Homework: RefLang

Instructions:

- Total points: 70 pt
  item Early deadline: Oct 31 (Wed) 2018 at 6:00 PM; Regular deadline: Nov 2 (Fri) 2018 at 6:00 PM (or till TAs start grading the homework)
- 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 5 in a HW7.scm file and store it under your code directory.
  - Submit the zip file to Canvas under Assignments, Homework 7.
- In this homework, we will implement the interpreter for Reflang. **Here are all the changes that are required**.
  - Extend the set of values in Value.java to add RefVal which stores the location.(Question 1)
  - Implement memory in form of an array. You need to create a file Heap.java (Question 2)
  - Implement ASTs for required expressions in AST.java.(Question 3 a)
  - Extend Formatter for these expressions.(Question 3 b)
  - Implement semantics of these expressions in Evaluator.java(Question 4)
  - Extend the grammar for these newly added expressions.(Question 4)
  - Test your Reflang for expressions developed in Reflang.(Question 5)

Questions:

1. (5 pt) First, we will add a new kind of value to the set of values. This new kind of value will represent reference values in our language. You can do that by:
   - Adding a new Java class, RefVal, to the interface Value.
   - Internally this class will maintain an integer index, loc, and
   - RefVal class must provide methods to access this loc.
   - The string representation of a RefVal (in method toString) will be created by prepending "loc:" to the value of loc.

   Sol

```
static class RefVal implements Value {
  private int _loc = -1;
  public RefVal(int loc) { _loc = loc; }
  public String tostring() {
    return "loc:" + this._loc;
  }
  public int loc() { return _loc; }
}
```

2. (10 pt) Design and implement Heap, a new abstraction representing area in the memory reserved for dynamic memory allocation.

   (a) Implement Heap as a Java interface named Heap with four methods ref, deref, setref, and free.
   - The return type of all four methods is Value.
   - The method ref takes a single parameter of type Value.
   - The method deref takes a single parameter of type Value.RefVal from Question 1.
   - The method setref takes a two parameters of type Value. First parameter is of type RefVal while second parameter is Value.
   - The method free takes a single parameter of type Value.RefVal from Question 1.

   (b) Implement a 16 bit heap as a Java class Heap16Bit inside the interface Heap.
   - The class Heap16Bit must implement the interface Heap, and thus provide implementation of each method ref, deref, setref, and free inside the interface.
   - The class would model memory as an array named _rep of type Value[]
   - The method ref
     - takes a single parameter val, of type Value
     - allocates memory and stores val in allocated memory at location l
     - returns a RefVal containing location l.
   - The method deref
     - takes a single parameter loc, of type RefVal
     - returns value stored at location l, where l is stored in loc.
   - The method setref
     - takes two parameters
     - first parameter loc, is of type RefVal which encapsulates location l
     - second parameter val, is of type Value
     - this method replaces the value stored at l with val
     - returns val
   - The method free
     - takes a single parameter loc, of type RefVal which encapsulates location l
     - deallocates the memory location l from _rep.
     - returns loc

   Sol

```
public interface Heap {

  Value ref (Value value);

  Value deref (Value.RefVal loc);

  Value setref (Value.RefVal loc, Value value);

  Value free (Value.RefVal value);

  static public class Heap16Bit implements Heap {
    static final int HEAP_SIZE = 65_536;

    Value[] _rep = new Value[HEAP_SIZE];
    int index = 0;

    public Value ref (Value value) {
      if(index >= HEAP_SIZE)
      return new Value.DynamicError("Out of memory error");
      Value.RefVal new_loc = new Value.RefVal(index);
      _rep[index++] = value;
      return new_loc;
    }

    public Value deref (Value.RefVal loc) {
      try {
        return _rep[loc.loc()];
      } catch (ArrayIndexOutOfBoundsException e) {
        return new Value.DynamicError("Segmentation fault at access " + loc);
      }
    }

    public Value setref (Value.RefVal loc, Value value) {
      try {
        return _rep[loc.loc()] = value;
      } catch (ArrayIndexOutOfBoundsException e) {
        return new Value.DynamicError("Segmentation fault at access " + loc);
      }
    }

    public Value free (Value.RefVal loc) {
      try {
        _rep[loc.loc()] = null;
        return loc;
      } catch (ArrayIndexOutOfBoundsException e) {
        return new Value.DynamicError("Segmentation fault at access " + loc);
      }
    }

    public Heap16Bit(){}
  }
```

```
}
```

3. (10 pt) Question 1 and Question 2 helped you creating the RefVal and Heap representation. Now we would be creating the AST node for the expressions.

