# Homework: FuncLang (Part II)

**Learning Objectives:**

1. Functional programming
2. Understand and expand FuncLang interpreter

## Instructions:

- Total points: 59 pt
- Early deadline: Oct 3 (Wed) 2018 at 6:00 PM; Regular deadline: Oct 5 (Fri) 2018 at 6:00 PM
- We will grade functional programming based on our tests
- Download hw5code.zip from Canvas
- Set up the programming project following the instructions in the tutorial from hw2 (similar steps)
- How to submit:
  - For questions 1–2, please submit one pdf or a zip file that contains all the source code
  - For questions 3–4, 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 5

## Questions:

1. (8 pt) Write FuncLang programs to process a list of strings:

   (a) (4 pt) Given you a list of strings, report the total length of all the strings in the list. You can use the built-in function length which returns the length of a string (e.g. (length "hello") returns 5). Hint: you can use functions define in the previous homework to define Total).
   $ (Total (list "hello" "hi" "you"))
   $ 10

   (b) Given you a list of strings, report the length of the longest string in the list
   $ (Max (list "hello" "hi" "you" "supercalifragilisticexpialidocious"))
   $ 34

   **Sol.**

   (a) (4 pt)

```
(define foldl
  (lambda (op zero lst)
        (if (null? lst)
          zero
          (foldl op (op (car lst) zero) (cdr lst)))))
(define Total
  (lambda (lt)
        (if (null? lt)
          0
          (foldl
            (lambda (x y) (+ (length x) y))
            (length (car lt))
            (cdr lt)))))
```

(b) (4 pt)

```
(define foldl
  (lambda (op zero lst)
    (if (null? lst)
      zero
     (foldl op (op (car lst) zero) (cdr lst)))))
(define Max
  (lambda (lt)
    (if (null? lt)
      0
      (foldl
        (lambda (x y) (if (> (length x) y) (length x) y))
        (length (car lt))
        (cdr lt)))))
```

2. (6 pt) Write a FuncLang program called Shuffle, which takes an input list, "shuffles" it and returns a list whose members are ordered randomly. You can use Random(lst) (it randomly selects an element from the list) to write your program.

   **Sol.**

```
(define shuffle
      (lambda (lst)
        (if (null? lst)
          (list)
          (let ((n (Random lst)))
            (cons n (shuffle (remove n lst)))))))
(define remove
  (lambda (a lst)
    (if (null? lst)
      (lst)
```

```
(if (= a (car lst))
  (cdr lst)
  (cons (car lst) (remove a (cdr lst)))))))
```

3. (20 pt) Extend the FuncLang interpreter by supporting ">" and "<" and "=" on strings and lists, supporting "=" on boolean values. For "=", we return true if the two strings have the exact length and content. Two list values are considered equal if they have the same size and each element of the list is equal to corresponding element in the other list. For "<" and ">", the string and list comparison is done using the length of the strings and lists. That is, "> first second" returns true if the first string/list is longer than the second string/list; and "< first second" returns true if the first string/list is shorter than the second string/list.

   For example,

   $ (= "abc" "abc")
   #t
   $ (= "abc" "abcdef")
   #f

   $ (> "abc" "abcd")
   #f
   $ (< "abc" "abcdef")
   #t

   $ (= #t #t)
   #t
   $ (= #t #f)
   #f

   $ (= (list) (list))
   #t
   $ (= (list 1 2 3 4) (list 1 2 3 4))
   #t
   $ (= (list 1 2 3 4) (list 1 2 3 4 5))
   #f
   $ (= (list 1 2 3 4 (list)) (list 1 2 3 4 (list)))
   #t
   $ (= (car (list 1 2 3)) 1)
   #t
   $ (= (car (list 1 2 3)) 2)
   #f
   $ (= (cdr (list 1 2 3)) 2)
   #f
   $ (= (cdr (list 1 2 3)) (list 2 3))
   #t
   $ (= (cdr (list 1 2 3)) (cdr (list 4 2 3)))
   #t

```
$ (= (cons 0 (list 1 2)) (list 0 (list 1 2)))
#f
$ (= (cons 0 (list 1 2)) (list 0 1 2))
#t
$ (> (list 1 2) (list))
#t
$ (> (list) (list 1))
#f $ (< (list 1 2) (cdr (list 2 3 4 5)))
#t
```

**Sol.** The logic in this answer can varies, but be sure to check if the tests hold. We make the method isList in PairVal public to use it in the comparison and also create size to return the size of the list (2 for pairs). Then defines the methods equalValue which checks for equality and compareValue which returns 0 for equal, -1 for less and 1 for greater. This logic can change and maybe use just one method.

