Homework: VarLang and DefineLang Solutions

Learning Objectives:

1. Get familiar with the concepts of scope, free and bound variables and environment.
2. Write programs in VarLang and DefineLang
3. Understand and extend VarLang interpreter.

Instructions:

1. Total points: 61 pts.
2. Early deadline: Feb 15 (Wed) 11:59 pm, Regular deadline Feb 17 (Fri) 11:59 pm. (you can continue working on the homework till TA starts to grade the homework)
3. Download hw3code.zip from Canvas.
4. Set up the programming project following the instructions in the tutorial from hw2 (similar steps).
5. How to submit:
   - For questions 1–4, you can write your solutions in latex or word and then convert it to PDF; or you can submit a scanned document with legible handwritten solutions. Please provide the solutions in one PDF file.
   - For questions 5–7, 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 and one PDF file to Canvas under Assignments, Homework 3.

Questions:

1. (8 pt) [VarLang programming, scoping] Write VarLang programs:

   (a) (4 pt) Write a VarLang program that contains at least two nested `let` expression that produces the value 342. This program must use all five arithmetic operators $+$, $-$, $*$, $/$, and $\%$.

   (b) (4 pt) Write a VarLang program that evaluates to value 684. The program must contain at least 1 "hole in the scope" of the `let` expressions. Explain how the "hole in the scope" was created in the program.

   Solution:

   (a) (4 pt) (let ((x 100) (y 141) (z 140)) (let ((a 5)) (* (% a 3) (- (+ x y) (/ z 2)))))

   (b) (4 pt) (let ((x 10) (y 16)) (let ((x (* x 70))) (- x y)))
   The expression (let ((x (* x 70))) ...) overrides the outer-scoped definition of x and so the expression (- x y) is a hole in the outer scoped definition of x.

2. (6 pt) [VarLang syntax and semantics, scoping] Compute the values of the following VarLang expressions (return an error if the values cannot be computed). Please show the intermediate steps that lead to the final results, and also show the environment changes at each step. For example, see slide 33 in VarLang lecture slide.

   (a) (3 pt) (let ((x 6) (y 8) (z 11)) (let ((x 7)) (* (+ x y) z)))
   (b) (3 pt) (let ((a 8)) (+ (let ((b 10)) b) (let ((c 3)) (+ c b)) 2 a))

   Solution:

a. Current Expression	**Current Environment**
(let ((x 6) (y 8) (z 11)) (let ((x 7)) (* (+ x y) z)))	Empty
(let ((x 7)) (* (+ x y) z))	$x \mapsto 6 :: y \mapsto 8 :: z \mapsto 11 ::$ Empty
(* (+ x y) z)	$x \mapsto 7 :: y \mapsto 8 :: z \mapsto 11 :: x \mapsto 6 ::$ Empty
165	$x \mapsto 7 :: y \mapsto 8 :: z \mapsto 11 :: x \mapsto 6 ::$ Empty
(let ((x 7)) 165)	$y \mapsto 8 :: z \mapsto 11 :: x \mapsto 6 ::$ Empty
(let ((x 6) (y 8) (z 11)) 165)	Empty
165	Empty

b. Current Expression	**Current Environment**
(let ((a 8)) (+ (let ((b 10)) b) (let ((c 3)) (+ c b)) 2 a))	Empty
(+ (let ((b 10)) b) (let ((c 3)) (+ c b)) 2 a)	$a \mapsto 8 ::$ Empty
b	$b \mapsto 10 :: a \mapsto 8 ::$ Empty
10	$b \mapsto 10 :: a \mapsto 8 ::$ Empty
(+ 10 (let ((c 3)) (+ c b)) 2 a)	$a \mapsto 8 ::$ Empty
(+ c b)	$c \mapsto 3 :: a \mapsto 8 ::$ Empty
(+ 3 Error)	$c \mapsto 3 :: a \mapsto 8 ::$ Empty
(+ 10 (let ((c 3)) (+ 3 Error)) 2 a)	$a \mapsto 8 ::$ Empty
(+ (let ((b 10)) 10) (let ((c 3)) (+ 3 Error)) 2 a)	$a \mapsto 8 ::$ Empty
(let ((a 8)) (+ (let ((b 10)) 10) (let ((c 3)) (+ 3 Error)) 2 a))	Empty
Error	Empty

