Programming Exercise 2

Due Feb 26 by 11:59pm **Points** 50 **Submitting** a file upload

For questions 1-8, write Scheme definitions for the following functions using continuation passing style (CPS) with tail recursion. The continuation argument should be the last argument. Then write the method to call the function with the appropriate continuation:

```
(define factorial-cps
  (lambda (n return)
    (if (zero? n)
        (return 1)
        (factorial-cps (- n 1) (lambda (v) (return (* n v)))))))

(define factorial
  (lambda (n)
    (factorial-cps n (lambda (v) v))))
```

For full marks, all recursive calls of your function should use only the first stack frame.

For questions 9 and 10, write the function using call/cc and a helper function. The helper function should be the only recursive function, and it should use "normal" recursion instead of tail recursion.

You do not have to convert simple scheme built-in functions that do not traverse a list (like null?, eq?, list?, number?, car, cons, cdr) to CPS, but all other helper functions you create should be in CPS.

1. multiplyby takes a number and a list of numbers and returns a list that is the input list with each element multiplied by the input number.
   You can write this function without any helper functions.
   ```
   > (multiplyby 5 '(1 2 3 4 10 11))
   (5 10 15 20 50 55)
   ```
2. crossmultiply takes two lists of numbers, each list represents a vector. Returns the *outer product* of the two vectors. The outer product is a matrix (a list of lists) and each list is the result of multiplying the second list by the corresponding value of the first list.
   ```
   > (crossmultiply '(1 2 3) '(3 0 2))
   ((3 0 2) (6 0 4) (9 0 6))
   > (crossmultiply '(8 -1 4 3) '(3 11))
   ((24 8) (-3 -1) (12 4) (9 3))
   ```
3. maxnumber takes a list of numbers that contains at least one number and returns the largest number in the list.
   You can write this function without any helper functions.
   ```
   > (maxnumber '(3 1 5 2 7 5 3 8 1 2))
   8
   ```
4. partialsums* takes a list that may contain sublists. The output should have the same list structure as the input, but the car of each list (and each sublist) should be the sum of all numbers in that list. There should be no other values in the list.
   You can write this function without any helper functions.
   ```
   > (partialsums* '(1 a 2 b))
   (3)
   > (partialsums* '((1) (2) (3)))
   (6 (1) (2) (3))
   > (partialsums* '((1 2) (10 20) (100 200)))
   (333 (3) (30) (300))
   > (partialsums* '(1 a (10 20 x y) c 2 (x y z) (a b 1 (c 200 d))))
   (234 (30) (0) (201 (200)))
   ```
5. trimatoms takes a list, possibly containing sublists, and a list of atoms. It returns the list of atoms with the first k atoms of the list removed where k is the number of non-null atoms in the first list.
   You can write this function without any helper functions.
   ```
   > (trimatoms '(a b c) '(1 2 3 4))
   (4)
   > (trimatoms '(((a)) (b ((c)))) '(1 2 3 4 5))
   (4 5)
   > (trimatoms '(((a)) () () (b ((c)))) '(1 2 3 4 5))
   (4 5)
   ```
6. exchange takes a list (possibly containing sublists) and a list of atoms. The output should be identical to the first list except that the atoms of the list should be the same as the atoms of the second list, in order.
   ```
   > (exchange '(((a)) (b ((c)))) '(1 2 3))
   (((1)) (2 ((3))))
   > (exchange '((((a a)) b) c) '(1 2 3))
   ((((1 1) 2) 3)
   ```
7. removeallatoms takes a list possibly containing sublists. It returns two lists (collected as pair), the first list is the original list with all non-null atoms remoted, and the second is a list of the atoms that were removed from the original list. You can do this using only the cps version of append as a helper function
   ```
   > (removeallatoms '(a) (b ((c d) e (f))))
   (((((((a)))) a b c d e f))
   ```
8. removesubsequence* takes a list of atoms and a general list. The first list is a *subsequence* of the second list. The method should return the second list with the first occurrence of the subsequence removed. So, if the first list is '(a b c), the first a if the second list is removed, the first b that appears after the removed a is removed, and the first c that appears after the removed b is removed - no matter how deep the atoms are nested.
   ```
   > (removesubsequence* '(a b) '(m (x b) (((a)) ((y z))) b))
   (m (x b) (()) ((y z))))
   ```
   As a hint, you need to keep track of two values between recursive calls. So use two values in the continuation function.
9. **Write the following function using *call/cc* and a single helper function that uses "normal" recursion instead of tail recursion.**
   suffix takes an atom and a list and returns a list containing all elements that occur *after* the last occurrence of the atom.
   ```
   > (suffix 'x '(a b c))
   (a b c)
   > (suffix 'x '(a b x c d x e f))
   (e f)
   ```
10. **Write the following function using *call/cc* and a single helper function that uses "normal" recursion instead of tail recursion.**
    xindex takes an atom and a list containing sublists. The output list should be the same as the input list except that any sublist (including the main list) that contains the given atom should be emptied of all contents (atoms, lists, etc.), and instead, the only content of that sublist should be the index of the first occurrence of the atom in that list.
    ```
    > (xindex 'x '((a b c) (d e x f) ((((g h) i) j) k (((l m x o)))))
    '((a b c) (3) ((((g h) i) j) k ((((3)))))
    > (xindex 'x '((a b c) (d e x g) (((h i) x) j) k (((l m x o)))))
    '(4)
    > (xindex 'x '((a b c) (d e x g) (((h i) x) j k (((l m x o)))))
    '((a b c) (3) (((2) j k (3))))
    ```

