Homework: RefLang Solution

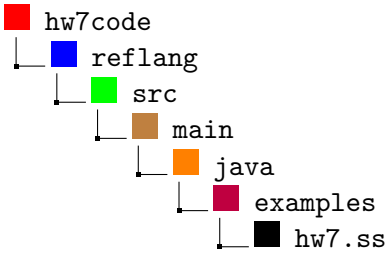
for this also

Learning Objectives:

1. RefLang programming
2. Understand and expand RefLang interpreter

Instructions:

- Total points: 48 + 8 pt
- Early deadline: April 5 (Wed) at 11:59 PM; Regular deadline: April 7 (Fri) at 11:59 PM (you can continue working on the homework till TA starts to grade the homework).
- Download hw7code.zip from Canvas
- Set up the programming project following the instructions in the tutorial from Homework 2 (similar steps).
- How to submit:
  - Write your solutions to question 1 in a file named "hw7.ss" and store it under your code directory.



  - Please submit your solutions in one zip file with all the source code files (just zip the complete project's folder).
  - Submit the zip file to Canvas under Assignments, Homework 7.

Questions:

1. (18 pt) [RefLang Programming] Write your solutions to this question in a `hw7.ss` file and store as mentioned in the instructions.

   (a) (3 pt) Write three RefLang programs that use aliases. Recall that an alias is created when two variables refer to the same memory location.

   (b) (15 pt) For the following definition of linked list

```
$ (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)))
```

i. (7 pt) Write a RefLang function, `find`, that returns the node of value `x` (which we call `node x`) in a linked list. In case of multiple matches, return the first node whose value is `x`. Return empty list if no such node is found. The function `find` will take two parameters, the head of the linked list `head` and the value `x`. See the example interactions below:

```
$ (define head (node 2))
$ (find head 2)
(lambda ( op ) (if op fst snd))
$ (getFst (find head 2))
2
$ (find head 1)
()
```

ii. (8 pt) Write a RefLang function, `insert`, that inserts a node, `element`, in a linked list after the node `x`. If `find` function returns empty list, then return `head`. See the example interactions below:

```
$ (define h (node 2))
$ (insert h 2 (node 3))
()
$ (getFst (find h 3))
3
$ (insert h 4 (node 3))
(lambda ( op ) (if op fst snd))
$ (getFst (insert h 4 (node 3)))
2
```

Solution:

(a)

```
1 $ (define alias (ref 0))
2 $ (let ((x alias)) x)
3 $ (let ((y alias)) (deref y))
```

(b)

i.

```
1 (define find
2   (lambda (head x)
3     (if (= x (getFst head))  head
4        (if (null? (deref (getSnd head)))  (list)
5        (find (deref (getSnd head)) x))
6     )
7   )
8 )
```

ii.

```
1 (define insert
2     (lambda (head x element)
3   (if (null? (find head x) ) head
4      (if (= (getFst head) x)
5       (let ((fi (deref (getSnd (find head x)))) (temp (set! (getSnd head)
            element) )) (set! (getSnd element) fi ))
```

```
        (insert (deref (getSnd head)) x element)
            )
        )
    )
)
```

2. (10 pt) [Aliasing] In RefLang, an expression like `(let ((class (ref 342)))(let ((course class))(deref course)))` creates two aliases (class and course) to the memory cell storing the value 342. Modify the RefLang interpreter, so it prints a message whenever an alias is created. See the example interaction below:

```
$ (let ((class (ref 342))) (let ((course class)) (deref course)))
Alias created:: Name -> class ref value -> loc:0
Alias created:: Name -> course ref value -> loc:0
342
$ (define year (ref 2023))
Alias created:: Name -> year ref value -> loc:1
$ (deref year)
2023
```

Solution: Found in hw7code-sol.zip.