3. (6 pt) [Free and bound variables] List free and bound variables for the following VarLang expressions:

   (a) (3 pt) (let ((c 7) (d c) (f 3)) (let ((e c) (g 6)) (* c (- f (+ e f) d))))
   (b) (3 pt) (let ((a 6) (l c)) (let ((y g) (k l) (x 2)) (/ x y (* g a))))

   Solution:
   The bound and free variables are in **bold** letter in the following expressions:

   (a)

```
1 (let ((c 7) (d c) (z 3))
2                  B
3       (let ((e c) (g 6))
4                  B
5         (* c (- g (+ e z) d))))
6            B     B     B B  B
```

(b)

```
1 (let ((a 6) (l c))
2                 F
3       (let ((y g) (k l) (x 2))
4                 F     B
5         (/ x y  (* (- g k) (+ a l))
6            B B         F B     B B
7     )
8   )
9 )
```

4. (3 pt) [DefineLang programming] Define breath and height of a rectangle (`b` and `h`) with a value of 15 and 2, respectively. Define a constant `two`, with the value of 2. Using the definitions of `two`, `b` and `h`, define and calculate the `perimeter` of the rectangle. Recall that the formula for perimeter is $2 * (b + h)$.

   Solution:

```
1 (define two 2)
2 (define b 15)
3 (define h 2)
4 (define perimeter (* two (+ b h)))
5 perimeter
```

5. (7 pt) [VarLang interpreter basic] The current semantics of the let expression in the VarLang language allows variable definitions to create a hole in the scope of the outer definition. Modify the semantics of the VarLang programming language so that redefinition of variables present in the outer scope is prohibited and results in a dynamic error.

   See some example VarLang programs below:

   $ (let ((a 1) (b 2)) (let ((a 14)) (+ a b)))
   Error: Creating a hole in the scope for variable a in (let ((a 14.0)) (+ a b))
   $ (let ((c 2) (d 3)) (let ((c (* d 14))) (- c d)))
   Error: Creating a hole in the scope for variable c in (let ((c (* d 14.0))) (- c d))
   $ (let ((a 1) (a 2)) a)
   2
   $ (let ((a 1) (a 2)) (let ((a 1)) a))
   Error: Creating a hole in the scope for variable a in (let ((a 1.0)) a)

   Solution: Found in hw3code-sol.zip. You will need to modify the following files as follows:

(a) (3 pt) Evaluator (Evaluator.java)

```
@Override
public Value visit(LetExp e, Env env){ // New for varlang.
  ...
  List<Value> values = new ArrayList<Value>(value_exps.size());

  // Check if there are a hole in the scope
  for (String name : names) {
    if (env.exists(name)) {
      return new DynamicError("Error: Creating a hole in the scope for variable "
          + name + " in " + ts.visit(e,env));
    }
  }
  ...
}
```

(b) (7 pt) Env (Env.java)

```
public interface Env {
  ...
  boolean exists(String search_var);

  ...
  static public class EmptyEnv implements Env {
    ...
    public boolean exists(String search_var) {
        return false;
    }
  }
  ...
  static public class ExtendEnv implements Env {
    ...
    public synchronized boolean exists(String search_var) {
        if (search_var.equals(_var))
            return true;
        return _saved_env.exists(search_var);
    }
    ...
  }
  ...
  static public class GlobalEnv implements Env {
      ...
      public synchronized boolean exists(String search_var) {
          if (map.containsKey(search_var))
              return true;
          return false;
      }
      ...
  }
  ...
}
```

6. (7 pt) [Environment] Extend the interpreter to support three predefined global constants in the environment. VarLang programs can directly use these variables. Any VarLang programs that try to redefine these variables in the `let` expressions will return an error.

