# Programming III COMP2209

## Coursework 1

**Individual Coursework** *(cf. collaboration policy)*

- *Aims of this coursework:*
  1. *Programming with a Scheme environment.*
  2. *Programming recursive functions on flat lists.*
- *Deadline: Week 6, Thursday November 10, 4.00pm.*
- *Electronic submission: Four files:*
  1. `assoc-all.scm`*: answer to question 1.*
  2. `remove-alist-all.scm`*: answer to question 2.*
  3. `group-by-key.scm`*: answer to question 3.*
  4. `map-values.scm`*: answer to question 4.*
- *Learning outcomes:*
  1. *program in a functional style;*
  2. *practice the key mechanisms underpinning the functional programming model;*
  3. *understand the concept of functional programming and be able to write programs in this style in the context of Scheme.*
- *Scheme Language: Standard (R5RS)*
- *Weighting: 25% for each question.*

| *Question* | *Overall Weight* | *Manual Marking* | *Automated Marking* |
|---|---|---|---|
| *Question 1* | *25%* | *10%* | *15%* |
| *Question 2* | *25%* | *0%* | *25%* |
| *Question 3* | *25%* | *0%* | *25%* |
| *Question 4* | *25%* | *10%* | *15%* |
| *Total* | *100%* | *20%* | *80%* |

- *Automated marking: Each question will be marked* ***automatically*** *by running a test sequence based on the illustrations given in appendix. Note that tests in appendix are purely illustrative: a correct solution is expected to satisfy these tests; however, satisfying a test sequence is not a guarantee of a correct solution. The automated marks will be determined by the tests in appendix and further tests to ensure that solutions are not hard-coded for a specific test sequence. Syntax of the notation used in the test sequence and details of how to run tests yourself are available from* `https://secure.ecs.soton.ac.uk/notes/comp2209/15-16/testing.html`
- *Marking scheme (manual marking): The manual mark componennt will be based on style (e.g. code structure and layout, choice of recursion schema, efficiency, memory usage). The breakdown is as follows.*

| *criterion* | *Description* | *Outcomes* | *Marks* |
|---|---|---|---|
| *base case* | *writing the base case of a recursion on flat lists; correctness, style and efficiency* | *1,2,3* | *30%* |
| *inductive case* | *writing the inductive case of a recursion on flat lists; correctness, style and efficiency* | *1,2,3* | *70%* |

This coursework is inspired by the Scala functions `groupBy` and `mapValues` on maps (see http://www.scala-lang.org/api/2.11.8/index.html#scala.collection.immutable.Map).

In this coursework, we consider a general form of association list in which any symbolic expression may be associated with any symbolic expression. Example:

```
( (a . 1)
  (2 . b)
  (() . (a b c))
  ((a b c) . (c d e))
  ((hel . lo) . (he . llo)) )
```

1. Define a function `assoc-all` that takes a symbolic expression and an association list and returns the list of values associated with that expression, in the order they occur in the association list.

   To practice recursion, you are asked to avoid functions such `map`, `filter` or any other higher-order function.

   Appendix A contains illustrations of the function `assoc-all`.

   The test suite checks several features of the solution:

   (a) Usage of imperative constructs is not permitted (tests `$check-no-set-cxr`, `$check-no-set`).

   (b) The total number of pairs constructed is given by the length of the solution (tests `$calculate-cons-usage`).

2. Define a function `remove-assoc-all` that takes a symbolic expression and an association list and returns a new association list containing the same associations as in the input list, except for those associations containing the expression in key position.

   To practice recursion, you are asked to avoid functions such `map`, `filter` or any other higher-order function.

   Appendix B contains illustrations of the function `remove-alist-all`.

   The test suite checks several features of the solution:

   (a) Usage of imperative constructs is not permitted (tests `$check-no-set-cxr`, `$check-no-set`).

   (b) The total number of pairs constructed is given by the length of the solution (tests `$calculate-cons-usage`).

   (c) The solution is expected to be a freshly-constructed association list without sharing with the input (test `$no-sharing`).

   (d) Associations are shared between input and result (test `$share-association`).

3. Define a function `group-by-key` that takes an association list and returns a new association list in which all values of associated with each key have been grouped in a list (by order of their occurrence).

