July 23^{rd} 2004 CMSC 10500-1 Final Exam Duration: 3 Hours

Instructions

1. There is a total of 11 problems worth 120 points.
2. You will be marked out of 60 points.
3. You may solve as many problems as you wish.
4. Name your functions exactly as given. You may name any additional functions as you please.
5. You can write the functions using whatever method you want (recursion or higher order functions), unless specifically stated otherwise.
6. Before you start answering the questions, execute

   `/home/gmkrishn/setup`

   - it will create a directory called "scm-final" in your home directory,
   - create files q1.scm, ..., q11.scm (in scm-final) and launch DrScheme.
   - These files will already have the basic definitions and the some tests as well.
   - To execute the tests, just execute `(run-test n)` where `n` is the number of the test. For each test, you will see a `true` if you passed the test, and a `false` otherwise.
7. In some problems there are many ways to represent the correct solution. In those cases, the problem statement tells you which of the different representations you should use. Make sure that your programs produce the correct representation. If your program is correct but does not produce the correct representation, you will be penalized 2 points.
8. After you have finished the exam, save your programs and quit scheme. Then type the following in the unix shell

   `/home/gmkrishn/submit`

   This will email me all your programs. Confirm with me that I have received your email.

Good Luck with the final exam

Scheme builtins

Here are some scheme builtin's which might help you in your final exam.

Math

(abs x)	Returns the absolute value of x
(remainder n a)	remainder on dividing n by a
(quotient n a)	quotient on dividing n by a
(even? n)	Is n even?
(add1 n)	returns n+1
(sub1 n)	returns n-1

List Processing

(empty? lst)	Returns true if lst is the empty list
empty	Denotes the empty list
(first lst)	Returns the first element of a non-empty list
(rest lst)	Returns the list obtained by removing the first
(list-ref k lst)	Returns the (k+1)'st element of the list

Higher Order functions

(build-list n f)	Returns (list (f 0) (f 1) ... (f (- n 1)))
(map f lst)	Returns a list obtained by applying f to each element of the list
(filter pred lst)	Returns those elements of lst for which pred holds
(foldr f base lst)	Returns `(f` x_1 `...  (f` x_{n-1} `(f` x_n `base)))`
(foldl f base lst)	Returns `(f (f (f base` x_1`)` x_2`) ...  `x_n`)`
(quicksort lst cmp)	Sorts the given lst using the comparision function given.
(apply f args)	Applies the function f to the given list of arguments.
(compose f g)	Returns the function `(lambda (x) (f (g x)))`

```
> (quicksort (list 3 5 4 6 1 9) < )
(list 1 3 4 5 6 9)
> (quicksort (list (make-posn 3 4) (make-posn 4 2) (make-posn 1 5))
             (lambda (p q) (< (posn-x p) (posn-x q))))
(list (make-posn 1 5) (make-posn 3 4) (make-posn 4 2))
> (apply (lambda (p q) (+ p (* 3 q))) (list 4 7))
25
```

Problems

1. Area of a convex polygon - 10 + 10 points

A convex polygon satisfies the following property:
For any point P inside the polygon, joining P to all the vertices of the polygon splits the inside of the polygon (and only the inside) into triangles.

A polygon is represented in scheme by a list of its vertices in anti-clockwise order. For example the polygon below is represented by

```
(list (make-posn 6 0)  (make-posn 3 3)
      (make-posn -1 2) (make-posn -4 0) (make-posn 0 -4))
```

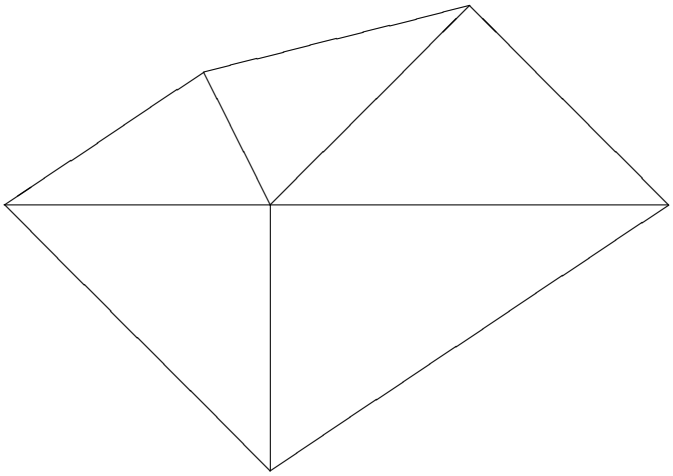
Here the central point (in the figure) is the origin.

Problem Write a scheme function `polygon-area` which consumes a list of posn structures, which represent the vertices of a convex polygon (in anti-clockwise order), and produces the area of the polygon. You may assume that the origin lies inside the polygon.

If the vertices of a triangle has coordinates $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ then the area of the triangle is given by the formula.

$$\frac{1}{2}\,|x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)|$$

where $|z|$ denotes the absolute value of z.