Global constants:

- course = 342
- gravity = 9.80665
- moon = 0.16

See some example VarLang programs below:

$ (let ((x 1)) (+ x gravity))
10.80665
$ (+ course gravity)
351.80665
$ moon
0.16
$ (let ((moon 1)) (* moon moon))
Error: Redefining predefined global constants
$ (define moon 1)
Error: Redefining predefined global constants

Solution: Found in hw3code-sol.zip. You will need to modify Evaluator file as follows:

(7 pt) Evaluator (Evaluator.java)

```
1  //3pt
2  public class Evaluator implements Visitor<Value> {
3    private Env initialEnv() {
4      GlobalEnv initEnv = new GlobalEnv();
5      initEnv.extend("gravity", new NumVal(9.80665));
6      initEnv.extend("course", new NumVal(342));
7      initEnv.extend("moon", new NumVal(0.16));
8      return initEnv;
9    }
10   Env initEnv = initialEnv(); // New for definelang
11   ...
12 // 2pt
13   @Override
14   public Value visit(LetExp e, Env env) { // New for varlang.
15     ...
16     List<Value> values = new ArrayList<Value>(value_exps.size());
17
18     // Stop redefining the global variables
19     if (names.toString().equals("[moon]") || names.toString().equals("[gravity]")
20         || names.toString().equals("[course]")) {
21       return new DynamicError("Error: Redefining predefined global constants");
22     }
23     ...
24   }
25 //2pt
26   ...
27   @Override
28   public Value visit(DefineDecl e, Env env) { // New for definelang.
29     String name = e.name();
30
31     // Stop redefining the global variables
```

```
32      if (name.toString().equals("moon") || name.toString().equals("gravity") || name.
            toString().equals("course")) {
33        System.out.println("Error: Redefining predefined global constants");
34        return new DynamicError("Error: Redefining predefined global constants");
35      }
36      ...
37    }
38    ...
39  }
```

7. (24 pt) [VarLang interpreter advanced] Extend the interpreter: Security is a major concern for any system. Therefore, it is important to ensure that there is no information leak and that the memory de-allocated from a program does not contain data which can be read by malicious programs. To avoid such a breach due to environment storage, we want to augment the VarLang language with a `lete` (encoded let) expression, to encode a value before storing it in the environment, and a `dec` expression to decode it before using it. All values are stored by encrypting them with key, and read by decrypting them with key.

   In the `lete` expression, each variable definition will contain the variable name, followed by two numbers where the first number is the value, and the second number is the encryption key. In the `dec` expression, the first number is the key for decryption, followed by the variable name. Note that currently, we do not require you to check whether the keys used in `lete` and `dec` are the same.

   Extend the VarLang programming language to support these two expressions. Implement an encrypted let (`lete` for let encrypted), which is similar to `let`, but it uses a key and a `dec` expression that is similar to `VarExp`.

   $ (lete ((x 1 2)) x)
   3
   $ (lete ((x 1 a)) x)
   Error: Expected Number
   $ (lete ((1 1 2)) 1)
   Error: Expected Identifier
   $ (lete ((x 1 20)) (dec 20 x))
   1
   $ (lete ((x 1 20)) (dec 10 x))
   11
   $ (lete ((x 1 20)) (dec b x))
   Error: Expected Number
   $ (lete ((x 1 20)) (dec 10 b))
   No binding found for name: b
   $ (lete ((y 8 10) (x 1 2)) (+ x y))
   21
   $ (lete ((y 12 10) (x 1 2)) (+ (dec 10 y) (dec 2 x)))
   13

   Solution: Found in hw3code-sol.zip. You will need to modify the following files:

- (7 pt) Grammar file (VarLang.g)

```
//1pt
v=varexp { $ast = $v.ast; }
...
| le=leteexp { $ast = $le.ast; }
| de=decexp { $ast = $de.ast; }
;
...
//3pt
leteexp  returns [LeteExp ast]
     locals [ArrayList<String> names = new ArrayList<String>(),
     ArrayList<Exp> value_exps = new ArrayList<Exp>(),
     ArrayList<Exp> key_exps = new ArrayList<Exp>()] :
       '(' Lete
       '(' ( '(' id=Identifier e=exp k=exp ')' { $names.add($id.text);
          $value_exps.add($e.ast); $key_exps.add($k.ast); } )+  ')'
       body=exp
       ')' { $ast = new LeteExp($names, $value_exps, $body.ast, $key_exps); }
   ;

//2pt
decexp returns [DecExp ast]:
       '(' Dec
       key=exp
       id=Identifier
       ')' { $ast = new DecExp($key.ast, $id.text); }
   ;
...
// Lexical Specification of this Programming Language
//  - lexical specification rules start with uppercase
...
//1pt
Lete : 'lete' ;
Dec : 'dec' ;
```