   A correct solution will make use of `assoc-all` and `remove-assoc-all` you have defined previously.

   Appendix C contains illustrations of the function `group-by-key`.

   The test suite checks several features of the solution:

   (a) Usage of imperative constructs is not permitted (tests `$check-no-set-cxr`, `$check-no-set`).

   (b) The solution performs a sub-quadratic number of allocations (test still to be confirmed).

4. Define a function `map-values` that takes an association list and a function and returns a new association list with the same keys in the same order as the input list, but where values are obtained by application of the function to the corresponding value in the input list.

   Appendix D contains illustrations of the function `map-values`.

   The test suite checks several features of the solution:

   (a) Usage of imperative constructs is not permitted (tests `$check-no-set-cxr`, `$check-no-set`).

(b) The total number of pairs constructed by the function is twice the length of the solution plus the total number of pairs created by the function it invokes (tests `$calculate-cons-usage1`, `$calculate-cons-usage2`).

(c) The solution is expected to be a freshly-constructed association list without sharing with the input (test `$no-sharing1`, `$no-sharing2`).

# A Illustrations for `assoc-all`

Test file available from https://secure.ecs.soton.ac.uk/notes/comp2209/16-17/cwk/cwk1/assoc-all-public.tst

```
(load "assoc-all.scm")
:marks 0
+++

(instrument 'reset '* 0)
:marks 0
---

(assoc-all 'a '())
()

(assoc-all 'a '((b . c)))
()

(assoc-all 'a '((b . c) (a . 1)))
(1)

(assoc-all 'a '((b . c) (a . 1) (c . e)))
(1)

(assoc-all 'a '((b . c) (a . 1) (c . e) (a . 2)))
(1 2)

(assoc-all 'a '((b . c) (a . 1) (c . e) (a . 2) (g . k)))
(1 2)

(assoc-all 'a '((b . c) (a . 1) (c . e) (a . 1) (g . k)))
(1 1)

(assoc-all (cons 'a 'b) '((b . c)  (c . e) ((a . b) . 1) ((a . b) . 2) (g . k)))
(1 2)

(instrument 'get 'set-cxr! '_)   ;;; check that there was no mutation
$check-no-set-cxr
:marks 2
0

(instrument 'get 'set! '_)       ;;; check that there was no assignment
$check-no-set
:marks 2
0

(instrument-count-pairs (lambda ()
                 (assoc-all 'a '((b . c)  (c . e) (a . 1)
                                 (a . 2) (g . k) (a . 3) (a . 4)))))
$calculate-cons-usage
:marks 4
4
```

# B Illustrations for `remove-alist-all`

Test file available from https://secure.ecs.soton.ac.uk/notes/comp2209/16-17/cwk/cwk1/remove-alist-all-public.tst

```
(load "remove-alist-all.scm")
:marks 0
+++

(instrument 'reset '* 0)
:marks 0
---

(remove-alist-all 'a '())
()

(remove-alist-all 'b '((b . c)))
()

(remove-alist-all 'a '((b . c) (a . 1)))
((b . c))

(remove-alist-all 'a '((b . c) (a . 1) (a . 3)))
((b . c))

(remove-alist-all 'a '((a . 1) (b . c) (a . 3)))
((b . c))

(remove-alist-all (cons 'a 'b) '(((a . b) . 1) (b . c) ((a . b) . 3)))
((b . c))

(let ((ll '((b . c) (a . 1) (a . 3))))
    (eq? (remove-alist-all 'b ll) (cdr ll)))
$no-sharing-on-spine
:marks 2
#f

(let ((ll '((b . c) (a . 1) (a . 3))))
    (eq? (car (remove-alist-all 'a ll)) (car ll )))
$share-association
:marks 2
#t

(instrument 'get 'set-cxr! '_)   ;;; check that there was no mutation
:marks 2
0

(instrument 'get 'set! '_)       ;;; check that there was no assignment
:marks 2
0

(instrument-count-pairs (lambda ()
                          (remove-alist-all 'a '((b . c)  (c . e) (a . 1)
                                                 (a . 2) (g . k) (a . 3) (a . 4)))))
$calculate-cons-usage
:marks 4
3

(instrument-count-pairs (lambda ()
                          (remove-alist-all 'a '((a . c)  (a . e) (a . 1) (a . 2)))))
$calculate-cons-usage
:marks 4
0
```