Exercise 2 (1)

Criteria	Ratings						Pts
multiplyby	**5.0 pts** **Excellent** Correct with proper use of CPS, tail-recursion and a nice functional style. No unnecessary helper functions is used.	**4.0 pts** **Good** A good tail-recursive, CPS solution with minor errors or unnecessary helper functions.	**3.0 pts** **Reasonable** Uses CPS, but the solution is not tail-recursive or the solution not written in a functional manner (uses let, etc.)	**2.0 pts** **Poor** A scheme solution that does not show an understanding of continuation passing style.	**1.0 pts** **Minimal** Has something in scheme that works for part of the problem.	**0.0 pts** **No Marks** The solution does not show an understanding of scheme.	5.0 pts
crossmultiply	**5.0 pts** **Excellent** Correct with proper use of CPS, tail-recursion and a nice functional style. No unnecessary helper function is used.	**4.0 pts** **Good** A good tail-recursive, CPS solution with minor errors or unnecessary helper functions.	**3.0 pts** **Reasonable** Uses CPS, but the solution is not tail-recursive or the solution not written in a functional manner (uses let, etc.)	**2.0 pts** **Poor** A scheme solution that does not show an understanding of continuation passing style.	**1.0 pts** **Minimal** Has something in scheme that works for part of the problem.	**0.0 pts** **No Marks** The solution does not show an understanding of scheme.	5.0 pts
max number	**5.0 pts** **Excellent** Correct with proper use of CPS, tail-recursion and a nice functional style. No unnecessary helper function is used.	**4.0 pts** **Good** A good tail-recursive, CPS solution with minor errors or unnecessary helper functions.	**3.0 pts** **Reasonable** Uses CPS, but the solution is not tail-recursive or the solution not written in a functional manner (uses let, etc.)	**2.0 pts** **Poor** A scheme solution that does not show an understanding of continuation passing style.	**1.0 pts** **Minimal** Has something in scheme that works for part of the problem.	**0.0 pts** **No Marks** The solution does not show an understanding of scheme.	5.0 pts
partialsums*	**5.0 pts** **Excellent** Correct with proper use of CPS, tail-recursion and a nice functional style. No unnecessary helper function is used.	**4.0 pts** **Good** A good tail-recursive, CPS solution with minor errors or unnecessary helper functions.	**3.0 pts** **Reasonable** Uses CPS, but the solution is not tail-recursive or the solution not written in a functional manner (uses let, etc.)	**2.0 pts** **Poor** A scheme solution that does not show an understanding of continuation passing style.	**1.0 pts** **Minimal** Has something in scheme that works for part of the problem.	**0.0 pts** **No Marks** The solution does not show an understanding of scheme.	5.0 pts
trimatoms	**5.0 pts** **Excellent** Correct with proper use of CPS, tail-recursion and a nice functional style. No unnecessary helper function is used.	**4.0 pts** **Good** A good tail-recursive, CPS solution with minor errors or unnecessary helper functions.	**3.0 pts** **Reasonable** Uses CPS, but the solution is not tail-recursive or the solution not written in a functional manner (uses let, etc.)	**2.0 pts** **Poor** A scheme solution that does not show an understanding of continuation passing style.	**1.0 pts** **Minimal** Has something in scheme that works for part of the problem.	**0.0 pts** **No Marks** The solution does not show an understanding of scheme.	5.0 pts