- (6 pt) AST file (AST.java)

```
public interface AST {
  ...
//3pt
  public static class LeteExp extends LetExp {
    List<Exp> _key_exps;

    public LeteExp(List<String> names, List<Exp> value_exps, Exp body, List<Exp>
        key_exps) {
      super(names, value_exps, body);
      _key_exps = key_exps;
    }

    public List<Exp> key_exps() {
      return _key_exps;
    }

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

```
  }
//2pt
  public static class DecExp extends VarExp {
    Exp _key;

    public DecExp(Exp key, String varExp) {
      super(varExp);
      _key = key;
    }

    public Exp key() {
      return _key;
    }

    public Object accept(Visitor visitor, Env env) {
      return visitor.visit(this, env);
    }
  }
  ...
//1pt
  public interface Visitor <T> {
    ...
    public T visit(AST.LeteExp e, Env env);
    public T visit(AST.DecExp e, Env env);
  }
}
```

- (3 pt) Printer file (Printer.java)

```
public String visit(AST.LeteExp e, Env env) {
public class Printer {
  ...
  public static class Formatter implements AST.Visitor<String> {
    ...
// 2pt
    public String visit(AST.LeteExp e, Env env) {
      String result = "(lete (";
      List<String> names = e.names();
      List<AST.Exp> value_exps = e.value_exps();
      List<AST.Exp> key_exps = e.key_exps();
      int num_decls = names.size();
      for (int i = 0; i < num_decls; i++) {
        result += " (";
        result += names.get(i) + " ";
        result += value_exps.get(i).accept(this, env) + " ";
        result += key_exps.get(i).accept(this, env) + ")";
      }
      result += ") ";
      result += e.body().accept(this, env) + " ";
      return result + ")";
    }
//1pt
    public String visit(AST.DecExp e, Env env) {
      String result = "(dec ";
      result += e.key().accept(this, env) + " ";
      result += e.name() + " )";
      return result;
```

```
    }
  ...
  }
  ...
}
```

- (8 pt) Evaluator (Evaluator.java)

```
public class Evaluator implements Visitor<Value> {
  ...
//4pt
  public Value visit(LeteExp e, Env env) {
    List<String> names = e.names();
    List<Exp> value_exps = e.value_exps();
    List<Exp> key_exps = e.key_exps();

    List<Value> values = new ArrayList<Value>(value_exps.size());
    List<Value> keys = new ArrayList<Value>(key_exps.size());

    for (Exp exp : value_exps)
      values.add((Value) exp.accept(this, env));
    for (Exp exp : key_exps)
      if (exp instanceof NumExp) {
        keys.add((Value) exp.accept(this, env));
      } else {
        return new DynamicError("Error: Expected Number");
      }

    for (String name : names)
      if (name.equals("<missing Identifier>")) {
        return new DynamicError("Error: Expected Identifier");
      }

    Env new_env = env;
    for (int i = 0; i < names.size(); i++) {
      Value val = values.get(i);
      Value key = keys.get(i);
      if (val instanceof NumVal) {
        val = new NumVal((((NumVal) val).v() + ((NumVal) key).v()));
      }
      new_env = new ExtendEnv(new_env, names.get(i), (val));
    }
    return (Value) e.body().accept(this, new_env);
  }
//4pt
  public Value visit(DecExp e, Env env) {
    Exp key_exp = e.key();
    if (key_exp instanceof NumExp) {
      Value val = env.get(e.name());
      if (val instanceof NumVal) {
        NumVal key = (NumVal) e.key().accept(this, env);
        return new NumVal((((NumVal) val).v()) - key.v());
      }
      return val;
    } else {
      return new DynamicError("Error: Expected Number");
    } } ... }
```