# C Illustrations for `group-by-key`

Test file available from https://secure.ecs.soton.ac.uk/notes/comp2209/16-17/cwk/cwk1/group-by-key-public.tst

```
(load "group-by-key.scm")
:marks 0
+++

(instrument 'reset '* 0)
:marks 0
---

assoc-all  ;;; needs definition of assoc-all
$exist-definition-of-assoc
---

remove-alist-all  ;;; needs definition of remove-alist-all
$exist-definition-of-remove
---

(group-by-key '())
()

(group-by-key '((b . c)))
((b c))

(group-by-key '((b . c) (a . 1)))
((b c) (a 1))

(group-by-key '((b . c) (a . 1) (a . 3)))
((b c) (a 1 3))

(group-by-key '((a . 1) (b . c) (a . 3)))
((a 1 3) (b c))

(group-by-key '(((a . b) . 1) (b . c) ((a . b) . 3)))
(((a . b) 1 3) (b c))

(group-by-key '(((a . b) . 1) (b . c) ((a . b) . 1)))
(((a . b) 1 1) (b c))

(instrument 'get 'set-cxr! '_)   ;;; check that there was no mutation
$check-no-set-cxr
:marks 2
0

(instrument 'get 'set! '_)       ;;; check that there was no assignment
$check-no-set
:marks 2
0

(<= (instrument-count-pairs (lambda ()
                              (group-by-key '((b . c)  (c . e) (a . 1)
                                              (a . 2) (g . k) (a . 3) (a . 4)))))
    27)
$calculate-cons-usage
:marks 4
#t
```

# D Illustrations for `map-values`

Test file available from https://secure.ecs.soton.ac.uk/notes/comp2209/16-17/cwk/cwk1/map-values-public.tst

```
(load "map-values.scm")
:marks 0
+++

(instrument 'reset '* 0)
:marks 0
---

(map-values '() (lambda (x) x))
()

(map-values '((b . c)) (lambda (x) x))
((b . c))

(map-values '((b . 2)) (lambda (x) (+ 1 x)))
((b . 3))

(map-values '((b . c) (a . 1)) (lambda (x) x))
((b . c) (a . 1))

(map-values '((b . 2) (a . 1))  (lambda (x) (* 10 x)))
((b . 20) (a . 10))

(map-values '((b . c) (a . 1) (c . e)) (lambda (x) x))
((b . c) (a . 1) (c . e))

(map-values '((b . c) (a . 1) (c . e) (a . 2)) (lambda (x) x))
((b . c) (a . 1) (c . e) (a . 2))

(map-values '((b . 3) (a . 4) (c . 5) (a . 6)) (lambda (x) (* x x)))
((b . 9) (a . 16) (c . 25) (a . 36))

(instrument 'get 'set-cxr! '_)   ;;; check that there was no mutation
$check-no-set-cxr
:marks 2
0

(instrument 'get 'set! '_)       ;;; check that there was no assignment
$check-no-set
:marks 2
0

(instrument-count-pairs (lambda ()
                          (map-values '((b . c)  (c . e) (a . 1) (a . 2) (g . k) (a . 3) (a . 4))
                                      (lambda (x) x))))
$calculate-cons-usage1
:marks 4
14

(instrument-count-pairs (lambda ()
                          (map-values '((b . c)  (c . e) (a . 1) (a . 2) (g . k) (a . 3) (a . 4))
                                      list)))
$calculate-cons-usage2
:marks 4
21

(let ((ll '((b . c) (a . 1) (a . 3))))
     (and (not   (eq?    ll (map-values ll (lambda (x) x))))
                 (equal? ll (map-values ll (lambda (x) x)))))
$no-sharing1
:marks 2
#t
```

```
(let ((ll '((b . c) (a . 1) (a . 3))))
     (and (not   (eq?    (car ll) (car (map-values ll (lambda (x) x)))))
                 (equal? (car ll) (car (map-values ll (lambda (x) x))))))
$no-sharing2
:marks 2
#t
```