5))))
  (app-exp (var-ref f) ((var-ref A) (var-ref B))))

How does it get evaluated?
How is this evaluated?

(let-exp (f A B)
  ((var-ref +) (lit-exp 3) (app-exp (var-ref *) ((lit-exp 4) (lit-exp 5))))
  (app-exp (var-ref f) ((var-ref A) (var-ref B))))

First we evaluate the binding list values
  (var-ref +) evaluates to (prim-proc +)
  (lit-exp 3) evaluates to 3
  (app-exp (var-ref *) ((lit-exp 4) (lit-exp 5))) evaluates to 20

So we evaluate the body in an extended environment where (f A B) are bound to ( (prim-proc +) 3 20)

(continued next slide)
That is, we need to evaluate \((\text{app-exp} \ (\text{var-ref} \ f) \ ((\text{var-ref} \ A) \ (\text{var-ref} \ B)))\) in where \((f \ A \ B)\) are bound to \(((\text{prim-proc} \ +) \ 3 \ 20)\)

To do this we evaluate \((\text{var-ref} \ f)\) by looking up \(f\) in this environment and getting \((\text{prim-proc} \ +)\),
we evaluate \((\text{var-ref} \ A)\) by looking up \(A\) and getting 3,
we evaluate \((\text{var-ref} \ B)\) by looking up \(B\) and getting 20

So we call \((\text{apply-proc} \ (\text{prim-proc} \ +) \ (3 \ 20))\) and this gives 23.
That was 8 questions. How many did you get right?