Evaluator.java

```
1   @Override
2   public Value visit(LetExp e, Env env) { // New for varlang.
3     List<String> names = e.names();
4     List<Exp> value_exps = e.value_exps();
5     List<Value> values = new ArrayList<Value>(value_exps.size());
6
7     for (Exp exp : value_exps)
8       values.add((Value) exp.accept(this, env));
9
10    Env new_env = env;
11    for (int index = 0; index < names.size(); index++) {
12      new_env = new ExtendEnv(new_env, names.get(index), values.get(index));
13
14      if (values.get(index) instanceof RefVal) {
15        Value.RefVal loc = (Value.RefVal) values.get(index);
16        if (!(heap.deref(loc) == null)) {
17          System.out.print("Alias created:: " + "Name -> " + names.get(index) + " ref
                value -> "
18              + values.get(index).tostring() + "\n");
19        }
20      }
21
22    }
23    return (Value) e.body().accept(this, new_env);
24  }
25
26  @Override
27  public Value visit(DefineDecl e, Env env) { // New for definelang.
28    String name = e.name();
29    Exp value_exp = e.value_exp();
30    Value value = (Value) value_exp.accept(this, env);
31    ((GlobalEnv) initEnv).extend(name, value);
```

```
32      if (value instanceof RefVal) { // For Memory leak detection
33        System.out.print("Alias created:: " + "Name -> " + name + " ref value -> " +
              value.tostring() + "\n");
34        ...
35      }
36      return new Value.UnitVal();
37    }
```

3. (20 pt) [Reference Arithmetic] In RefLang, language arithmetic operations are not permitted on a reference value.

   (a) (6 pt) Modify the semantics of dereference expression such that it can dereference locations specified as explicit natural numbers. See the example interaction below:

```
1 $ (let ((class (ref 342))) (deref 0))
2 342
```

   (b) (6 pt) Modify the semantics of assignment expression such that it can assign locations specified as explicit natural numbers. See the example interaction below:

```
1 $ (let ((class (ref 342))) (set! 0 541))
2 541
```

   (c) (8 pt) Modify the semantics of addition and subtraction expressions such that the addition and subtraction are permitted on reference values. In this resulting language, adding one or more numeric values to a reference value will result in a reference value. See the example interaction below:

```
1 $ (define a (ref 342))
2 loc:0
3 $ (+ 1 a)
4 loc:1
5 $ (+ 1 a 1)
6 loc:2
7 $ (deref (+ 0 a))
8 342
9 $ (deref (+ 1 a))
10 Null pointer at loc:1
```

   In this language, subtracting one or more numeric values from a reference value will result in a reference value. See the example interaction below:

```
1 $ (define a (ref 342))
2 loc:0
3 $ (+ 1 a)
4 loc:1
5 $ (- (+ 1 a) 1)
6 loc:0
```

Solution:

(a) Found in hw7code-sol.zip. You will need to modify the following files:

(3 pt) Evaluator

```
1  public class Evaluator implements Visitor<Value> {
2    ...
3    @Override
4    public Value visit(DerefExp e, Env env) { // New for reflang
5      Exp loc_exp = e.loc_exp();
6      if (loc_exp.accept(this, env) instanceof Value.NumVal) { // For natural
           number dereference
7        Value.NumVal natNum = (Value.NumVal) loc_exp.accept(this, env);
8        return heap.derefInt(natNum);
9      } else {
10       Value.RefVal loc = (Value.RefVal) loc_exp.accept(this, env);
11         return heap.deref(loc);
12     }
13   }
14   ...
15 }
```

(3 pt) Heap

```
1  public interface Heap {
2    ...
3    Value derefInt(Value.NumVal natNum); // For natural number dereference
4    ...
5    static public class Heap16Bit implements Heap {
6      ...
7     public Value derefInt(Value.NumVal natNum) { // For natural number dereference
8          try {
9              for (int i = 0; i < index; i++) {
10                 if (natNum.v() == new Value.NumVal(i).v()) {
11                     return _rep[i];
12                 }
13             }
14             return new Value.DynamicError("No reference found at " + natNum.v());
15         } catch (ArrayIndexOutOfBoundsException e) {
16             return new Value.DynamicError("Segmentation fault at access " +
                   natNum.v());
17         }
18     }
19     ...
20   }
21   ...
22 }
```

(b) Found in hw7code-sol.zip. You will need to modify the following files:

(3 pt) Evaluator

```
1  public class Evaluator implements Visitor<Value> {
2    ...
3    @Override
4      public Value visit(AssignExp e, Env env) { // New for reflang
5        ...
6        Value rhs_val = (Value) rhs.accept(this, env);
```

```
7       if (lhs.accept(this, env) instanceof Value.NumVal){ // For natural number
            assignment
8         Value.NumVal natNum = (Value.NumVal) lhs.accept(this, env);
9         if (heap.derefInt(natNum) != null && rhs_val.getClass().isAssignableFrom(
              heap.derefInt(natNum).getClass())) {
10          Value assign_val = heap.setrefInt(natNum, rhs_val);
11          return assign_val;
12        } else {
13          return new Value.DynamicError("Assigning a value of incompatible type
                to the location in " + ts.visit(e, null));
14        }
15      } else {
16        Value.RefVal loc = (Value.RefVal) lhs.accept(this, env);
17        Value assign_val = heap.setref(loc, rhs_val);
18        return assign_val;
19      }
20    }
21  ...
22 }
```

(3 pt) Heap

```
1  public interface Heap {
2    ...
3    public Value setrefInt(Value.NumVal loc, Value value); // For natural number
         assignment
4    ...
5    static public class Heap16Bit implements Heap {
6      ...
7      public Value setrefInt(Value.NumVal loc, Value value) { // For natural number
           assignment
8        try {
9          for (int i = 0; i < index; i++) {
10           if (loc.v() == new Value.NumVal(i).v()) {
11             _rep[i] = value;
12             return _rep[i];
13           }
14         }
15         return new Value.DynamicError("No reference found at " + loc.v());
16       } catch (ArrayIndexOutOfBoundsException e) {
17         return new Value.DynamicError("Segmentation fault at access " + loc);
18       }
19     }
20     ...
21   }
22   ...
23 }
```

(c) Found in hw7code-sol.zip. Each expression is 4 pt each.

(8 pt) Evaluator

```
1 public class Evaluator implements Visitor<Value> {
2   ...
3   @Override
4   public Value visit(AddExp e, Env env) {
5     List<Exp> operands = e.all();
6     double result = 0;
```

```
7     boolean isRef = false;  // For RefVal implementation
8     for(Exp exp: operands) {
9       Object avoid_double_heap = exp.accept(this, env);
10      if (avoid_double_heap instanceof Value.NumVal) { // For RefVal
            implementation
11        NumVal intermediate = (NumVal) avoid_double_heap; // Dynamic type-
              checking
12        result += intermediate.v(); //Semantics of AddExp in terms of the target
              language.
13      } else {
14        isRef = true;
15        RefVal intermediate = (RefVal) avoid_double_heap; // Dynamic type-
              checking
16        result += intermediate.loc();
17      }
18    }
19    if (isRef) {
20      return new Value.RefVal((int) result);
21    } else {
22      return new NumVal(result);
23    }
24  }
25
26  ...
27  @Override
28  public Value visit(SubExp e, Env env) {
29    List<Exp> operands = e.all();
30    double result;  // For RefVal implementation
31    boolean isRef = false;
32    Object avoid_double_heap = operands.get(0).accept(this, env);
33    if (avoid_double_heap instanceof Value.NumVal) {
34      NumVal lVal = (NumVal) avoid_double_heap;
35      result = lVal.v();
36    } else {
37      isRef = true;
38      result = ((Value.RefVal) avoid_double_heap).loc();
39    }
40    for(int i=1; i<operands.size(); i++) {
41      Object avoid_double_heap_op = operands.get(i).accept(this, env);
42      if (avoid_double_heap_op instanceof Value.NumVal) {
43        NumVal rVal = (NumVal) avoid_double_heap_op;
44        result = result - rVal.v();
45      } else {
46        isRef = true;
47        result -= ((Value.RefVal) avoid_double_heap_op).loc();
48      }
49    }
50    if (isRef) {
51      return new Value.RefVal((int) result);
52    } else {
53      return new NumVal(result);
54    }
55  }
56  ...
57 }
```

4. (8 pt) (extra credit) [Memory leak detection]

