Preliminary Examination II
Computer Science 312, Cornell University 17 April 2007

Before starting the exam, write your name on this page and your netid on both this page and the next page. Also, be sure to take a cookie because you will need it for the exam.

There are 6 problems on this exam. It is 6 pages long; make sure you have the whole exam. You will have 90 minutes in which to work on the problems. You will likely find some problems easier than others; read all problems before beginning to work, and use your time wisely. The prelim is worth 100 points total. The point breakdown for the parts of each problem is printed with the problem. Some of the problems have several parts, so make sure you do all of them!

This is an closed-book examination; you **may not** use outside materials, calculators, computers, etc.

Do all written work on the exam itself. If you are running low on space, write on the back of the exam sheets and be sure to write (OVER) on the front side. It is to your advantage to show your work—we will award partial credit for incorrect solutions that are headed in the right direction. If you feel rushed, try to write a brief statement that captures key ideas relevant to the solution of the problem.

If you finish in the last ten minutes of the exam, please remain in your seat until the end of the exam as a courtesy to your fellow students.

Name and NetID _____

Problem	Points	Score
1	1	
2	12	
3	30	
4	24	
5	20	
6	13	
Total	100	

NetID only _____

1. Cookie [1 pt]

   What kind of cookie did you take?

2. True/False [12 pts]

   (parts a–f; 2 points off for each wrong answer, 1 point off for each blank answer)

a. ____ A hash table can efficiently implement an immutable set abstraction.

b. ____ If using reference counting, an imperative update to a pointer variable causes updates to two reference counts.

c. ____ Balanced binary tree data structures ensure that every leaf is at the same depth from the root.

d. ____ With a balanced binary tree, it is possible to find all elements in the tree in order in time linear in the size of the tree.

e. ____ If a function is $O(4^n)$, it is also $O(2^n)$.

f. ____ If a function is $O(n)$, it cannot be $\Omega(n^2)$.

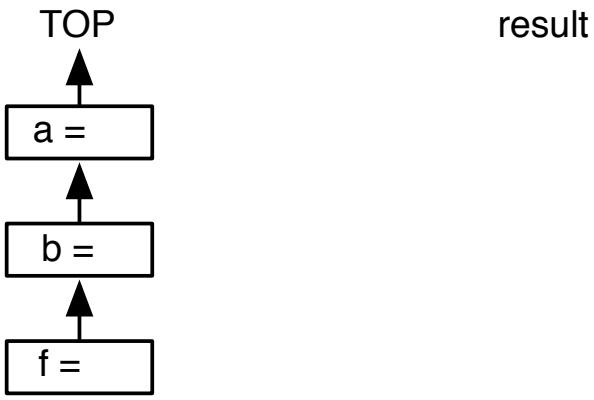
3. Environment model [30 pts] (parts a–c)

Consider the following SML program:

```
let val a: (string->int) option ref = ref NONE
    val b: int->string->int = fn a:int => fn b:string => a + String.size(b)
    fun f(b: int):int = b * ((valOf !a) "hi")
in
    a := SOME(b(3));
    a
end
```

(a) [5 pts] Circle each bound variable occurrence in the above code and draw an arrow pointing to its binding occurrence.

(b) [20 pts] Draw the result produced by evaluating this expression in the environment model, by completing the following diagram. Show all boxes created during evaluation, and draw an arrow from the word "result" to the result value or box.



(c) [5 pts] Draw an asterisk (*) next to any box in the diagram that is allocated on the heap and becomes garbage once the program is done (assume the result is a root).

4. Mutable data abstractions [24 pts] (parts a–d)

Suppose we want to implement a mutable set abstraction, with the following signature:

```
signature MUTABLE_SET = sig
  (* An 'a mset is a mutable set of elements of type 'a *)
  type 'a mset

  (* create(eq) creates an empty mutable set in which elements are
   * compared for equality using eq. *)
  val create : ('a * 'a -> bool) -> 'a mset

  (* add(s, x) adds the element x to s. Returns true if s already
   * contained x, false otherwise. *)
  val add: 'a mset * 'a -> bool

  (* contains(s, x) returns whether s contains x.  *)
  val contains: 'a mset * 'a -> bool
end
```

(a) [5 pts] Given the data structures we have seen, what is the best average-case asymptotic performance (amortized or not) we can expect for the `add` and `contains` operations? Explain your answer briefly.

(b) [5 pts] How might we change the `create` operation to allow a faster implementation? Write a new type and specification for `create` that makes this possible, and give the improved asympotic performance of the other operations.

Suppose we want to implement `MUTABLE_SET` but we already have a good *immutable* set implementation named `Set` handy. This implementation and its signature (`SET`) are defined as follows:

```
structure Set :> SET = struct
  ... (* you don't need to know *) ...
end
signature SET = sig
  (* An 'a set is an immutable set of elements of type 'a *)
  type 'a set
   (* create(eq) creates an empty set in which elements are
    * compared for equality using eq. *)
  val create : ('a * 'a -> bool) -> 'a set
   (* add(s, x) returns (s', f) where s' contains the elements of s
    * and also x. Returns true if s already contained x, false if it did not. *)
  val add: 'a set * 'a -> bool
   (* contains(s, x) returns whether s contains x.  *)
  val contains: 'a set * 'a -> bool
end
```

(c) [10 pts] Show how to implement `MUTABLE_SET` simply, using `Set` and values of the type `Set.set`. Include the abstraction function and rep invariant.

```
structure MutableSet :> MUTABLE_SET = struct
  type 'a mset =




  fun create(eq) =




  fun add(s, x) =




  fun contains(s, x) =




end
```

(d) [4 pts] Suppose that the `MUTABLE_SET` signature also included the following operation:

```
(* copy(s) returns a new set containing the same elements as s.
 * Performance: O(1) time.
 *)
val copy: 'a mset -> 'a mset
```

Show how to extend your `MutableSet` implementation with an implementation of this constant-time operation.

5. Asymptotic complexity [20 pts] (parts a–b)

   (a) [15 pts] Consider this recurrence:

$$T(0) = T(1) = 1$$

$$T(n) = \lg n + T(n/2)$$

   Show that $T(n)$ is $O(\lg^2 n)$. (Recall $\lg^2 x = (\lg x)^2$.) As in the problem set, you may assume that n is a power of two (i.e., $n/2$ is an integer).

   (b) [5 pts] Show that $T(n)$ is also $\Theta(\lg^2 n)$.

6. Memory layout [13 pts]

Suppose that you are implementing the run-time system for a programming language similar to SML. You have available n 32-bit words of memory with addresses ranging from 0 to $4(n-1)$. Each address corresponds to a real memory location (the system does not have virtual memory). The stack grows downward from the top of the address space, and the heap grows upward from some address h_0. Suppose that the minimum size of a heap memory block is 4 words, and the heap and the stack take up h and s words respectively. What is the minimum amount of unused space between the heap and the stack at which you can run a mark-and-sweep garbage collection and be sure that you will succeed? Explain and justify any assumptions you make.