2009 November 30

Functional
Example Test Questions

1.

You are given the following two functions, which you do not have to implement, where # is the length function, and list member counting begins at 1.

```
prefix(p, list) = list[1 .. min(p, #list)]
suffix(q, list) = list[max(1, q) .. #list]
```

Write a functional program `sublist-all(p, q, list-of-lists)` that returns a list of the sub-lists of each of the lists in `list-of-lists`. The definition of a sublist is the following.

```
sublist(p, q, list) = list[max(1,p) .. min(q, #list)]
```

Example: (sublist-all 2 4 '((a b c d e) (a b c) (a) ((a) (b) (c) (d)))
⇒ ((b c d) (b c) nil ((b) (c) (d)))

The definition is to have no explicit recursion, no lambda, and it is have only one pass over the list of lists.

2.

What is functional programming? What are the prime attributes of functional programs?

3.

Use the following definition of **isMember** for Parts A and B. Higher value is given definitions that do not use support functions or lambda functions.

```
(defun isMember (anItem theList)
 (cond ((member anItem theList) (list anItem))
       (t nil)))
```

Complete the functional program **union**, with no explicit recursion, to compute the union of two sets. **Use filter(predicate, list)**

`(defun union (set_A set_B)` ;; complete the definition …

4.

B Complete the functional program **intersection**, with no explicit recursion, to compute the intersection of two sets. **Do not use filter**.

`(defun intersection (set_A set_B)` ;; complete the definition …

5.

Given a list of integers we can define the first differences as the list of successive differences of the integers as in the following example.
(**first-differences** '(1 10 23 5 6 11 2)) → (9 13 –18 1 5 –9)

A. Define, using **mapcar** and no explicit recursion, the function **first-differences** that returns the list of first differences of the list **int-list**. You may find the function (**butlast** 10 20 30 40 50) → (10 20 30 40) useful.

`(defun first-differences int-list)` ;; You supply the rest

B. Define, using **maplist** and no explicit recursion, the function **first-differences** that returns the list of first differences of the list **int-list**. You may define support functions with no explicit recursion. You may find the function (**butlast** 10 20 30 40 50) → (10 20 30 40) useful.

```
(defun first-differences int-list)  ;; Need for last item in maplist
```

6.

An arithmetic progression is a sequence of the form r, r + s, r + 2*s, … , r + n*s, for some integer r and some positive integer s. By definition, if the length of the sequence is less than three, its elements form an arithmetic progression. The first differences between successive terms is a list consisting of the constant s, and the second differences is the list consisting of the constant 0.

(2, 5, 8, 11, 14, 17) is an arithmetic progression with r = 2, s = 3 and n = 5
(3, 3, 3, 3, 3) is the list of first differences
(0, 0, 0, 0) is the list of second differences

A. Define, with no explicit recursion, the function **is_arithmetic_progression** that returns **t** if the sequence of integers is an arithmetic progression; otherwise it returns **nil**. Use **cond** for conditional statements. Use the function **first-differences** from Question 7 and the fact that every element of the second differences is equal to zero.

```
(defun is-arithmetic-progression (int-seq)   ;; You supply the rest
```

B. Define, with no explicit recursion, the function **is_arithmetic_progression** that returns **t** if the sequence of integers is an arithmetic progression; otherwise it returns **nil**. Use **cond** for conditional statements. This time generate an arithmetic progression based on the first two terms of **int-seq** and its length, and then verify the generated sequence is equal to **int-seq**.

```
(defun is-arithmetic-progression (int-seq)   ;; You supply the rest
```

7.

A. Define the following function to generate a list containing the first **n** terms of the cosine expansion, where **x** is a real number and **n ≥ 1**. Do not define support functions; use lambda-functions instead. You are given the **factorial** function '**!**'. Use **(genlist start next length)** to get the numerators. Use **(range from to)** to get the denominators. Then combine the two lists.

$$\cos ine(x,n) = \sum_{i=0}^{n-1} \frac{(-1)^i x^{2i}}{(2n)!}$$

(defun cosine-terms(x n) ;; You supply the rest

B. Your **genlist** expression in part A should use a lambda function. Define a support function equivalent to your lambda function. Rewrite your **genlist** expression to use the support function instead of the lambda function.

8.

B Write a Lisp functional expression that creates the following set (list) of items.
You must use **(genlist start next length)**. Use a lambda function for **next**. Use only the fours standard arithmetic operators +, –, * and \.

$$\{p : 1..20 \bullet (-1)^p y^{-2p}\}$$

C Your **genlist** expression in Part A should use a lambda function. Define the support function **next_term** that is equivalent to a correct lambda function. Rewrite your **genlist** expression to use the support function instead of the lambda function.

9.

A. Define the following function to generate a list containing the first **n** terms of the **gortz** expansion, where **x** is a real number and **n ≥ 1**. Do not use explicit recursion. Do not define support functions; use lambda-functions instead. Use **(genlist start next length)** to generate the

denominators.
Use **(range from to)** to generate the numerators. Then combine the two lists. Assume the function **foobar** is available.

$$gortz(x,n) = \sum_{i=1}^{n} \frac{foobar(3i-1)}{(-1)^i x^{3i}}$$

(defun gortz-terms (x n) ;; You supply the rest

B. Your **genlist** expression in Part A should use a lambda function. Define a support function equivalent to your lambda function. Rewrite your **genlist** expression to use the support function instead of the lambda function.

10.