Bonus Problems

- **(5 points)** Rewrite `polygon-area` so that it works for any convex polygon, which does not necessarily enclose the origin.
- **(5 points)** Implement `polygon-area` without using recursion (only higher order procedures). *Note:* This one is a little tricky, though simple. You have been warned.

2. Human Genome - 5 points

Chromosomes are essentially long sequences of DNA (Deoxy-ribo Nucleic Acid). In case of humans a DNA is one of the following: A (Adenine), G (Guanine), C (Cytosine) or T (Thiamine).

Because of the underlying chemical structure of these DNA, A and T have an affinity for each other, and similarly with C and G[1]. In fact the 46 chromosomes in a human cell, come as 23 pairs. Given one chromosome, its complementary pair is obtained by replacing A with T, T with A, C with G, and G with C, respectively.

Problem We will represent a DNA sequence as a list of the symbols `'A 'T 'C 'G`. Write a scheme function `comp-pair` which consumes a DNA sequence and produces its complementary pair.

```
> (comp-pair empty)
empty
> (comp-pair (list 'A 'T 'A 'G 'C 'A 'T 'C 'G))
(list 'T 'A 'T 'C 'G 'T 'A 'G 'C)
```

3. Genome Project - 10 points

One of the problems in the genome project is identifying DNA sequences in chromosomes. This gains importance as there is a high degree of co-relation between having a certain gene and being susceptible to certain diseases. With our representation of a chromosome, the problem translates into searching for sublists in a list.

Problem Write a scheme function `locations` which consumes two DNA sequences (i.e. lists of symbols) called `seq` and `gene`, and produces a list of positions in `gene` where the given sequence `seq` occurs. Note that different occurrences of `seq` in `gene` may overlap.

```
> (define seq (list 'A 'T 'A 'T))
> (define gene (list 'A 'C 'T 'A 'T 'A 'T 'A 'T 'G 'C))
> (locations seq gene)
(list 4 6)
```

[1] A and T form a triple hydrogen bond, while C and G form a double hydrogen bond

4. Continued Fractions - 10 points

An *infinite continued fraction* is an expression of the form

$$\cfrac{N_1}{D_1 + \cfrac{N_2}{D_2 + \cfrac{N_3}{D_3 + \ldots}}}$$

Many numbers like `(sqrt 2)` have nice continued fraction expansions. Truncating such an infinite fraction to k-terms gives a *k-term continued fraction* which is of the form

$$\cfrac{N_1}{D_1 + \cfrac{N_2}{\ddots + \cfrac{N_k}{D_k}}}$$

Problem Write a scheme function `cont-frac` which consumes two functions from numbers to numbers, `num` and `den` and a number `k` and computes the value of the k-term continued fraction where N_i is `(num i)` and D_i is `(den i)`.

```
> (define (one x) 1)
> (define (two x) 2)
> (add1 (cont-frac one one 20))
1.61803398501735793897314 08...
> (add1 (cont-frac one two 10))
1.41421355164605469430412 82...
> (add1 (cont-frac two two 10))
1.73204903677758318739054 29...
```

The three examples give the golden ratio ($\frac{1+\sqrt{5}}{2}$), $\sqrt{2}$ and $\sqrt{3}$ respectively.

5. Linear Recurrence - 10 + 5 points

A sequence of numbers T_n is defined as:

$$T_n = \begin{cases} a & \text{if } n = 1 \\ b & \text{if } n = 2 \\ c & \text{if } n = 3 \\ \alpha T_{n-1} + \beta T_{n-2} + \gamma T_{n-3} & \text{if } n > 3 \end{cases}$$

where a, b, c, α, β and γ are some numbers.

Write a scheme function `linrec` which consumes three arguments: a list of 3 numbers (`initial`) containing a, b and c (in that order), a list of 3 numbers (`coeff`) containing α, β, γ (in that order) and `n` and produces T_n. Make sure that your function calculates the answers in a reasonable time for moderately large values (up to 4 digit number) of `n` as well.

```
> (define (fib n) (linrec (list 1 1 2) (list 1 1 0) n))
> (fib 13)
233
> (define (test n) (linrec (list 1 2 3) (list 1 1 1) n))
> (test (test 13))
268254778799746837205136502137851327959301960256090327027006394319
936334398746233071935977074664657763741725741708904290119940976571
474638353155339801870601410486852924784520031132080245181694836221
321426230970627430493102514889078840834792775910302445439848430321
502432060997273179757868103891668059410728984051644696297850231781
469864426907669260469765411440065048700993085191183501
```

Bonus Problem

(5 points) Generalize the above program as follows: `initial` and `coeff` are arbitrary lists of numbers of the same length. If k is the length of the two lists, then the sequence T_n is a k'th order linear recurrence. For $1 \leq n \leq k$, the value of T_n is given by the `inital` list. For larger values of n use the recurrence

$$T_n = \sum_{i=1}^{k} \alpha_i T_{n-i}$$

where the `coeff` list contains $\alpha_1, \alpha_2, \ldots, \alpha_k$ (in that order). Note that the original problem corresponds to $k = 3$.

```
> (define (pow3 k) (linrec (list 1 0 0 0) (list 0 0 0 3) (* 4 k)))
> (pow3 6) ; gives 3^n
729
```

6. Equality of Sets - 10 points

`=, boolean=?` and `symbol=?` can only be used to check whether two numbers or boolean values or symbols are the same. Scheme has a more powerful keyword called `equal?`. `(equal? x y)` returns true if `x` and `y` are the same. This works not only for all atomic data but also for structures (it checks if corresponding fields are equal) as well as lists (same length, and equal elements in corresponding positions).

