プログラミング論 I

・全ての問題を解き，解答用紙に記入して提出してください．氏名の記入を忘れないこと．
　期限は，7月12日8時40分とします（期限厳守）．それ以降は受け取りません．
・以下の問いでわかるように、授業の内容を理解し、プログラムの作成能力が習得できているかを問うもの．慌てて解くのでは無く，時間をかけて自分の力でじっくり解くのが望ましい．
・解答用紙は返却しませんが，7月12日に，「回答例」を配布します．

1. 次の Scheme 式の実行結果を解答せよ

(1) `(and (> 100 1) (< 50 200))`
(2) `(not (or (= 2 1) (= 2 2)))`
(3) `(string=? "map" "cat")`
(4) `(rest (rest (rest (list 1 2 3))))`

2. 次の Scheme 式を実行するとエラーが発生する．エラーの原因を分かりやすく説明せよ．

(1) `(rest empty)`
(2) `(= (list 1 2 3) (list 1 2 3))`

3. 次の関数についての問題

(1) 下記の関数 **my_sum** について，「**(my_sum 5 3)**」から「**8**」に至る過程の概略を説明しなさい．

```
;; my_sum number[>=0] number[>=0] -> number[>=0]
(define (my_sum x y)
  (cond
    [(= x 0) y]
    [else (my_sum (- x 1) (+ y 1))]))
```

(2) 下記の関数 **numlist** について，「**(numlist 3)**」から「**(list 3 2 1)**」に至る過程の概略を説明しなさい．

```
;; numlist: number[>=1] -> (listof number)
(define (numlist n)
  (cond
    [(= n 1) (list 1)]
    [else (cons n (numlist (- n 1)))]))
```

(3) 下記の関数 **filter1** について，「(filter1 string<=? (list "map" "apple" "dog") "cat") 」から「**"apple"**」に至る過程の概略を説明しなさい．

```
;;filter1: (X X -> boolean) (listof X) X -> (listof X)
(define (filter1 rel_op alox t)
  (cond
    [(empty? alox) empty]
    [else (cond
            [(rel_op (first alox) t)
                (cons (first alox)
                  (filter1 rel_op (rest alox) t))]
            [else (filter1 rel_op (rest alox) t)])]))
```

(4) 下記の関数 **invert** について，「**(invert (list 100 200 300))**」から「**(list 300 200 100)**」に至る過程の概略を説明しなさい．

```
;; invert : (listof X) -> (listof X)
;; to construct the reverse of alox
(define (invert alox0)
 (local (;; accumulator is the reversed list of all those items
          ;; on alox0 that precede alox
          (define (rev alox accumulator)
           (cond
            [(empty? alox) accumulator]
            [else
```

```
            (rev (rest alox) (cons (first alox) accumulator))])))
      (rev alox0 empty)))
```

4. 構造体 personal_info 型が次のように定義されている

```
(define-struct personal_info (name age address))
```

(1) 構造体 **personal_info** 型のリストを入力とし，リストの要素数を出力とする関数 **count** の Contract(規約)を書きなさい
(2) (1) の関数 **count** を定義しなさい
(3) 構造体 **personal_info** 型のリストを入力とし，平均年齢を出力とする関数 **average_age** の Contract(規約)を書きなさい
(4) (3) の関数 **age_average** を定義しなさい
(5) 構造体 **personal_info** 型のリストを入力とし，20 歳以上の人の name のリストを出力とする関数 **select_by_age** の Contract(規約)を書きなさい
(6) (5) の関数 **select_by_age** を定義しなさい

5. 次の２つの関数 is-smaller, is-larger についての問題

```
(define-struct personal_info
               (name age address))
;; is-smaller: personal_info personal_info -> boolean
(define (is-smaller x y)
  (cond
    [(not (personal_info? x)) false]
    [(not (personal_info? y)) false]
    [(string>=? (personal_info-name x)
                (personal_info-name y)) false]
    [else true]))
;; is-larger: personal_info personal_info -> boolean
(define (is-larger x y)
  (cond
    [(not (personal_info? x)) false]
    [(not (personal_info? y)) false]
    [(string<=? (personal_info-name x)
                (personal_info-name y)) false]
    [else true]))
```

(1) ２つの関数 **is-smaller, is-larger** を，単一の関数 **select_by_name** に抽象化しなさい（参考：「設計の抽象化」の回の資料）
(2) (1) で定義した抽象関数 **select_by_name** を使い，**is-smaller** と同等の関数 **is-smaller1** を定義しなさい

