**CS1101: Lab 7 – Using Assignment Operators to Create Memory Devices**

In any system involving microprocessors, an important component to consider is the memory device. These devices are responsible for storing all data, programs, and operating parameters of the system. Memory devices range in sizes (amount and width of data that can be stored) as well as types (SRAM, DRAM, etc). In this lab, we will be creating simple memory devices in order to gain practice using assignment operators in Scheme.

**Lab Motivation and Goals**

* Gain experience using the set! operator
* Decide when set! or set-structure! is necessary
* Write functions that use set! and set-structure! properly

**Exercises**

An electrical engineer is creating a microprocessor system with several memory devices. He is having trouble organizing data in the memory devices and wants to create a series of functions for manipulating the data in various ways. The data definition for a piece of data as well as a memory device is provided below. The engineer eventually decides to use pointers to reference pieces of data in the memory devices. The data definition for a pointer is also provided below. Copy these into your Lab 6 file before continuing.

;; A datum is

;; (make-datum number number)

(define-struct datum (value address))

;; A memory-device is either

;; empty, or

;; (cons datum memory-device)

;; A pointer is

;; (make-pointer symbol datum)

(define-struct pointer (name data))

The engineer decides that his system would be optimal with two memory devices. The first is a RAM, and the second is an EEPROM. The definitions for the memory devices are provided below. Copy these into your Lab 6 file as well.

 (define EEPROM empty)

 (define RAM empty)

1. To begin, functions should be implemented to search the memory device for a particular piece of data
	1. Write a function **find-value** that consumes a particular value (number) and a memory-device and returns the first datum in the memory-device with the given value.
	2. Write a function **find-by-address** that consumes an address (number) and a memory-device and returns the first datum in the memory-device with the given address.
2. In order to make the first functions created useful, a function should also be created to add datum to the memory device.
	1. Write a function **add-to-EEPROM** that consumes a datum and adds it to the EEPROM. It is important that data with the same address are not added to the EEPROM; otherwise, data corruption will occur. The contract for the function should be as follows:

add-to-EEPROM : datum -> void

* 1. Create two pieces of data and add them to the EEPROM using the function created in part a.
	2. Use a find function from Exercise 1 to search for a piece of data that you added to the EEPROM. What result do you expect to get? Is this what happens?
	3. Since the microprocessor has two memory devices, it is tempting to write a general function **add-to-memory**, which takes a datum and a memory-device and adds the datum to the given memory-device. This function would make it possible to add datum to any memory device and not only the EEPROM. Create the function **add-to-memory** and explain what happens when you run the program.
	4. Write down what you have observed regarding set!. Why is it possible to use one find function for multiple memory-devices, but not possible to do the same for adding datum to memory-devices?
1. Write a function **delete-from-EEPROM** that takes in a datum and deletes it from the EEPROM.

**Everyone should be able to finish up to this point**

1. At times it is beneficial to have pointers to various pieces of data in a memory device.
	1. Create several pieces of datum and add these to the EEPROM using the function created in Exercise 2.
	2. Create pointers to the datum created in part a.
	3. Write a function **add-value** that consumes a value (number), address (number), and a memory-device. It produces void; however, the datum with the address input to the function will have its value increased by the value input to the function. When completing this problem, use set-structure! to update the datum.
	4. Write a function **add-value-2** that consumes a value (number) and an address (number) and produces void. It should update the datum in EEPROM with the given address. The datum should have its value increased by the value input to the function. In this problem, use set! rather than set-structure! to complete the task.
	5. Use the function **add-value** to add 2 to a datum referenced by one of your pointers from part b. Does this result in the value being changed in both the EEPROM and the pointer?
	6. Perform the same operation with the function **add-value-2**. Does this result in the value being changed in both the EEPROM and the pointer?
	7. Comment on the effects of set! and set-structure!. When should each be used?
2. Write a function **sort** that properly organizes the EEPROM such that the datum with the lowest address is first in the EEPROM and the datum with the highest address is last.