University of Toronto
CSC324 – Principles of Programming Languages, Winter 2007

Assignment 2 (note change in due date)

Due Date: This assignment is due on **Monday February 12, 2007** at **noon**.
Weight: This assignment is worth **5%** of your course grade.

Silent Policy
A silent policy will take effect 24 hours before this assignment is due. This means that no question will be answered, whether it is asked on the newsgroup, by email or in person.

Handing in this Assignment
1. What to hand in on paper**
Please use an unsealed envelope, stapling the cover page provided here to the front. Put inside:

1. A printout of your code.
2. Your answer to question 3.
3. For each procedure that you write, you must describe the testing strategy that you used. This should include a table listing the test cases that you designed for your procedure, what output your procedure returned, and an explanation of why the test case and output are significant in verifying the correctness of the procedure. Read the following for a discussion of good testing practices.
   `http://www.cs.toronto.edu/~sheila/324/w07/general/testing.pdf`

You must hand in this part of the assignment in the CSC324 drop box in the Bahen Computer Lab, BA2220.
***** We strongly prefer paper submissions, but if you cannot get in to the university on time, you may submit your answer to question 3 and your testing strategy electronically.*** We will accept your submission in plain ASCII (in which case your submission file must be called a2.txt), in PostScript (a2.ps), or in PDF (a2.pdf). Please don't waste a lot of time making it look pretty. It need only be legible and clearly structured.

2. What to hand in electronically
In addition to your paper submission, you must submit your code electronically. The procedures (including helper procedures) for each individual question are to be submitted separately using the filename named in the description of the question. Submit the files on CDF following instructions at:

`https://www.cdf.toronto.edu/students`

Warning: marks will be deducted for incorrect electronic submission.

Since we also plan to test your code electronically:

- You must make sure that your code runs on CDF.
- You must use the exact procedure names and argument(s) specified.
- You must not display anything but the procedure output (no text messages to the user, fancy formatting, etc., just what is in the assignment handout.)

Clarification Page
It is your responsibility to consult the A2 Clarification Page
`http://www.cs.toronto.edu/~sheila/assns/A2/a2_faq.html`
and the newsgroup for any corrections or clarifications to the assignment.

How you will be marked
Code will be marked with respect to correctness, style, and documentation. You will also receive marks for the testing strategies.

University of Toronto
CSC324 – Principles of Programming Languages, Winter 2007

Assignment 2

Due Date
This assignment is due on **Monday February 12, 2007** at **noon**.

Please include the following information with your submission.
(No signature is necessary if you submit electronically.)

Last Name: **Email:**

First Name: **CDF Login:**

Student Number:

Date and Time you are handing this in:

Total number of grace days used prior to this assignment:

Number of grace days requested for use on this assignment:

*All answers are my own, written in isolation, without help from others. This submission is in accordance with the University of Toronto Code of Behaviour on Academic Matters (*http://www.artsandscience.utoronto.ca/ofr/calendar/rules.htm#behaviour).

Signature: _____

University of Toronto
CSC324 – Principles of Programming Languages, Winter 2007

Assignment 2 (cont.)

This assignment is a warm-up to get you used to programming in Scheme and to thinking as a functional programmer. The assignment requires you to write a series of procedures. Even the hardest ones are short; the difficulty lies in turning your brain around to think about the solution in the right way, and to encode it elegantly! In order to help you with this new way of thinking, you may only use Scheme procedures that we have covered in class. If there is something you'd like to use and you're not sure about it, ask through the newsgroup. You are allowed to write helper procedures. We also expect you to use recursion in your solutions, if a recursive solution is possible.
Please read page 1 of this assignment and note that you are expected to hand in your testing strategy for each piece of code that you write.

1. [5 marks] Filename for submission: a2-1.scm
Write a procedure **(sub-l x y lst)** that returns the list **lst** with every occurrence of **x** in **lst** replaced by **y.** Note that **lst** can be nested.

Example: (sub-l '1 '2 '(g 1 (+ 1 10))) returns (g 2 (+ 2 10))

2. [5 marks] Filename for submission: a2-2.scm
Write a procedure **(topsyturvy2-4 lst)** that swaps the second and fourth elements of every list contained in **lst** that has at least four elements. The resulting list is returned by the procedure. Note that **lst** can be a nested list.

Example: (topsyturvy2-4 'foo) returns foo.