exchange	**5.0 pts** **Excellent** Correct with proper use of CPS, tail-recursion and a nice functional style. No unnecessary helper function is used.	**4.0 pts** **Good** A good tail-recursive, CPS solution with minor errors or unnecessary helper functions.	**3.0 pts** **Reasonable** Uses CPS, but the solution is not tail-recursive or the solution not written in a functional manner (uses let, etc.)	**2.0 pts** **Poor** A scheme solution that does not show an understanding of continuation passing style.	**1.0 pts** **Minimal** Has something in scheme that works for part of the problem.	**0.0 pts** **No Marks** The solution does not show an understanding of scheme.	5.0 pts
removeallatoms	**5.0 pts** **Excellent** Correct with proper use of CPS, tail-recursion and a nice functional style. No unnecessary helper function is used.	**4.0 pts** **Good** A good tail-recursive, CPS solution with minor errors or unnecessary helper functions.	**3.0 pts** **Reasonable** Uses CPS, but the solution is not tail-recursive or the solution not written in a functional manner (uses let, etc.)	**2.0 pts** **Poor** A scheme solution that does not show an understanding of continuation passing style.	**1.0 pts** **Minimal** Has something in scheme that works for part of the problem.	**0.0 pts** **No Marks** The solution does not show an understanding of scheme.	5.0 pts
removesubsequence*	**5.0 pts** **Excellent** Correct with proper use of CPS, tail-recursion and a nice functional style. No unnecessary helper function is used.	**4.0 pts** **Good** A good tail-recursive, CPS solution with minor errors or unnecessary helper functions.	**3.0 pts** **Reasonable** Uses CPS, but the solution is not tail-recursive or the solution not written in a functional manner (uses let, etc.)	**2.0 pts** **Poor** A scheme solution that does not show an understanding of continuation passing style.	**1.0 pts** **Minimal** Has something in scheme that works for part of the problem.	**0.0 pts** **No Marks** The solution does not show an understanding of scheme.	5.0 pts
suffix	**5.0 pts** **Excellent** Correct, proper use of call/cc with good functional style. No unnecessary helper functions.	**4.0 pts** **Good** Correct with good functional style but either uses call/cc in an inefficient manner. No unnecessary helper functions.	**3.0 pts** **Reasonable** Attempts to use call/cc in a recursive solution, but does not use call/cc correctly or has unnecessary helper functions.	**2.0 pts** **Poor** A scheme solution that does not show an understanding of call/cc.	**1.0 pts** **Minimal** Has something in scheme that works for part of the problem.	**0.0 pts** **No Marks** The solution does not show an understanding of scheme.	5.0 pts
xindex	**5.0 pts** **Excellent** Correct, proper use of call/cc with good functional style. No unnecessary helper functions.	**4.0 pts** **Good** Correct with good functional style but either uses call/cc in an inefficient manner. No unnecessary helper functions.	**3.0 pts** **Reasonable** Attempts to use call/cc in a recursive solution, but does not use call/cc correctly or has unnecessary helper functions.	**2.0 pts** **Poor** A scheme solution that does not show an understanding of call/cc.	**1.0 pts** **Minimal** Has something in scheme that works for part of the problem.	**0.0 pts** **No Marks** The solution does not show an understanding of scheme.	5.0 pts
						Total Points: 50.0	