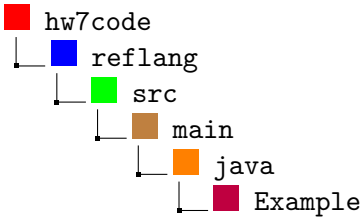
Homework: RefLang

Learning Objectives:

1. RefLang programming
2. Understand and expand RefLang interpreter

Instructions:

- Total points: 46 pt
- Early deadline: April 8 (Friday) at 11:59 PM; Regular deadline: April 10 (Sunday) at 11:59 PM
- Download hw7code.zip from Canvas
- Set up the programming project following the instructions in the tutorial from hw2 (similar steps)
- How to submit:
  - Please submit your solutions in one zip file with all the source code files (just zip the complete project's folder).
  - Write your solutions to question 1 in a file named "hw7.scm" and store it under your code directory.



  - Submit the zip file to Canvas under Assignments, Homework 7.

Questions:

1. (17 pt) [RefLang Programming] Write your solutions to this question in a hw7.scm file and store it under your code directory mentioned in the instructions.

   (a) (2 pt) Write a Reflang program that uses aliases and provide the transcript/output of running the programs.

   (b) (15 pt) Given the definition of the linked list below:

```
(define pairNode (lambda (fst snd) (lambda (op) (if op fst snd))))
(define node (lambda (x) (pairNode x (ref (list)))))
(define getFst (lambda (p) (p #t)))
(define getSnd (lambda (p) (p #f)))
(define head (node 1))
```

i. (7 pt) Write a RefLang function that returns the node of value x (which we call `node x`) in a linked list. If there are multiple such nodes, return the first node whose value is x. If there are no such nodes found, return (list). The function `find` will take two parameters, the head of the linked list *head* and the value *x*: `(define find (lambda (head x) (...)`

ii. (8 pt) Write a RefLang program that inserts a node `ele` in a linked list after `node x`. If there are no such node found, return `head`.

2. (5 pt) [Free detection] In Reflang an expression like the following creates two usage of "free" for variables: (`class` and `course`). Once the `Let` statement has been finished executing, two variales that are allocated will be freed.

```
(let ((class (ref 342))) (let ((course class)) (deref course)))
```

Modify the Reflang interpreter so that it prints a message when a variable is freed. An example appears below.

$ (let ((class (ref 342))) (let ((course class)) (deref course)))
Variable Freed : name class ref value loc:0
Variable Freed : name course ref value loc:0
342
$ (define x (ref 23))
$ (free x)
Variable Freed :name x ref value loc:1

3. (5 pt) [Understanding the memory deallocation] In the Reference language or RefLang, we can free a location already been freed. For instance, first, we declare a variable x as below,
$ (define x (ref 1))
Now, if we free the location x, we can do that by,
$(free x)
If we again try to free that same location, we can do that in RefLang,
$(free x)

Now, change the behavior of the RefLang in such a way that such incidents will be reported. For instance, if we use the "free" operation on the same location more than once, it will throw an error.

$ (define x (ref 1))
$(free x)
$(free x) Wrong memory location (loc:0). Already freed.

4. (5 pt) [Typed heap] This question is about a semantic variation of the heap abstraction. Typed heap enforces the property that in a memory location, only values of compatible types can be stored. Two types are compatible if one is the subtype of the other. Extend the Reflang interpreter to support typed heap. Modify the semantics of assign expression assignexp to check that upon setting the value of a location, the type of the new value is compatible with the type of the old value already stored in that location; otherwise, raise a dynamic error.
[Hint: in Java you can use `isAssignableFrom` to check for compatibility of types.]
The following example scripts illustrates the semantics of typed heap.
$ (let ((x (ref 0))) (set! x 12))
12
$ (let ((x (ref 0))) (set! x #t))

Assigning a value of an incompatible type to the location in (set ! x #t)
$ (let ((x (ref (ref 0)))) (set! x (ref (ref (ref 5)))))
loc:4

5. (14 pt) [Adding new features to RefLang] Add a new predicate, ref# to check if two values are of same type or not. The following example scripts illustrate the semantics of this predicate:
$ (define loc (ref 3))
$ (ref# loc (ref 2))
#t
$ (ref# (let ((c (ref 342))) (set! c (ref 541))) 2)
#f
$ (ref# 1 2)
#t