Example: (topsyturvy2-4 '(a b (c d) (e f g h) i))) returns (a (e h g f) (c d) b i)

Example: (topsyturvy2-4 '((a b c d) (e f g h))) returns ((a d c b) (e h g f))

3. [5 marks] Hand in answer on paper.
What does the following procedure do? To answer this question please annotate the procedure with comments. Do not forget to include Pre- and Post-conditions and descriptions of arguments and the return value. Give a meaningful sample call to this procedure.

```
(define (unknown a-list c1 c2 c3)
  (cond ((null? a-list) (list c1 c2 c3))
        ((number? (car a-list)) (unknown (cdr a-list) (+ c1 1) c2 c3))
        ((symbol? (car a-list)) (unknown (cdr a-list) c1 (+ c2 1) c3))
        ((list? (car a-list)) (unknown (cdr a-list) c1 c2 (+ c3 1)))
        (else (unknown (cdr a-list) c1 c3 c3))
  )
)
```

4. [15 marks] Filename for submission: a2-4.scm
Write a predicate procedure **(seek-mem? a lst)** where **a** is an atom and **lst** is a list, that returns #t if **a** is a member of **lst** and () otherwise. Note that **lst** may be a nested list. Do not use the built-in procedure member.

Example (seek-mem? 5 '(((a b 5 (x y))))) returns #t

Example (seek-mem? 2 '(((a b c d e)))) returns ()

5. [15 marks] Filename for submission: a2-5.scm
Write a predicate procedure **(palindrome? lst)** where **lst** is a list, that tests to see if **lst** is a flat list and if it is that has the same sequence of elements when it is read from right to left as when it is read from left to right. You may **not** use the built-in Scheme procedure **reverse** as a helper procedure but you can define your own helper procedures.

Example: (palindrome? '(n o o n)) returns #t
Example: (palindrome? '(n o n o)) returns ()
Example: (palindrome? '(a d a f f o d l l s l l d o f f a d a)) returns #t

6. [30 marks] This question has 5 parts. Filename for submission of all parts of this question: a2-6.scm
a) Write a procedure **(prime-fact n)** that returns the prime factorization of an integer **n** > 1. (See http://mathworld.wolfram.com/PrimeFactor.html for a description of prime factorization. 1 is not a prime number.) The output should be a list whose elements are the prime factors, listed in non-decreasing order. Note that if a number is used **m** times in the factorization, it should be listed **m** times, as illustrated in the examples below.

Example: (prime-fact 60) returns (2 2 3 5)

Example (prime-fact 9) returns (3 3)

b) Write a procedure **(count-facts n)** that returns the number of prime factors of an integer **n** > 1.
Note that if a number is used multiple times in the factorization, it should be counted each time, as illustrated in the examples below.

Example: (count-facts 60) returns 4

Example: (count-facts 9) returns 2

c) Now write a procedure **(count-dist-facts n)** that returns the number of *distinct* prime factors of an integer **n** > 1. Note that if a number is used multiple times in the factorization, it should be counted only *once*, as illustrated in the examples below.

Example: (count-dist-facts 60) returns 3

Example: (count-dist-facts 9) returns 1

d) Using (count-facts n), write a predicate procedure **(prime? n)** that returns #t if integer **n** > 1 is a prime number and () otherwise.

Example: (prime? 4) returns ()

Example: (prime? 3) returns #t

e) Using the procedures above, write a procedure **(no-primes-to m)** that takes a integer **m** > 1 as input and returns the number of prime numbers up to the number **m**, including **m** if **m** is prime.

Example: (no-primes-to 5) returns 3

Example: (no-primes-to 10) returns 4

END OF ASSIGNMENT

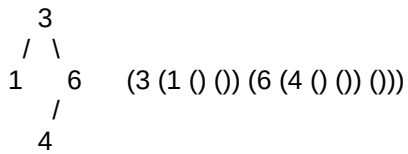
EXTRA PROBLEM – This is NOT part of the assignment. No marks will be assigned.

If you've finished the assignment and would like to continue writing some code, please try the following problem. We won't be marking your solutions, so please don't hand them in with your assignment. We'll go over this problem in the final Scheme tutorial. Have fun!

Write procedures for binary search trees. We represent a tree as a list (key left right), and an empty tree or subtree as the empty list (). You can assume that the keys (i.e., datum) in the nodes are integers. A binary search tree contains in each node a key that is bigger than all keys in its left subtree and less than all keys in its right subtree.

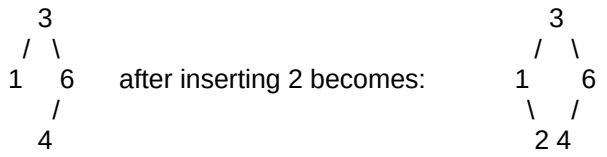
Examples:

empty tree: ()



Implement a procedure called **(bst-insert n bst)**, that takes an integer **n** and a binary search tree **bst**, and returns a new binary search tree with **n** inserted in the appropriate place. If **bst** already contains **n**, then **bst** is unchanged.

Example: (bst-insert 2 '(3 (1 () ()) (6 (4 () ()) ()))) returns (3 (1 () (2 () ())) (6 (4 () ()) ()))



Once you've finished this procedure, think of some other procedures you might write to manipulate the tree and try them out.