(5 pt)

```
static class PairVal implements Value {
  // make isList public
  public boolean isList() {
    if(_snd instanceof Value.Null) return true;
    if(_snd instanceof Value.PairVal &&
            ((Value.PairVal) _snd).isList()) return true;
    return false;
  }

  // define size
  public int size() {
    if (!isList()) return 2;
    int result = 0;
    if (!(_fst instanceof Value.Null)) {
      result += 1;
    }

    Value next = _snd;
    while(!(next instanceof Value.Null)) {
      result += 1;
      next = ((PairVal) next)._snd;
    }

    return result;
  }
}
```

(15 pt)

```
public class Evaluator implements Visitor<Value> {

  public static boolean equalValue(Value v1, Value v2) {
    if (v1 instanceof NumVal && v2 instanceof NumVal) {
      NumVal first = (NumVal) v1;
      NumVal second = (NumVal) v2;
      return Double.compare(first.v(), second.v()) == 0;
    } else if (v1 instanceof StringVal && v2 instanceof StringVal) {
      String s1 = ((StringVal)v1).v();
      String s2 = ((StringVal)v2).v();
      return s1.equals(s2);
    } else if (v1 instanceof PairVal && v2 instanceof PairVal) {
      boolean b1 = equalValue(((PairVal)v1).fst(), ((PairVal)v2).fst());
      boolean b2 = equalValue(((PairVal)v1).snd(), ((PairVal)v2).snd());
      if (b1 && b2) return true;
      return false;
    } /*else if (v1 instanceof FunVal && v2 instanceof FunVal) {
      return v1 == v2;
    } */else if (v1 instanceof BoolVal && v2 instanceof BoolVal) {
      return ((BoolVal)v1).v() == ((BoolVal)v2).v();
    } else if (v1 instanceof Null && v2 instanceof Null){
      // list
      return true;
    } else {
      return false;
    }
  }

  public static int compareValue(Value v1, Value v2) {
    if (equalValue(v1, v2)) {
      return 0;
    }

    if (v1 instanceof NumVal && v2 instanceof NumVal) {
      NumVal first = (NumVal) v1;
      NumVal second = (NumVal) v2;
      return Double.compare(first.v(), second.v());
    } else if (v1 instanceof StringVal && v2 instanceof StringVal) {
      String s1 = ((StringVal)v1).v();
      String s2 = ((StringVal)v2).v();
      return s1.compareTo(s2);
    } else if (v1 instanceof PairVal && v2 instanceof PairVal) {
      PairVal p1 = (PairVal) v1;
      PairVal p2 = (PairVal) v2;
      if (p1.isList() && p2.isList()) {
```

```
            // we define a size method in PairVal
            return Integer.compare(p1.size(), p2.size());
          } else if (!p1.isList() && !p2.isList()) {
            // this case can be omitted in hw
            // if they are both pairs, the result cannot be applied
            return 0;
          }
        }

        // default case can be 1 or -1 to define different
        return -1;
      }

    @Override
    public Value visit(LessExp e, Env env) { // New for funclang.
      Value first = (Value) e.first_exp().accept(this, env);
      Value second = (Value) e.second_exp().accept(this, env);
      return new Value.BoolVal(compareValue(first, second) < 0);
    }

    @Override
    public Value visit(EqualExp e, Env env) { // New for funclang.
      Value v1 = (Value) e.first_exp().accept(this, env);
      Value v2 = (Value) e.second_exp().accept(this, env);
      return new BoolVal(equalValue(v1, v2));
    }

    @Override
    public Value visit(GreaterExp e, Env env) { // New for funclang.
      Value first = (Value) e.first_exp().accept(this, env);
      Value second = (Value) e.second_exp().accept(this, env);
      return new Value.BoolVal(compareValue(first, second) > 0);
    }
}
```

4. (25 points) Extend the functionality of the Funclang interpreter to add support for following eight predicates: number?, boolean?, string?, procedure?, pair?, list?, null?, unit? A predicate is a function from Funclang value to boolean, i.e. #t or #f.

