CS 510 – The IMPLICIT-REFS and EXPLICIT-REFS Languages
Exercise Booklet 2

Note: in the exercises below we use a box to indicate a breakpoint. When execution reaches an expression inside a box, it stops.

1 IMPLICIT-REFS

Exercise 1

Depict the environment and store that is extant at the breakpoint.

```
let a = 2
in let b =3
   in begin
       set a = b;
       a
      end
```

Exercise 2

Depict the environment and store that is extant at the breakpoint.

```
let a = 2
in let b = proc(x) {
      begin
       set a = x;
       a
      end
   }
in (b 3)
```

Exercise 3

Depict the environment and store that is extant at the breakpoint.

```
let a = 2
in let b = proc(x) {
      begin
       set a = x;
       a
      end
```

```
   }
in (b 3) + (b 4)
```

Exercise 4

In order to model mutable variables we have introduced a store into our runtime system. Why couldn't we just have used an environment and have an assignment such as `set x = e` simply update the expressed value of `x` with that resulting from evaluating `e`? The answer has to do with *sharing*: we want to be able to share portions of memory for efficiency reasons.

Write a program in which two different closures share the same address in memory (this situation is known as *aliasing*. Hint: use a closure to create a different reference to the same memory address.

Exercise 5

The following program illustrates the use of a technique called "backpatching". What does this program evaluate to?

```
let f = proc (x) { x }
in begin
     set f = proc(x) { if zero?(x) then 0 else x + (f (x-1)) };
     (f 5)
   end
```

Exercise 6

Use backpatching to define factorial and then compute the factorial of 5.

2 EXPLICIT-REFS

Exercise 7

What is the result of executing the following program?

```
let r = newref(10)
in r
```

Exercise 8

Does the following program produce an error when executed?

```
let p2function = newref(proc (x) { x + 1 })
in p2function
```

Exercise 9

Consider the following program.

```
let g =
     let counter = newref(0)
     in proc (d) {
          begin
            setref(counter , deref(counter) + 1);
            deref(counter)
          end }
in (g 3)-(g 4)
```

1. What is the result of executing the following program?
2. Draw the environment and store that results just after the program finishes execution

Exercise 10

A memory address is said to be *reachable* if it can be accessed through a series of dereferences starting from some variable in the environment. An unreachable memory address is known as *garbage*. Write a program that produces a memory address in the store which is unreachable.