6. 下記の書籍データセットについての問題

name	price	category	publisher
Eleven on Top	$17.79	Mystery	St. Martin's Press
Freakonomics	$15.57	Business	William Morrow
Harry Potter	$17.99	Children	Levine Books
The Da Vinci Code	$14.97	Mystery	Doubleday

(1) この書籍データセットを，１つの変数 **a** として定義しなさい
(2) 書籍データセットに，新しい１つの書籍を挿入するための関数 **insert_into_booklist** の Contract(規約) を書きなさい
(3) 関数 **insert_into_booklist** を定義しなさい

プログラミング論Ⅰ　回答例

1. 次の Scheme 式の実行結果を解答せよ

(1) `(and (> 100 1) (< 50 200))`

```
true
```

(2) `(not (or (= 2 1) (= 2 2)))`

```
false
```

(3) `(string=? "map" "cat")`

```
false
```

(4) `(rest (rest (rest (list 1 2 3))))`

```
empty
```

2. 次の Scheme 式を実行するとエラーが発生する．エラーの原因を分かりやすく説明せよ．

(1) `(rest empty)`

rest は，要素数が１個以上のリストに対して，リストの先頭を取った残りを出力するもの．空リストを **rest** に適用するとエラーが発生する．

(2) `(= (list 1 2 3) (list 1 2 3))`

= は数値同士の比較を行うもの．数値では無いので、エラーが発生する．

3. 次の関数についての問題

(1) 下記の関数 **my_sum** について，「**(my_sum 5 3)**」から「**8**」に至る過程の概略を説明しなさい．

```
;; my_sum number[>=0] number[>=0] -> number[>=0]
(define (my_sum x y)
  (cond
    [(= x 0) y]
    [else (my_sum (- x 1) (+ y 1))]))
```

```
(my_sum 5 3)
→ (my_sum 4 4)
→ (my_sum 3 5)
→ (my_sum 2 6)
→ (my_sum 1 7)
→ (my_sum 0 8)
→ 8
```

(2) 下記の関数 **numlist** について，「**(numlist 3)**」から「**(list 3 2 1)**」に至る過程の概略を説明しなさい．

```
;; numlist: number[>=1] -> (listof number)
(define (numlist n)
  (cond
    [(= n 1) (list 1)]
    [else (cons n (numlist (- n 1)))]))
```

```
(numlist 3)
→ (cons 3 (numlist 2))
→ (cons 3 (cons 2 (numlist 1)))
→ (cons 3 (cons 2 (list 1)))
→ (list 3 2 1)
```

(3) 下記の関数 **filter1** について，「(filter1 string<=? (list "map" "apple" "dog") "cat") 」から「**(list "apple")**」に至る過程の概略を説明しなさい．

```
;;filter1: (X X -> boolean) (listof X) X -> (listof X)
(define (filter1 rel_op alox t)
  (cond
    [(empty? alox) empty]
```

```
    [else (cond
            [(rel_op (first alox) t)
                (cons (first alox)
                  (filter1 rel_op (rest alox) t))]
            [else (filter1 rel_op (rest alox) t)])]))

       (filter1 string<=? (list "map" "apple" "dog") "cat")
       → (filter1 string<=? (rest (list "map" "apple" "dog")) "cat")
       → (cons "apple" (filter1 string<=? (rest (list "apple" "dog")) "cat"))
       → (cons "apple" (filter1 string<=? (rest (list "dog")) "cat"))
       → (cons "apple" (filter1 string<=? empty "cat"))
       → (cons "apple" empty)
       → (list "apple")
```

(4) 下記の関数 **invert** について，「**(invert (list 100 200 300))**」から「**(list 300 200 100)**」に至る過程の概略を説明しなさい．

```
;; invert : (listof X) -> (listof X)
;; to construct the reverse of alox
(define (invert alox0)
 (local (;; accumulator is the reversed list of all those items
         ;; on alox0 that precede alox
         (define (rev alox accumulator)
          (cond
           [(empty? alox) accumulator]
           [else
            (rev (rest alox) (cons (first alox) accumulator))])))
       (rev alox0 empty)))

      (invert (list 100 200 300))
      →(rev (list 100 200 300) empty)
      →(rev (list 200 300) (list 100))
      →(rev (list 300) (list 200 100))
      →(rev empty (list 300 200 100))
      →(list 300 200 100)
```

4. 構造体 personal_info 型が次のように定義されている

```
(define-struct personal_info (name age address))
```

(1) 構造体 **personal_info** 型のリストを入力とし，リストの要素数を出力とする関数 **count** の Contract(規約)を書きなさい

```
;; count: (listof personal_info) -> number
```