   The predicate number? evaluates to #t if its input is a number. Similarly, boolean?, string? procedure? pair?, list?, null?, and unit? evaluate to #t if their inputs are boolean, string, procedure, pair, list, null, and unit respectively. Following transcript illustrates some of these predicates.

   > (number? 342)
   #t
   > (boolean? (> 300 42))
   #t
   > (string? "342")

```
#t
> (procedure? (lambda (x) x))
#t
> (list? (list 3 4 2))
#t
> (list? (cons 1 2))
#f
> (pair? (cons 1 2))
#t
> (list? (cons 1 (list)))
#t
> (list? (list ))
#t
> (null? (list ))
#t
```

Funclang doesn't currently support expressions that produce unit values, so you wouldn't be able to test unit?, but you should implement it and we will verify your code directly to see if it behaves as intended (e.g (unit? value)).

**Sol**

Grammar file (5pt):

```
    exp returns [Exp ast]:
        ...
        | isnum=isNumExp { $ast = $isnum.ast; }
        | isbool=isBoolExp { $ast = $isbool.ast; }
        | isstring=isStringExp { $ast = $isstring.ast; }
        | isproc=isProcedureExp { $ast = $isproc.ast; }
        | islist=isListExp { $ast = $islist.ast; }
        | ispair=isPairExp { $ast = $ispair.ast; }
        | isunit=isUnitExp { $ast = $isunit.ast; }
        ;

isNumExp returns [IsNumExp ast] :
                '(' 'number?'
                    e=exp
                ')' { $ast = new IsNumExp($e.ast); }
                ;

isBoolExp returns [IsBoolExp ast] :
        '(' 'boolean?'
            e=exp
        ')' { $ast = new IsBoolExp($e.ast); }
        ;

isStringExp returns [IsStringExp ast] :
```

```
        '(' 'string?'
            e=exp
        ')' { $ast = new IsStringExp($e.ast); }
        ;

isProcedureExp returns [IsProcedureExp ast] :
        '(' 'procedure?'
            e=exp
        ')' { $ast = new IsProcedureExp($e.ast); }
        ;

 isListExp returns [IsListExp ast] :
        '(' 'list?'
            e=exp
        ')' { $ast = new IsListExp($e.ast); }
        ;

 isPairExp returns [IsPairExp ast] :
        '(' 'pair?'
            e=exp
        ')' { $ast = new IsPairExp($e.ast); }
        ;

 isUnitExp returns [IsUnitExp ast] :
        '(' 'unit?'
            e=exp
        ')' { $ast = new IsUnitExp($e.ast); }
        ;
```

AST.java (5pt)

```
public interface AST {
  public static class IsNumExp extends Exp {
    private Exp _arg;
    public IsNumExp(Exp arg){
      _arg = arg;
    }
    public Exp arg() { return _arg; }
    public Object accept(Visitor visitor, Env env) {
      return visitor.visit(this, env);
    }
  }

  public static class IsBoolExp extends Exp {
    private Exp _arg;
    public IsBoolExp(Exp arg){
      _arg = arg;
```