A Write a Lisp functional expression that creates the following set (list) of items. You may not use lambda functions or support functions. You must use **(range from to)** only once and pass through the range only once. Assume the functions **foobar** and **(exp x n)** ≡ x^n are available.

$$\{p : -6..20 \bullet foobar((3 - p * p)^5)\}$$

11.

B Write a function, FUNSEARCH, using only functionals (no recursion) to return the first item of every sublist in a list of lists where the sublist contains a particular member, otherwise return the sublist. You may use lambda expressions and Lisp's member function. The following is an example.

```
theList = ((1 2 3 x) (4 5 x 6) (7 8 9) (10 11) (x 13 14 15))
(funsearch 'x theList) ==> (1 4 (7 8 9) (10 11) x)
```

12.

C Write a functional program, `compress(list1 list2)`, (no explicit recursion) that uses a lambda function to produces the sum of the pair-wise subtraction of the smaller numbers from larger numbers.

Example `(compress '(10 20 30 40) '(5 21 33 39))` → `10`

13.

In Lisp write a functional program, `replace`, (no recursion) that uses a lambda function to replace with nil every item at the top level of a list that is a list.

```
(replace '(1 () 2 nil 3 (a b) 4 (a (b) c) 5))
     ⇒ (1 nil 2 nil 3 nil 4 nil 5)
```

14.

A Using the following function `isMember`, write the functional program `intersection` to compute the intersection of two sets.

```
(defun isMember (anItem theList)
(cond ((member anItem theList) (list anItem))
      (t nil)))

(defun intersection (set_A set_B) ;; complete the definition …
```

B Give the above definition of `intersection` using a lambda expression to replace the call to isMember.

15.

B Write a functional program, `checktype1`, that given an input list, returns a corresponding list of T or nil depending upon whether or not an item at the top level of the list is an atom (t) or a list (nil). The empty list is to be considered as a list by the program. Do not use a support function. Use a lamda expressions. You may not use listp

Example `(checktype1 '(a b () (c) d)` → `(t t nil nil t)`

C Write a functional program, `checktype2`, that given an input list which is a list of lists, returns a list of lists where each sublist is a corresponding sublist of t or nil depending upon whether or not the second level items are an atom (t) or list (nil).). The empty list is to be considered a list the program. You may not use listp but you may use a support function to get the simplest program.

Example `(checktype2 '((a) (b () c) (() (d) ((()))) (e))`
→ `( ((t) (t t) (nil nil) nil) (t))`

16.

Define a functional function `sigma2` with no explicit recursion that generalizes `sigma` to two integer indices. It should be able to compute the following expression. Note that the function `term` has two parameters i and j. You cannot use `sigma` because `sigma` accepts a `term` with only one argument. You cannot use any explicitly recursive functions. Assume + is a binary operator. You may assume the following functionals are available – allpairs, bu, curry, comp, compl, distl, distr, filter, genlist, mapcar, maplist, range, reduce, rev, trans.

$$sigma2(term,k,l,m,n) = \sum_{i=k}^{l} \sum_{j=m}^{n} term(i,j)$$

17.

A Define the following function to generate the first n terms of the sine expansion. where x is a real number. Use lambda-functions where necessary. Do not define support functions. You are given the `factorial`, !, function. Use `(genlist length next start)` to get the numerators. Use `(range from to)` to get the denominators. Then combine the two lists.

$$\text{sine}(x, n) = x - x^3/3! + x^5/5! - x^7/7! + \ldots + x^{2n-1}/(2n-1)!$$

```
(defun sine-terms(x n) ;;  You supply the rest
```

B Your genlist expression in part A should use a lambda function. Define a support function equivalent to your lambda function. Rewrite your genlist expression to use the support function instead of the lambda function.

18.

Define a functional program, with no explicit recursion, that produces the list of the sum of the integers i..max, for i = 1 .. max. Do not use lambda-functions.

```
(fun-sum 3) → (6 5 3) ≡   ((+ 1 2 3) (+ 2 3) (+ 3))
(fun-sum 4) → (10 9 7 4) ≡   ((+ 1 2 3 4) (+ 2 3 4) (+ 3 4) (+ 4))
```

```
(defun fun-sum (max)   ;;  You supply the rest
```

19.

A palindrome is a list that reads the same backward and forward. For instance (a b c b a) is a palindrome, but (a b c a a) is not. Define, with no explicit recursion, the following function that returns t if the input list is a palindrome; otherwise it returns nil.

```
(defun is_palindrome (the_list) ;; you supply the rest
```

20.

An arithmetic progression is a sequence of the form r, r + s, r + 2*s, … , r + n*s, for some integer r and some positive integer s. For example (2, 5, 8, 11) is an arithmetic progression with r = 2, s = 3 and n = 3. By definition, if the length of the sequence is less than three, its elements form an arithmetic progression.

Define, with no explicit recursion, the following function that returns t if the sequence of integers is an arithmetic progression; otherwise it returns nil Use cond for conditional statements. You may define support functions using functionals with no explicit recursion. You may find the functions (butlast 10 20 30 40 50) → (10 20 30 40), and length useful.

21.

Define a functional program, with no explicit recursion, that produces the list of the partial factorial values `max!/i!`, for i = 0 .. max-1. Do not use lambda-functions. You may assume **factorial** is available.

```
(part-factorial 3) → (3!/0! 3!/1! 3!/2!) → (6 6 3)
(part-factorial 4) → (4!/0! 4!/1! 4!/2! 4!/3!) → (24 24 12 4)

(defun part-factorial (max)  ;; You supply the rest
```