

CMSC 15100, Summer 2005
Exam 1

Name _____

1a (from 3)	
1b (from 2)	
1c (from 5)	
1d (from 10)	
<hr/>	
2a (from 5)	
2b (from 5)	
2c (from 10)	
<hr/>	
3a (from 5)	
3b (from 5)	
3c (from 10)	
3d (from 10)	
3e (from 5)	
3f (from 10)	
3g (from 5)	
3h (from 5)	
3i (from 5)	
<hr/>	
4 (bonus)	
<hr/>	
Total (from 100)	

1 [20 total points]. This question asks you to develop a program that tracks information about superheroes. Superheroes have names, secret identities, and an arbitrary number of superpowers. For example, one superhero is named Superman, has Clark Kent as his secret identity, and can fly, see through walls and shoot heat rays from his eyes. Another is named Batman, has Bruce Wayne as his secret identity, and has no superpowers.

1a [3 points]. Write a data definition and any appropriate structure definitions for superheroes.

1b [2 points]. Make three examples of this new kind of data — make examples from Superman and Batman, and also give one more example superhero.

1c [5 points]. Write a template for functions that process superheroes.

1d [10 points]. Write the function *would-beat?* : *superhero superhero* \rightarrow *boolean*, which returns *true* if the first superhero would beat the second if they fought. For the purposes of this problem, a superhero can beat another superhero if he or she has more powers (thus disregarding decades of comics-fan speculation, I'm sure, but hey — it makes the problem easier).

(You don't need to write the template or data sections over again. Also, remember that whenever you process a recursive piece of information in a structure's field you will need a recursive auxiliary function to help you do it.)

2 [20 total points]. As crime-fighters, superheroes often want to know how many crimes have taken place in the city. Our superheroes fight crime in Trinity City, the city that is subdivided neatly into threes: the whole city is divided into three areas, each of which is divided into three subareas and so on, until it is divided all the way down to individual buildings (with three sides, of course).

If we wanted to give a data representation of Trinity City, we could write it as:

```
;; A TC-region is either:  
;; - a number (representing the number of crimes in one building); or  
;; - (make-area lft cntr right) where lft, cntr, and right are TC-regions.  
(define-struct area (left center right))
```

For example, `(make-area 1 5 4)` is a *TC-region* that has had 10 crimes total. `(make-area (make-area 0 0 0) 3 (make-area 0 1 1))` is a larger region that has had 5 crimes total.

2a [5 points]. Write two more examples of TC-regions, one of which contains at least 6 area structures in it. (Just to make sure we're on the same page: `(make-area 1 5 4)` contains one area structure, and `(make-area (make-area 0 0 0) 3 (make-area 0 1 1))` contains three.)

2b [5 points]. Write a template for functions that process TC-regions.

2c [10 points]. Write the function *total-crimes* : *TC-region* → *number*, which returns the total number of crimes in the whole city. (You don't need to write the template or data definitions over again.)

3 [60 total points]. The following questions pertain to list processing.

3a [5 points]. Write the data definition we have used for lists of numbers.

3b [5 points]. Write a template for functions that process lists of numbers.

3c [10 points]. Write the function $roots : (listof\ number) \rightarrow (listof\ number)$, which returns a list of the square roots each number in its input list. For instance $(roots\ (list\ 1\ 25))$ should be $(list\ 1\ 5)$. (You don't need to write the template or data sections over again. Hint: the function $sqrt : number \rightarrow number$ is a built-in function that finds the square root of a given number.)

3d [10 points]. Write the function *give-raises* : (*listof number*) → (*listof number*), which takes a list of salaries and increases them all by 10%. (You don't need to write the template or data sections over again.)

3e [5 points]. Draw boxes around the differences between your solutions to previous two questions.

3f [10 points]. Write an abstracted function that can do either job when given the appropriate arguments. Only write the new function itself here (see the next questions).

3g [5 points]. What is this new function's contract?

3h [5 points]. Redefine *roots* so that it does nothing but call your new abstracted function with the appropriate arguments but has the same behavior as it did originally.

3i [5 points]. Redefine *give-raises* so that it does nothing but call your new abstracted function with the appropriate arguments but has the same behavior as it did originally.

4 [20 points] (**extra credit**). Recall the function *fold* from class:

```
(define (fold lst base combine)
  (cond
    [(empty? lst) base]
    [else (combine (first lst) (fold (rest lst) base combine))]))
```

Rewrite your new abstracted function so that still has the same contract but uses *fold* as a helper function and does not make any recursive calls itself — that is, the recursion in *fold* should be the only recursion in this definition. (Hint: you will need to use either **lambda** or **local**.)