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

  public static class IsStringExp extends Exp {
    private Exp _arg;
    public IsStringExp(Exp arg){
      _arg = arg;
    }
    public Exp arg() { return _arg; }
    public Object accept(Visitor visitor, Env env) {
      return visitor.visit(this, env);
    }
  }

  public static class IsProcedureExp extends Exp {
    private Exp _arg;
    public IsProcedureExp(Exp arg){
      _arg = arg;
    }
    public Exp arg() { return _arg; }
    public Object accept(Visitor visitor, Env env) {
      return visitor.visit(this, env);
    }
  }

  public static class IsListExp extends Exp {
    private Exp _arg;
    public IsListExp(Exp arg){
      _arg = arg;
    }
    public Exp arg() { return _arg; }
    public Object accept(Visitor visitor, Env env) {
      return visitor.visit(this, env);
    }
  }

  public static class IsPairExp extends Exp {
    private Exp _arg;
    public IsPairExp(Exp arg){
      _arg = arg;
    }
    public Exp arg() { return _arg; }
```

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

  public static class IsUnitExp extends Exp {
    private Exp _arg;
    public IsUnitExp(Exp arg){
      _arg = arg;
    }
    public Exp arg() { return _arg; }
    public Object accept(Visitor visitor, Env env) {
      return visitor.visit(this, env);
    }
  }
  public interface Visitor <T> {
    public T visit(AST.IsNumExp e, Env env);
    public T visit(AST.IsBoolExp e, Env env);
    public T visit(AST.IsStringExp e, Env env);
    public T visit(AST.IsProcedureExp e, Env env);
    public T visit(AST.IsListExp e, Env env);
    public T visit(AST.IsPairExp e, Env env);
    public T visit(AST.IsUnitExp e, Env env);
  }
}
```

Evalutor.java (10pt)

```
public class Printer {
  public static class Formatter implements AST.Visitor<String> {

    public class Evaluator implements Visitor<Value> {
      @Override
      public Value visit(IsNumExp e, Env env) {
        Value val = (Value) e.arg().accept(this, env);
        return new BoolVal(val instanceof Value.NumVal);
      }

      @Override
      public Value visit(IsBoolExp e, Env env) {
        Value val = (Value) e.arg().accept(this, env);
        return new BoolVal(val instanceof Value.BoolVal);
      }

      @Override
      public Value visit(IsStringExp e, Env env) {
        Value val = (Value) e.arg().accept(this, env);
```

```
        return new BoolVal(val instanceof Value.StringVal);
      }


      @Override
      public Value visit(IsProcedureExp e, Env env) {
        Value val = (Value) e.arg().accept(this, env);
        return new BoolVal(val instanceof Value.FunVal);
      }

      @Override
      public Value visit(IsListExp e, Env env) {
        Value val = (Value) e.arg().accept(this, env);
        if(val instanceof Value.PairVal){
          return new BoolVal(((PairVal) val).isList());
        }
        else if(val instanceof Value.Null){
          return new BoolVal(true);
        }

        return new BoolVal(false);
      }

      @Override
      public Value visit(IsPairExp e, Env env) {
        Value val = (Value) e.arg().accept(this, env);
        return new BoolVal(val instanceof Value.PairVal);
      }

      @Override
      public Value visit(IsUnitExp e, Env env) {
        Value val = (Value) e.arg().accept(this, env);
        return new BoolVal(val instanceof Value.UnitVal);
      }
    }
```

Printer.java (5pt)

```
  public String visit(AST.IsNumExp e, Env env) {
    String result = "(number? ";
    result += e.arg().accept(this, env);
    return result + ")";
  }

  public String visit(AST.IsBoolExp e, Env env) {
    String result = "(boolean? ";
    result += e.arg().accept(this, env);
```

```
    return result + ")";
  }

  public String visit(AST.IsStringExp e, Env env) {
    String result = "(string? ";
    result += e.arg().accept(this, env);
    return result + ")";
  }

  public String visit(AST.IsProcedureExp e, Env env) {
    String result = "(procedure? ";
    result += e.arg().accept(this, env);
    return result + ")";
  }

  public String visit(AST.IsPairExp e, Env env) {
    String result = "(pair? ";
    result += e.arg().accept(this, env);
    return result + ")";
  }

  public String visit(AST.IsListExp e, Env env) {
    String result = "(list? ";
    result += e.arg().accept(this, env);
    return result + ")";
  }

  public String visit(AST.IsUnitExp e, Env env) {
    String result = "(unit? ";
    result += e.arg().accept(this, env);
    return result + ")";
  }
    }
  }
```