University of Toronto
CSC324 –Programming Languages, winter 2011

Assignment # 2

Details
- Topics: Scheme
- Weight: 15%
- Due Date: this assignment is due on Feb 28th, 2011 @ 11:59pm
- This assignment is to be done individually.

Marking
- Your code will be marked for correctness (80%) and code documentation (20%).
-Since we plan to test your code using an auto-marker, you must use the exact function names/arguments as specified and use the same file names and folder name as specified.
- YOU WILL LOSE 25% OF THE TOTAL MARK IF YOU USE WRONG FILE NAMES, FUNCTION NAMES OR DIRECTORY NAME.
- YOU WILL LOSE 10% OF THE TOTAL MARK IF YOUR SUBMISSION FAILS TO LOAD.

Submission
Place all your files in a folder (directory) named after your UTORID (not your CDF login id..) and submit that folder zipped through the portal (if your UTORID is abcd, the file should be named abcd.zip). A link for submission is provided in the assignments folder in the portal (http://portal.utoronto.ca)

Resources

The Scheme Programming Language:	http://www.scheme.com/tspl2d/index.html
Google Scheme groups:	http://groups.google.com/group/comp.lang.scheme/
Scheme Interperter:	http://www.racket-lang.org/

Description
This assignment is to get you used to Scheme syntax, the programming environment and most importantly thinking as a functional language programmer. The assignment requires you to write a series of functions. Even the hardest ones are very short; the difficulty lies in turning your brain around to think about the solution in the right way!

- In order to help you with this new way of thinking, I am disallowing the use of any of Scheme's imperative features: do, begin and all imperative functions ending with ! (e.g. set!, set-car!, vector-set!…).
- You are also not allowed to use the high order functions: map and reduce
- You may use the following functions and special forms: and (and other logical predicate), append, atom, car, cdr(etc..), cond, cons, define, eq (and other equality predicates), length, list, listp, null, numberp, or, display, > (and other comparison predicates),+,-,*,/ This list is incomplete, however,

you can probably solve the entire assignment using only the functions and special forms listed above. If there is something you'd like to use and that is not on the list, ask through discussion board.

(a) **[5 marks]** Write a function **apply-proc** which takes as arguments a unary function and a list, and returns a list of values obtained by applying the function to every element in the list, in order. You can assume that the input function is applicable to every element in the input list. *Sample invocations:*

```
]=> (apply-proc positive? '(1 2 5 -9))
(#t #t #t #f)
]=> (apply-proc length '( () (1) (1 2)  (1 (2 3)) ) )
(0 1 2 2)
```

Write your answer in a file called apply-proc.scm

(b) **[5 marks]** Use **apply-proc** to write a function **squareList** that takes as an argument a list of numbers and returns a list of squares of the numbers in the input list, in order.

Write your answer in a file called squareList.scm

(c) [10 marks] Write a function **min-max** that takes a non-empty list of numbers as an argument and returns a list of two elements: the largest number that appears in the input list and the smallest number that appears in the input list. *Sample invocation:*

```
]=> (min-max '(-10 0 3 -4))
(-10 3)
```

Write your answer in a file called min-max.scm

(d) **[10 marks]** Write a function **intersect** that computes the intersection of two lists. In other words, given two lists as arguments, it returns a list of elements contained in both lists in order. Note that you cannot use the built-in function member. *Sample invocations:*

```
]=> (intersect '(1 2 3 4) '(10 2 4 100) )
    (2 4)
]=> (intersect '(john david) '(david 2 sky 4) )
    (david)
```

Write your answer in a file called intersect.scm

(e) **[10 marks]** Define a function **select-even** that takes a list of numbers as input and returns a list that contains all even numbers from the input list, in the same order as they appear in the input list. You can use built-in predicate **even?** *Sample invocation:*

```
]=> (select-even '(1 2 3 4))
(2 4)
```

Write your answer in a file called select-even.scm

(f) **[20 marks]** Define a function **mult-num** that takes a list as input and returns the product of all numbers in the list. List elements that are not numbers should be ignored. If the input contains no numbers, 1 should be returned. *Sample invocations:*

```
]=> (mult-num '(1 2 3 f 4))
24
]=> (mult-num '(14 () (100) 2.5))
35
```

Write your answer in a file called mult-num.scm

(g) **[20 marks]** Define a function **select** that takes a unary predicate and a list as input and returns a list that contains all elements from the input list, of which the predicate is true, in the same order as they appear in the input list. You can assume that the input predicate is applicable to every element of the input list. *Sample invocation;*

```
]=> (select null? '( () (1 2 3) ))
( () )
```

Write your answer in a file called select.scm