   (a) Extend the AST.java and add the representation of the following nodes.
   - refexp
   - derefexp
   - assignexp
   - freeexp

   (b) Extend the Formatter for these new AST nodes in a manner consistent with existing AST nodes.

   Sol

   (a) AST.java

```
public static class RefExp extends Exp {
  private Exp _value_exp;

  public RefExp(Exp value_exp) {
    _value_exp = value_exp;
  }

  public Object accept(Visitor visitor, Env env) {
    return visitor.visit(this, env);
  }

  public Exp value_exp() { return _value_exp; }

}

public static class DerefExp extends Exp {
  private Exp _loc_exp;

  public DerefExp(Exp loc_exp) {
    _loc_exp = loc_exp;
  }

  public Object accept(Visitor visitor, Env env) {
    return visitor.visit(this, env);
  }

  public Exp loc_exp() { return _loc_exp; }

}

public static class AssignExp extends Exp {
  private Exp _lhs_exp;
  private Exp _rhs_exp;
```

```
    public AssignExp(Exp lhs_exp, Exp rhs_exp) {
      _lhs_exp = lhs_exp;
      _rhs_exp = rhs_exp;
    }

    public Object accept(Visitor visitor, Env env) {
      return visitor.visit(this, env);
    }

    public Exp lhs_exp() { return _lhs_exp; }
    public Exp rhs_exp() { return _rhs_exp; }

  }

  public static class FreeExp extends Exp {
    private Exp _value_exp;

    public FreeExp(Exp value_exp) {
      _value_exp = value_exp;
    }

    public Object accept(Visitor visitor, Env env) {
      return visitor.visit(this, env);
    }

    public Exp value_exp() { return _value_exp; }

  }
```

(b) Printer.java

```
  public String visit(AST.RefExp e, Env env) {
    String result = "(ref␣";
    result += e.value_exp().accept(this, env);
    return result + ")";
  }

  public String visit(AST.DerefExp e, Env env) {
    String result = "(deref␣";
    result += e.loc_exp().accept(this, env);
    return result + ")";
  }

  public String visit(AST.AssignExp e, Env env) {
    String result = "(set!␣";
    result += e.lhs_exp().accept(this, env) + "␣";
    result += e.rhs_exp().accept(this, env);
    return result + ")";
  }

  public String visit(AST.FreeExp e, Env env) {
```

```
    String result = "(free ";
    result += e.value_exp().accept(this, env);
    return result + ")";
  }
```

4. (30 pt) Goal of this question is to understand and implement the semantics of the expressions created for Reflang.

   (a) (5 pt) In Question 2, we have designed an Interface called Heap, which supports 4 different methods. Add a global heap of type Heap16Bit to the interpreter. This object will be the heap used by all expressions in the evaluator.

   (b) (5 pt) Implement visit method for refexp in Evaluator.java according to the semantics of ref expression.

   (c) (5 pt) Implement visit method for derefexp in Evaluator.java according to the semantics of deref expression.

   (d) (5 pt) Implement visit method for assignexp in Evaluator.java according to the semantics of assignexp expression.

   (e) (5 pt) Implement visit method for freeexp in Evaluator.java according to the semantics of freeexp expression specified.

   (f) (5 pt) In order to get your Reflang working, you would be required to extend the grammar file. Extend the grammar file for supporting four new expressions of Reflang.

Sol

(a)

```
public class Evaluator implements Visitor<Value> {
  Heap heap = new Heap16Bit();
}
```

(b)

```
public Value visit(RefExp e, Env env) { // New for reflang.
  Exp value_exp = e.value_exp();
  Value value = (Value) value_exp.accept(this, env);
  return heap.ref(value);
}
```

(c)

```
public Value visit(DerefExp e, Env env) { // New for reflang.
  Exp loc_exp = e.loc_exp();
  Value.RefVal loc = (Value.RefVal) loc_exp.accept(this, env);
  return heap.deref(loc);
}
```

(d)

```
public Value visit(AssignExp e, Env env) { // New for reflang.
  Exp rhs = e.rhs_exp();
  Exp lhs = e.lhs_exp();
  //Note the order of evaluation below.
```

```
    Value rhs_val = (Value) rhs.accept(this, env);
    Value.RefVal loc = (Value.RefVal) lhs.accept(this, env);
    Value assign_val = heap.setref(loc, rhs_val);
    return assign_val;
  }
```