Problem A set is represented by a list containing the elements of the set. Hence the order in which the elements of the set appear in the list is irrelevant. The elements of a set can be of arbitrary type i.e. numbers, symbols, booleans, structures, lists (of arbitrary depth). Hence two sets are equal exactly when the lists representing the sets have exactly the same elements, but possibly in a different order. Write a scheme function `equal-set?` which consumes two sets and checks if they are equal. You may assume that every element appears only once (since we are representing a set).

```
> (equal-set? (list 1 4 'a 3) (list 'a 3 1 4))
true
> (define seta (list (make-posn 5 4) 'a 6)
> (define setb (list 'a (make-posn 4 5)) 6)
> (equal-set? seta setb)
false
```

7. Composing Functions - 10 points

Scheme provides a builtin `compose` which takes 2 or more functions as arguments and returns a function which is the composition of these. For example `(compose sqr add1 (lambda (x) (* 2 x)))` represents the function which on input `y` returns `(sqr (add1 (* 2 y)))`.

Problem Write a scheme function `composition` which takes a list of functions `lof` and a list of multiplicities (which are non-negative numbers) `lom` and returns a function which represents the composition of all the given function (with the corresponding multiplicities).

Assume the following:

- the function compositions make sense, i.e. types match.
- all the functions take one argument.
- the list of functions and the list of multiplicities have the same length.

Just like `compose` your function should apply the functions from the right to the left, i.e. functions towards the end of list get applied earlier.

```
> (define f (composition (list first rest) (list 1 1)))
> (define g (composition (list first rest) (list 1 3)))
> (f (list 1 2 3 4 5 6))
2
> (g (list 1 2 3 4 5 6))
4
```

8. n-tuples - 10 points

Write a scheme function `tuples` which consumes a list of lists and returns the list of all tuples where the first element comes from the first list, second element from the second list and so on.

The scheme function `allpairs` which generates all pairs is given as a starting point.

```
> (tuples (list (list 0 1) (list 'a 'b) (list 'x 'y)))
(list (list 0 'a 'x) (list 0 'a 'y)
      (list 0 'b 'x) (list 0 'b 'y)
      (list 1 'a 'x) (list 1 'a 'y)
```

```
        (list 1 'b 'x) (list 1 'b 'y)
)
```

The list of tuples should be generated in the lexicographic order, i.e. the last element of the tuple should change fastest and the first element should change the slowest, and the coordinate should change in the order in which the elements of the corresponding list are given.

9. Choosing k out of n - 10 points

Problem Write a scheme function `choose` which consumes a list `lst` (of numbers and symbols) and a number `k` and produces a list of all ways of choosing k elements from the list. You may assume that all the elements of the list are either numbers or symbols.

```
> (choose (list 1 2 3 4 5) 3)
(list (list 1 2 3) (list 1 2 4) (list 1 2 5) (list 1 3 4)
      (list 1 3 5) (list 1 4 5) (list 2 3 4) (list 2 3 5)
      (list 2 4 5) (list 3 4 5))
```

Note that `(list 1 2 3)` represents choosing 1, 2 and 3 from the list. Here `(list 1 2 3)` and `(list 1 3 2)` represent the same choice. Ensure that in each choice, the numbers/symbols should appear in the order in which they appear in the list. Also, make sure that the choices themselves are ordered lexicographically.

Hint: Consider all the ways of choosing k elements where the first element is present. It is equivalent to choosing k-1 elements from the remaining list. Similarly choosing k elements where the first element is not present is equivalent to choosing k elements from the remaining list. Be sure to handle the boundary cases ($k < 0, k = 0, k = n, k > n$) where n is the length of the list.

10. Balanced Trees - 10 points

A binary tree is said to be **balanced** if for every node the number of nodes in its left subtree and right subtree differ by at most 1 (i.e. difference should be 0, -1, or +1).

Problem Write a scheme function `balanced?` which consumes a binary tree and returns `true` if the binary tree is balanced.
Hint: Write a helper function which consumes a binary tree and returns `false` if it is not balanced and the total number of nodes in the tree if it is balanced.

11. Binary Search Trees - 10 points

Problem Write a scheme function `isbst?` which consumes a binary tree which contains numbers as data in every node, and checks if the given tree is a binary search tree.
Hint: Tree traversals.