CS312 Preliminary Exam 2
Cornell University, Spring 2005

April 19, 2005

Instructions

Write your name and netid on this page. There are 5 problems on this exam; check now that you have the whole exam. The prelim is worth 100 points total. The point breakdown for the parts of each problem is printed with the problem. 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. This is an closed-book examination; you *may not* use outside materials, calculators, computers, etc. You have 90 minutes. **Good luck!**

Name and NetID ____

Problem	Points	Score
1	20	
2	10	
3	25	
4	25	
5	20	
Total	100	

1. Short Answers [20 pts] (parts a–e)

   For each of the questions below, provide a short answer (1-2 sentences). Ambiguous answers will receive no points.

   (a) [4 pts] Consider an stack abstract data type that provides an `empty()` method to create an empty stack. In order to save space, we'd like to implement `empty()` such that it returns the same heap object every time it is called. When is this approach safe, and when it is not? Explain why.

   (b) [4 pts] What is the amortized run-time complexity of inserting n elements into a standard hash table if the table size is increased by a fixed amount each time it becomes full, and each resizing requires re-hashing all of the elements currently present in the table?

   (c) [4 pts] Why is it better to use variable-sized blocks for memory management, instead of fixed-size blocks?

   (d) [4 pts] Red-black trees are binary search trees that are approximately balanced. Briefly explain what "approximately balanced" means.

   (e) [4 pts] Describe the main disadvantages of garbage collection via reference counting, compared to the other garbage collection techniques.

2. Zardoz Refs [10 pts] (parts a–b)

For the following programs, fill in the box with an expression that causes each program to evaluate to 42.

(a) [4 pts]

```
let
  val f = ref (fn x => x + 1)
  val zardoz = [    ]
in
  (!f)(40)
end
```

Your answer:

(b) [6 pts]

```
let
  datatype node = NULL | PTR of node ref
  fun f(h:node, c:node):int =
    case c of
      NULL => raise Fail "No cycle."
    | PTR p => if !p = h then 1
               else 1 + f(h, !p)
  val zardoz:node = [    ]
in
  f(zardoz, zardoz) * 21
end
```

Your answer:

3. **Complexity** [25 pts]   (parts a–e)

Consider a datatype that describes tree structures. We assume that nodes do not carry values, since those are not relevant to this problem.

```
datatype tree = Leaf | Node of tree * tree
```

We can define a function that computes the height of a tree as follows:

```
fun height(t:tree):int =
  case t of
    Leaf => 0
  | Node(t1,t2) => 1 + Int.max(height(t1), height(t2))
```

Let $T(n)$ be the running time of function `height` for a tree with n nodes and leaves.

(a) [3 pts]   Write a formula $T(1)$ for a tree with one leaf ($n = 1$), and a recurrence relation for $T(n)$, for a tree with $n > 1$ nodes and leaves. Use constants c_1, c_2, etc. for pieces of computation that are $O(1)$.

$$T(1) \;=$$

$$T(n) \;=$$

(b) [6 pts]   Find a solution for your recurrence relation, and prove your result by induction. Make sure you specify what kind of induction you use.

A tree is *balanced* if, for each node in the tree, the height difference of its children is less than 1. The function below uses function `height` to determine whether a tree is balanced:

```
fun balanced(t:tree):bool =
  case t of
    Leaf => true
  | Node(t1,t2) =>
      balanced(t1) andalso balanced(t2) andalso
      (abs(height(t1) - height(t2)) <= 1)
```

(c) [3 pts] Write a recurrence relation for the running time $B(n)$ of function `balanced`, for a tree with n nodes and leaves. For simplicity, assume that the tree is complete, so that all of the leaves are at the same depth. Use constants c_i for pieces of computation that are $O(1)$.

$$B(n) \quad =$$

(d) [3 pts] What is the run-time complexity of function `balanced`? You don't have to prove the formula.

(e) [10 pts] Write another function `balanced2` that has better run-time complexity than function `balanced`.

```
fun balanced2(t:tree):bool =
```

4. **The Environment Model** [25 pts] (parts a–c)

(a) [5 pts] Draw the environment diagram for the following declarations:

```
val x = ref (2, 3)
val y = (x, !x)
```

(b) [15 pts] Now consider a more complicated program:

```
let val x = ref 1
    fun f(y) = ( x := !y + 1; x )
    val x = ref (!x + 1)
    fun g(h) = (h(x), h)
in
    g(f)
end
```

Assume that the language allows recursive function definitions, so the environment pointer in the closure of a function f points to the environment that defines f.

Draw the environment produced by the evaluation of the above program. Indicate the result value and the current environment right before variables start being popped off the environment.

(c) [5 pts] Indicate the heap cells can be reclaimed by the garbage collector at the end of the program from part (b).

5. Types [20 pts] (parts a–c)

(a) [5 pts] Match each of the following expressions with their corresponding types.

```
1. fn x => fn (y,z) => y              A. 'a -> 'b -> ('b * 'a)
2. fn x => fn y => (y,x)              B. 'a -> ('a -> 'b) -> 'b
3. fn (x,y) => fn z => y              C. 'a -> ('b * 'c) -> 'b
4. fn x => fn y => fn z => y          D. ('a * 'b) -> 'c -> 'b
5. fn x => fn y => y x                E. 'a -> 'b -> 'c -> 'b
```

(b) [5 pts] Write an SML program that would be free of run-time type errors if executed, but which fails to type-check (that is, SML issues a type error message). Do not use if expressions.

(c) [10 pts] Consider the following SML declarations where types have been omitted:

```
fun f(x,y,z) =
   (y x) x  orelse  z (x 1)
```

What is the type that SML infers for `x`, `y`, and `z`?