Fundamentals of Computer Science, Fall 2013, Exam 1

17 October 2013

Your Name

Your Instructor

1. This exam is open-book, open-notes. You may use any books, any notes, any written materials you brought along. Keep in mind that "design" means "follow the design recipes" that you know from parts I, II, and III (HtDP/2e) or I, II, and IV (HtDP/1e).

2. **The use of any electronic tools (laptops, phones, etc.) will result in your immediate expulsion and a 0 score.** Any other attempt to obtain solutions from anyone else will be punished in the same way.

3. Make sure that you have all seven problems.

4. Read every problem carefully. If you are stuck, move on to the another problem; return later with new ideas.

5. Your solutions may use any ISL syntax: `define`, `define-struct`, `cond`, `else`, `quote`, `quasiquote`, `unquote`, and literal constants. (Yes, this is the entire syntax.) You may use the ISL functions and library functions listed on page **??**. For signatures, you may use the abbreviations `List-of` and `Maybe`. Everything else you need you must define.

Problem 1 (6 points) *Identify the correct/incorrect data definitions below and explain in one line why they are correct/incorrect.*

1. *A* Carrier *is one of:*
   - `"horse"`
   - `"pony"`
   - `"mule"`
   - `"ox"`
   - `"camel"`
   - `"elephant"`

   Solution 1 (2 points for correct explanation) Carrier *is correctly defined because it simply enumerates constants.*

2. *A* Rank *is* `(make-rank String PositiveNumber NegativeNumber)`.

   ```
   (define-struct rank (name score))
   ```

   Solution 2 (2 points for correct explanation) *The data definition for* Rank *is incorrect because it uses the* `make-rank` *constructor on three arguments although the corresponding structure type definition introduces only two fields.*

3. *A* LOQ *is one of:*
   - `empty`
   - `(cons Quotation LOQ)`

   Solution 3 (2 points for correct explanation) *The data definition for* LOQ *is incorrect because it refers to* Quotation*, an undefined data collection.*

Problem 2 (6 points) *Identify the correct/incorrect signatures below and explain in one line why they are correct/incorrect.*

1. `f : Number Boolean -> Number`

   Solution 4 (2 points for correct explanation) *The signature is correct because its pieces refer to known BSL collections of data.*

2. `g : Tree String -> Number` *where*

   ```
   (define-struct node (left right))
   (define-struct info (name phone))
   ;; A Tree is one of:
   ;; – (make-info String Number)
   ;; – (make-node Tree Tree)
   ```

   Solution 5 (2 points for correct explanation) *The signature is correct because its pieces refer to known BSL collections of data or are defined here.*

3. `h : Box List -> Number`

   ```
   (define-struct box (label content))
   ;; A Box is a (make-box String Number)
   ```

   Solution 6 (2 points for correct explanation) *The signature is meaningless (incorrect) because* `List` *is an undefined term.*

Problem 3 (12 points) *For each of the following data definitions, develop a template for functions that process instances of the defined data collections:*

1. *A* EVEN *is one of:*

   ;; – `empty`
   ;; – `(cons Image (cons Number EVEN))`

   Solution 7 (4 points)

```
(define (template-for-2-lon a-2lon)
  (cond ;;
    [(empty? a-2lon) ...]
    [else (... (first a-2lon)
               (second a-2lon)
;;
               (template-for-2-lon (rest (rest a-2lon)))
;;
               ...)]))
```

2. *A* PITS *is* `(make-place Number Number PositiveNumber)`

   ```
   (define-struct place (x y t))
   ```

   Solution 8 (4 points)

```
(define (template-for-pis p)
  (... (place-x p) .. (place-y p) .. (place-t p) ...))
;;
```

3. A Tree *is one of:*

```
;; – empty
;; – (make-branch Tree Number Tree)

(define-struct branch (left info right))
```

Solution 9 (4 points)

```
(define (template-for-tree t)
  (cond ;;
    [(empty? t) ...]
    [(branch? t)
     ;;
     (... (template-for-tree (branch-left t))
          (branch-info t)
          ;;
          (template-for-tree (branch-right t))
          ....)]))
```

Problem 4 (8 points) *Design the function* `the-word`. *It consumes a list of strings and produces the string made up of the first letters in these strings.*

Solution 10

```
;;
;;  LOS is one of:
;; -- empty
;; -- (cons String LOS)

;;
;; LOS -> String
;; the list of first letters in the give strings

;;
(check-expect (the-word '("hello" "elf" "loaf" "pan")) "help")

(define (the-word l)
  (cond ;;
    [(empty? l) ""]
    [else
     ;;
     ;;
     (string-append (first (explode (first l)))
                    ;;
                    (the-word (rest l)))]))
```

