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: Miscellaneous

1. [6 points] Answer the following true/false questions:

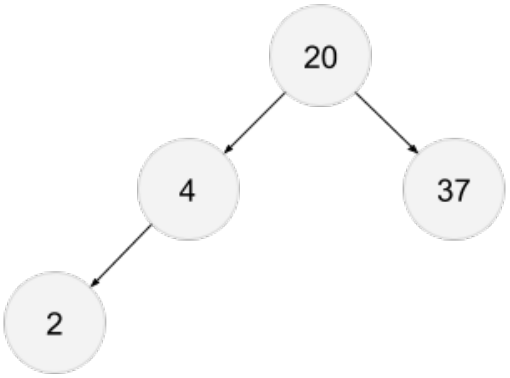
   - T / F: All AVL-Trees are Binary-Search Trees (BSTs)
   - T / F: All Linked-Lists are BSTs
   - T / F: A sorted, doubly-linked list has a better worst-case runtime for findMin() as a BST
   - T / F: Hash functions need to be fast for the hash table to work correctly
   - T / F: Hash tables are usually fast and don't waste any memory space as well
   - T / F: Some things can be computed in x86 but not in IBCM
   - T / F: You can only access the top of the stack in x86
   - T / F: ECX is the lower 4 bytes of the register rcx
   - T / F: RBP is a special register that you should not alter directly
   - T / F: Because RSP is the stack pointer, the register can only be altered indirectly (e.g., push)
   - T / F: call and jmp do the same thing in x86, which one we use is personal preference
   - T / F: x86 distinguishes between pointers and references just like c++ does

2. [4 points] Convert -3.75 into an IEEE 754 Floating Point number. Leave your answer in hexadecimal.

3. [2 points] When implementing an AVL-Tree, do we want to store the balance-factors within the tree nodes, or compute them on the fly? Briefly explain your answer.

Page 3: Trees

4. [4 points] Take the following AVL tree and perform the following three operations on it in order: Insert 1, Insert 3, Delete 4. Draw the tree that results after ALL THREE of these operations have completed.



5. [4 points] Write a short method that finds the maximum value in binary search tree. This method needs to be *recursive*. Your method should not be more than five lines of code.

```
/* TreeNode object contains int val, TreeNode* left, TreeNode* right */
/* Initially invoked as findMaxRecursive(root); root will not be null*/
int AVLTree::findMaxRecursive(TreeNode* curNode){
```

6. [4 points] Write a short method that finds the maximum value in a binary search tree. This time, the method needs to be *iterative* (i.e., no recursive calls). Your method should not be more than five lines of code.

```
/* TreeNode object contains int val, TreeNode* left, TreeNode* right */
/* We've written the first line for you */
int AVLTree::findMaxIterative(){
    TreeNode* curNode = this->root; //assume root will not be null.
```

Page 4: Hashing

7. [5 points] Fill out the best case and worse case runtimes for hash table implementations in the table below. You may assume that the hash table allows for duplicate values to be inserted. All answers should be tight bounds.

Operation	**Best Case**	**Worst Case**
insert() with seperate chaining		
insert() with linear probing		
insert() with double hashing		
rahashing with seperate chaining		
rahashing with linear probing		

8. [7 points] Suppose we are using the hash function below and performing the operations below on the hash table you see. Draw the resulting hash table after all of the operations are performed. Use quadratic probing as your collision resolution strategy.

```
tableSize = 10
hash(k) = k

insert 25
insert 95
insert 11
insert 29
insert 5
insert 14
insert 68
insert 31
```

Index	**Data**
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Page 5: IBCM and Assembly

9. [3 points] In IBCM, recall that the branch-and-link (BRL) command will jump to the specified address and also put the address of the program counter plus one into the accumulator. What might this command be useful for? Be as precise as you can. *Hint: Think about calling conventions.*

10. [3 points] Write a short IBCM snippet that computes whether or not the value in the accumulator is negative. You can write your code as assembly style lines (e.g., load 1) and can assume you have constants available. You may also use labels as you do in x86 assembly to handle jump locations (e.g., loop: load i). If the value is positive, you should place a 0 in the accumulator. If negative, you should place a 1 in the accumulator.

11. [3 points] Briefly explain why we have *calling conventions*. Why are they useful?

12. [3 points] What does the following x86 method do? *We are NOT looking for a step by step breakdown of what each instruction does. We are looking for the high level problem that this method solves.*

```
func: cmp rsi, 0;
      jmpe done
      sub rsi, 1
      call func
      imul rax, rdi
      jmp end
done: mov rax, 1
end:  ret
```

Page 6: Scrap Page for Work

You may use this page to do scratch work, but NOTHING on this page will be graded.