CS 2150 Exam 2

Name

You MUST write your e-mail ID on **EACH** page and put your name on the top of this page, too.

If you are still writing when "pens down" is called, your exam will be ripped up and not graded – sorry to have to be strict on this!

There are 6 pages to this exam. Once the exam starts, please make sure you have all the pages. Questions are worth different amounts of points.

Answers for the short-answer questions should not exceed about 20 words; if your answer is too long (say, more than 30 words), you will get a zero for that question!

This exam is CLOSED text book, closed-notes, closed-calculator, closed-cell phone, closed-computer, closed-neighbor, etc. Questions are worth different amounts, so be sure to look over all the questions and plan your time accordingly. Please sign the honor pledge below.

You step in the stream,
But the water has moved on.
This page is not here.

Page 2: Some trees...some hashing

1. [3 points] What is the Big-Omega worst case runtime of inserting into an AVL-Tree? How much cost do the tree rotations contribute to this runtime? Explain your answer.

2. [3 points] Fill in the following psuedo-code for the binary search tree remove method. Assume the BST allows no duplicate entries.

```
BinarySearchTree::remove(TreeNode thisNode, int valueToRemove):
  if(thisNode.value != valueToRemove):
    call remove on left subtree if valueToRemove is smaller
    call remove on right subtree if valueToRemove is larger
  else if(thisNode.value == valueToRemove):
    if(thisNode has 0 children):
      //???




    if(thisNode has 1 child):
      //???




    if(thisNode has 2 children):
      //???




end remove()
```

3. [3 points] Prove that inserting into a Binary Search Tree is $\Omega(log(n))$ in the worst case (i.e., the insert MUST take at least $log(n)$ time worst-case). You may use the fact that for any BST with height h, the number of nodes in the tree n is bounded by $2^{h+1} - 1$ (which we proved in class).

4. [3 points] List one advantage of having a high load factor in a hash table, and one advantage of having a low load factor.

Page 3: Hash Tables

5. [5 points] Suppose we have a hash table that uses the following hash function: $H(x) = \text{ASCII}(x[0])\%\text{TS}$

   In other words, the hash of a string is equal to the ASCII value of the first index character in the string, modded by the table size. Insert the items below into the given table and use **linear probing** as your collision resolution strategy. The ASCII value of the first character is given to you in parentheses next to each string.
   Macbook (M=77), Strawberry (S=83), Donkey (D=68), melon (m=109), apricot (a=97)

INDEX	VALUE
0	
1	
2	
3	
4	

6. [5 points] Do the exact same, except this time use **separate chaining** as your collision resolution strategy.

INDEX	VALUE
0	
1	
2	
3	
4	

7. [6 points] Lastly, provide the worst-case big-theta runtimes for inserting n items into a hash table given each collision resolution strategy. Give your answer as a function of s (the size of the table) and n (the number of elements being inserted into the table). Briefly explain your answer.

Strategy	**Runtime**	**Brief Explanation**
Linear Probing		
Separate Chaining		
Double Hashing		

Page 4: IBCM and Assembly

On this page, you will design a calling convention for *IBCM*. There are several ways you can do this. Please be as precise as you are able, and try to show us that you understand IBCM and calling conventions by describing your answers as clearly and concisely as possible.

8. [1 points] For a free point, give us an overview of your approach in this space. You might want to do this question after you've done the others below.

9. [3 points] Describe how the caller will *pass parameters* to a subroutine and how the callee will *retrieve those parameters*.

10. [3 points] Describe where your method will allocate and store *local variables*.

11. [3 points] Describe how the callee will deliver *return values* back to the caller.

12. [3 points] Does your calling convention support *recursion*? Why or why not?

Page 5: More IBCM and Assembly

13. [6 points] The following C++ function recursively computes the binomial coefficient $\binom{n}{k}$. Fill in the missing instructions so that the assembly is equivalent to the C++. (Note: It is not necessary that you know what a binomial coefficient is; just translate the C++ function to x86)

```
int binomialCoef(int n, int k) {
  if (k == 0 || n == k) return 1;
  return binomialCoef(n-1, k) + binomialCoef(n-1, k-1);
}
```

```
binomialCoef:

  push rbx

  ___________________
                    1
  je    done

  cmp   rsi, rdi

  ___________________
                    2
  dec   rdi

  push  rdi

  push  rsi

  call  binomialCoef

  ___________________
                    3
```

```
  pop   rsi

  pop   rdi

  _____________________
                      4
  call  binomialCoef

  add   rax, rbx

  jmp   end
done:

  _____________________
                      5
end:

  _____________________
                      6
  ret
```

14. [3 points] What does the following x86 function do? (Either give the equivalent C++ code, or describe *in a few words* what it does)

```
foo:
  push rbx
  mov  rax, rdi
  cmp  rdi, 1
  je   foo_end
  mov  rbx, rdi
  dec  rdi
  call foo
  imul rax, rbx
foo_end:
  pop  rbx
  ret
```

Page 6: This page intentionally left blank

You may use this space for scratch work, however **nothing you write on this page will graded**.