   (a) (3 pt) Modify RefLang evaluator to support memory leak detection for global variables. When we redefine a global variable, the previous reference value is no longer accessible, leading to the memory leak. See the example interaction below:

```
$ (define c (ref 342))
loc: 0
$ (deref c)
342
$ (define c (ref 541))
loc:1
Info: Memory Leak Detected
```

   (b) (5 pt) Improve your memory leak detector to support the revocation of the leaked memory: once the leaked memory is detected, you can set it to `null` for future reuse. See the example interaction below:

```
$ (define c (ref 342))
loc: 0
$ (deref c)
342
$ (define c (ref 541))
loc:1
Info: Memory Leak Detected
$ (define b (ref 0))
Info: Used Recovered Memory
loc: 0
```

Solution:

   (a) Found in hw7code-sol.zip. You will need to modify the Evaluator file.

```
public class Evaluator implements Visitor<Value> {
  ...
  static Map<String, Value> heapMemory = new HashMap<String, Value>(); // For
      Memory leak detection
  ...
  @Override
  public Value visit(DefineDecl e, Env env) { // New for definelang.
    String name = e.name();
    Exp value_exp = e.value_exp();
    Value value = (Value) value_exp.accept(this, env);
    ((GlobalEnv) initEnv).extend(name, value);
    if (value instanceof RefVal) { // For Memory leak detection
      System.out.print(value.tostring() + "\n");
      if (heapMemory.containsKey(name)) {
        System.out.print("Info : Memory Leak Detected" + "\n");
        heapMemory.put(name, value);
      } else {
        heapMemory.put(name, value);
      }
    }
    return new Value.UnitVal();
  }
  ...
```

```
23 }
```

(b) Found in hw7code-sol.zip. You will need to modify the following files:
(2 pt) Evaluator

```
1  public class Evaluator implements Visitor<Value> {
2    ...
3    static Map<String, Value> heapMemory = new HashMap<String, Value>(); // For
         Memory leak detection
4    ...
5    @Override
6    public Value visit(DefineDecl e, Env env) { // New for definelang.
7      String name = e.name();
8      Exp value_exp = e.value_exp();
9      Value value = (Value) value_exp.accept(this, env);
10     ((GlobalEnv) initEnv).extend(name, value);
11     if (value instanceof RefVal) { // For Memory leak detection
12       System.out.print(value.tostring() + "\n");
13       if (heapMemory.containsKey(name)) {
14         System.out.print("Info : Memory Leak Detected" + "\n");
15         // For improving memory leak detection
16         heap.free((RefVal) heapMemory.get(name));
17         heapMemory.remove(name);
18         heapMemory.put(name, value);
19       } else {
20         heapMemory.put(name, value);
21       }
22     }
23     return new Value.UnitVal();
24   }
25   ...
26 }
```

(3 pt) Heap

```
1  public interface Heap {
2    ...
3    static List<Integer> heapRemoved = new ArrayList<Integer>();  // For improving
         memory leak detection
4    ...
5    static public class Heap16Bit implements Heap {
6      ...
7      public Value ref(Value value) {
8          if (index >= HEAP_SIZE)
9              return new Value.DynamicError("Out of memory error");
10         if (heapRemoved.size() > 0) { // For improving memory leak detection
11             System.out.print("Info: Used Recovered Memory\n");
12             int old_loc = heapRemoved.remove(0);
13             Value.RefVal new_loc = new Value.RefVal(old_loc);
14             _rep[old_loc] = value;
15             return new_loc;
16         } else {
17             Value.RefVal new_loc = new Value.RefVal(index);
18             _rep[index++] = value;
19             return new_loc;
20         }
21     }
```

```
22      ...
23      public Value free(Value.RefVal loc) {
24          try {
25              if (!heapRemoved.contains(loc.loc())) { // For improving memory leak
                    detection
26                  heapRemoved.add(loc.loc());
27              }
28              _rep[loc.loc()] = null;
29              return loc;
30          } catch (ArrayIndexOutOfBoundsException e) {
31              return new Value.DynamicError("Segmentation fault at access " + loc);
32          }
33      }
34    ...
35    }
36  ...
37  }
```