(e)

```
  public Value visit(FreeExp e, Env env) { // New for reflang.
    Exp value_exp = e.value_exp();
    Value.RefVal loc = (Value.RefVal) value_exp.accept(this, env);
    heap.free(loc);
    return new Value.UnitVal();
  }
```

(f) grammar file

```
...
| ref=refexp { $ast = $ref.ast; }
| deref=derefexp { $ast = $deref.ast; }
| assign=assignexp { $ast = $assign.ast; }
| free=freeexp { $ast = $free.ast; }
;

refexp returns [RefExp ast] :
'(' Ref
e=exp
')' { $ast = new RefExp($e.ast); }
;

derefexp returns [DerefExp ast] :
'(' Deref
e=exp
')' { $ast = new DerefExp($e.ast); }
;

assignexp returns [AssignExp ast] :
'(' Assign
e1=exp
e2=exp
')' { $ast = new AssignExp($e1.ast, $e2.ast); }
;

freeexp returns [FreeExp ast] :
'(' Free
e=exp
')' { $ast = new FreeExp($e.ast); }
;
```

5. (15 pt) Goal of this question is to test our implementation and understand the semantics of the Reflang interpreter. Write your solutions to this question in a HW7.scm file and store it under your code directory.

(a) (2 pt) Perform the following operations on your implementation of Reflang and provide transcript in HW7.scm file.
(deref (ref 1))
(free (ref 1))
(let ((loc (ref 1))) (set! loc 2))
(let ((loc (ref 3))) (set! loc (deref loc)))

(b) (2 pt) Write 2 Reflang programs which use aliases and also provide the transcript of running those programs.

(c) (11 pt) In this question you will implement a linked list. In a linked list, one element of the node is reference to another node. Each node will have two fields. First field of the node is a number while second element will be reference to other node, defined as:

$(define pairNode (lambda (fst snd) (lambda (op) (if op fst snd))))

(remember in lambda encoding, we use functions to represent data and operations, here is the similar idea).

i. (2 pt) define the head of the linked list with node 1

ii. (5 pt) write a lambda method 'add', which
- takes two parameters
- first parameter 'head' is head of linked list
- second parameter 'ele' is a node
- the function adds ele at the end of linked list, if successful, the value of the lambda method is ele.

iii. (4 pt) write a 'print' function
- takes node as parameter (representing head of linked list)
- returns a list of numbers present in linked list.

Following transcripts will help you understand the functions more:

$ (add head (node 2))
(lambda (op) (if op fst snd))
$ (add head (node 3))
(lambda (op) (if op fst snd))
$ (print head)
(1 2 3)
$ (add head (node 0))
(lambda (op) (if op fst snd))
$ (add head (node 6))
(lambda (op) (if op fst snd))
$ (print head)
(1 2 3 0 6)

Sol

1.
```
$ (deref (ref 1))
1
$ (free (ref 1))
$ (let ((loc (ref 1))) (set! loc 2))
2
$ $(let ((loc (ref 3))) (set! loc (deref loc)))
3
```

2. (a)
```
$ (define a (ref 0))
loc:0
$ (let ( (x a)) x)
loc:0
```

(b)
```
$ (define alias (free a))
```

(c)
```
$ (let ((loc3 (ref 4))) (let ((loc4 loc3)) (deref loc4)))
4
```

3. (a)
```
(define pairNode (lambda (fst snd) (lambda (op) (if op fst snd))))
(define node (lambda (x) (pairNode x (ref (list)))))
(define head (node 1))
```

(b)
```
(define getFst (lambda (p) (p #t)))
(define getSnd (lambda (p) (p #f)))

(define add
  (lambda (head ele)
    (if (null? (deref (getSnd head)))
        (set! (getSnd head) ele)
      (add (deref (getSnd head)) ele))))
```

(c)
```
(define print
  (lambda (head)
    (if (null? (deref (getSnd head)))
      (cons (getFst head) (list))
      (cons (getFst head)
          (print (deref (getSnd head)))))))
```

or

```
(define print
  (lambda (head)
    (if (null? (deref (getSnd head)))
        (getFst head)
```

```
        (cons (getFst head)
            (print (deref (getSnd head)))))))
```