Programming Languages – Interpreter Lab 1

1. Look at the code to handle addition. Add equivalent code to handle subtraction, multiplication, division, and exponentiation. Optional: add square root, absolute value, other math things.

   Test these: `(add 3 4)` `(sub 5 50)` `(mul 2 -10)` `(div 5 1)`

2. Add sub-expressions to the interpreter. In other words, `(add 1 (add 2 3))` doesn't work. Why not? Make it work.

   Test these:

   ```
   (add (add 3 4) (sub 4 5))
   (sub (div 20 4) (mul 3 (add 1 1)))
   ```

3. Add variable definitions to Mini-Racket by writing code to handle `(define var expression)` and uncommenting the `symbol?` line inside of `mini-eval` that calls `lookup-variable-value`.

   Test these:

   ```
   (define x 4)
   x                                ; should return 4
   (define y (add 4 9))
   y                                ; should return 13
   (define z (mul 2 x))
   z                                ; should return 8
   (add x y)                        ; should work ok
   (add x y z)                      ; why does this crash?  Can you make
                                    ; it not crash?
   ; Make up some more tests with nested add/sub/mul etc.
   ; What happens if you re-define a variable?  Is this good or bad behavior?
   ```

4. Implement a simple conditional statement:

   ```
   (ifzero expr1 expr2 expr3)
   ```

   Semantics: if expr1 evaluates to zero, evaluate and return expr2, otherwise, evaluate and return expr3.

   Add functions `(ifzero? expr)` and `(eval-ifzero expr1 expr2 expr3)`, then uncomment the `ifzero?` line inside `mini-eval`.

   Tests:

   ```
   (ifzero 0 3 4) ==> 3
   (ifzero 1 3 4) ==> 4

   (define a 0)
   (define b 2)
   (define c 4)
   (ifzero a b c) ==> 2

   (ifzero (add -1 1) (sub a b) (mul b c)) ==> -2
   ```

Hash table reference:

```
Make a hash table:                                    (define ht (make-hash))
Put a key-value pair into a hash table:               (hash-set! ht key value)
Test if a key is in a hash table:                     (hash-has-key? ht key)
Retrieve a value from a hash table based on the key: (hash-ref ht key)

Examples:

(define ht1 (make-hash))
(hash-set! ht1 'foo 5)
(hash-set! ht1 'bar 10)
(hash-has-key? ht1 'foo) ==> #t
(hash-has-key? ht1 'bar) ==> #t
(hash-has-key? ht1 'baz) ==> #f
(hash-ref ht1 'foo) ==> 5
(hash-ref ht1 'baz) ==> *error*
```

Challenges

- Add more math functions, like absolute value, square root, etc.
- Add the ability to prevent variables from being redefined.
- Add the ability to "undefine" a variable. Use hash-remove!.
- Add an alternate if statement that takes four arguments, called “ifequal”. Semantics: If the first two args are equal, evaluate and return the third arg, else the fourth.
- Add a let statement.