(2) (1) の関数 **count** を定義しなさい

```
;; count: (listof personal_info) -> number
(define (count a_list)
  (cond
    [(empty? a_list) 0]
    [else (+ 1
           (count (rest a_list)))]))
```

(3) 構造体 **personal_info** 型のリストを入力とし，平均年齢を出力とする関数 **average_age** の Contract(規約)を書きなさい

```
;; average_age: (listof personal_info) -> number
```

(4) (3) の関数 **average_age** を定義しなさい

```
;; count: (listof personal_info) -> number
(define (count a_list)
  (cond
    [(empty? a_list) 0]
    [else (+ 1
             (count (rest a_list)))]))
```

（2）の count 関数と同じ

```
;; sum: (listof personal_info) -> number
(define (sum a_list)
  (cond
    [(empty? a_list) 0]
    [else (+ (personal_info-age (first a_list))
             (count (rest a_list)))]))

;; average_age: (listof personal_info) -> number
(define (average_age a_list)
  (/ (sum a_list) (count a_list)))
```

(5) 構造体 **personal_info** 型のリストを入力とし，20 歳以上の人の name のリストを出力とする関数 **select_by_age** の Contract(規約)を書きなさい

```
;; select_by_age: (listof personal_info) -> (listof string)
```

(6) (5) の関数 **select_by_age** を定義しなさい

```
;; select_by_age: (listof personal_info) -> (listof string)
(define (select_by_age a_list)
  (cond
    [(empty? a_list) empty]
    [else
     (cond
       [(>= (personal_info-age (first a_list)) 20)
        (cons (personal_info-name (first a_list))
              (select_by_age (rest a_list)))]
       [else (select_by_age (rest a_list))])]))
```

5. 次の２つの関数 is-smaller, is-larger についての問題

```
(define-struct personal_info
               (name age address))
;; is-smaller: personal_info personal_info -> boolean
(define (is-smaller x y)
  (cond
    [(not (personal_info? x)) false]
    [(not (personal_info? y)) false]
    [(string>=? (personal_info-name x)
                (personal_info-name y)) false]
    [else true]))
;; is-larger: personal_info personal_info -> boolean
(define (is-larger x y)
  (cond
    [(not (personal_info? x)) false]
    [(not (personal_info? y)) false]
    [(string<=? (personal_info-name x)
                (personal_info-name y)) false]
    [else true]))
```

(1) ２つの関数 **is-smaller, is-larger** を，単一の関数 **select_by_name** に抽象化しなさい（参考：「設計の抽象化」の回の資料）

```
;; select_by_name: personal_info personal_info (string string -> boolean) -> boolean
(define (select_by_name x y op)
  (cond
    [(not (personal_info? x)) false]
    [(not (personal_info? y)) false]
    [(op (personal_info-name x)
              (personal_info-name y)) false]
    [else true]))
```

(2) (1) で定義した抽象関数 **select_by_name** を使い，**is-smaller** と同等の関数 **is-smaller1** を定義しなさい

```
;; is-smaller1: personal_info personal_info -> boolean
(define (is-smaller1 x y)
  (select_by_name x y string>=?))
```

6. 下記の書籍データセットについての問題

name	price	category	publisher
Eleven on Top	$17.79	Mystery	St. Martin's Press
Freakonomics	$15.57	Business	William Morrow
Harry Potter	$17.99	Children	Levine Books
The Da Vinci Code	$14.97	Mystery	Doubleday

(1) この書籍データセットを，１つの変数 **a** として定義しなさい
book 構造体を次のように定義する.
(define-struct book (name price category publisher))

問題文の書式データセットは，book 構造体を使って，変数 a として次のように定義できる.

```
(define a
  (list (make-book "Eleven on Top"     17.79 "Mystery"  "St. Martin's Press")
        (make-book "Freakonomics"     15.57 "Business" "William Morrow")
        (make-book "Harry Potter"     17.99 "Children" "Levine Books")
        (make-book "The Da Vinci Code" 14.97 "Mystery"  "Doubleday")))
```

(2) 書籍データセットに，新しい１つの書籍を挿入するための関数 **insert_into_booklist** の Contract(規約) を書きなさい

(1) のように，書籍データセットを，構造体 book のリストとして扱う場合には，**insert_into_booklist** の Contract は次のようになる

```
;; insert_info_booklist; (listof book) book -> (listof book)
```

(3) 関数 **insert_into_booklist** を定義しなさい

```
(define (insert_into_booklist booklist a_book)
  (cons a_book booklist))
```