Problem 5 (16 points) *Develop a data definition for the representation of points in the upper-right part of the Cartesian plane. Design a function that consumes lists of such points and returns a list of their respective Manhattan distances.* **Domain Knowledge** *The Manhattan distance of a point is the sum of its distances to the x and y axes, respectively. Recall that a distance is a non-negative number.* **Intuition** *It takes that many blocks to follow the grid pattern to the origin (the (0,0) point).*

Solution 11

```
;; data definition: ;;
;; URQ is (make-posn PositiveNumber PositiveNumber)

;;
;; [List-of URQ] -> [List-of NonnegativeNumber]
;;
;; produce the Manhattan distances to origin for the points on l

;;
(check-expect (dist (list (make-posn -3 4) (make-posn -1 8)))
              (list 7 9))

(define (dist l)
  (cond ;;
    [(empty? l) empty]
    ;;
    [else (cons (mh-distance (first l)) (dist (rest l)))]))

;;

;;
;; URQ -> NonnegativeNumber
;; determine the Manhattan distance of a URQ p to the origin

;;
(check-expect (mh-distance (make-posn 3 4)) 7)

(define (mh-distance p)
  (+ (posn-x p) (posn-y p)))
```

Problem 6 (10 points) *A geometer measures the distances between some points on a completely straight freeway. The origin is the beginning of the freeway. The measurements are done in miles. Design a function that converts a list of such incremental measurements to a list of distances to the origin. Both the given and the expected values are lists of numbers.*

If `(list 1 3 4)` *is returned, the first point is 1 mile from the origin, the second point is 3 miles from the origin, and so on.*

If `(list 1 3 1 2)` *is given, it means that the first point is 1 mile away from the origin, the second point is 3 miles away from the first point, the third point is 1 mile away from the second, and so on.*

Solution 12

```
;;
;; [List-of PositiveNumber] -> [List-of PositiveNumber]
;; convert a list of relative distances to absolute distances

;;
(check-expect (convert '()) '())
(check-expect (convert (list 1 2 3)) (list 1 3 6))

(define (convert l)
  (cond ;;
    [(empty? l) '()]
    ;;
    [else
     (cons (first l) (add-all (first l) (convert (rest l))))]))

;;
;; Number [List-of PositiveNumber] -> [List-of PositiveNumber]
;; add n to all numbers on list l

(check-expect (add-all 1 '()) '())
(check-expect (add-all 1 (list 2 3)) (list 3 4))

(define (add-all n l)
  (cond
    [(empty? l) '()]
    [else (cons (+ (first l) n) (add-all n (rest l)))]))
```

Alternative solution, designed with recipes from parts I–III:

```
;; [List-of PositiveNumber] -> [List-of PositiveNumber]
;; convert list of relative distances to list of absolute distances

(check-expect (geo '(1 2 1)) '(1 3 4))
(check-expect (geo '(1 2 3 2)) '(1 3 6 8))

(define (geo lor0)
  (geo-relative-to (sum lor0) lor0))

;; [List-of Number] -> [List-of Number]
;; convert list of relative distances to list of absolute distances

(check-expect (geo-relative-to 4 '(1 2 1)) '(1 3 4))
(check-expect (geo-relative-to 8 '(1 2 3 2)) '(1 3 6 8))

(define (geo-relative-to total lor)
  (cond
    [(empty? lor) '()]
    [(cons? lor)
     (cons (- total (sum (rest lor)))
           (geo-relative-to total (rest lor)))]))

;; [List-of Number] -> Number
;; sum of all numbers
(define (sum lor)
  (foldr + 0 lor)) ;; cheap trick, not allowed by exam.
```

Problem 7 (2 points) *Explain in a complete sentence of at most 30 words what this course is trying to teach you. To earn those two points, it matters that (1) your paragraph's grammar/spelling is correct and (2) its content relates to the course material. Your positive/negative opinion is immaterial.*

Solution 13 *Here is an answer that would get 2 points:*

> This (1) course (2) introduces (3) the (4) syntax (5) and (6) the (7) programming (8)
> idioms (9) of (10) a (11) small (12) but (13) powerful (14) but (15) completely (16) irrelevant (17)
> programming (18) language (19).

It would also make your instructors utterly unhappy.

In contrast, the following sentence would get you full credit and leave ecstatic instructors in its wake:

> This (1) course (2) explains (3) a (4) language-independent (5) approach (6) to (7)
> the (8) systematic (9) design (10) of (11) programs (12), some (13) of (14) whose (15) elements (16)
> also (17) apply (18) to (19) other (20) domains (21) of (22) expertise (23).

We trust that you can construct ungrammatical, incorrectly spelled, and overly long sample sentences on your own.