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: First Exam Stuff

1. [4 points] Suppose I want to find the smallest element in a data structure that stores integers. Which of the following data structures would support a findSmallestElement() method whose optimal asymptotic complexity is $o(n)$ (that is little-oh of n). Circle all that apply:

   - Array
   - Sorted Array
   - Linked-List
   - Sorted Linked-List
   - BST
   - AVL-Tree
   - Red-Black Tree
   - Hash Table

2. [4 points] Convert -122 to an 8-bit two's-complement number. Note that $122 = 64 + 32 + 16 + 8 + 2$.

3. [2 points] Suppose you have an array whose base address is 0x0008 and that the array stores 4-byte integers. Now suppose we'd like to find the element at index 3. What address can we go to and find this element? *Note that the addresses here are in hex.*

4. [2 points] Suppose you have linked-list on a 64-bit machine whose base address is 0x0008 and that the list stores 4-byte integers. Also suppose that the size of the list is 5, and that the list only contains a head pointer (no other member variables). How many bytes of memory does this data structure use?

Page 3: Trees

5. [2 points] What is the binary search tree *ordering property*. Briefly, why is it important?

6. [3 points] Why did we need a reference to a pointer in the AVL tree lab? What was the advantage of using a reference to a pointer?

7. [3 points] Draw the binary search tree that is created if the following numbers are inserted in the tree in the given order: 12 15 3 35 21 42 14

8. [4 points] Write a method in pseudo-code that finds the second largest integer element in an *AVL-Tree*. Your method **MAY NOT** alter the state of the tree (i.e., no removing or inserting nodes), and must be $\Theta(\log n)$

```
int findSecondSmallest(AVLNode * curNode){
  //TODO: YOUR SOLUTION HERE
  // ...
```

```
public class AVLNode{
  int value;
  int balance;
  AVLNode * left;
  AVLNode * right;
}
```

Page 4: Hashing

9. [3 points] Write a "bad" hash function that fails to conform to at least two of the three *properties of a good hash function* we studied in class. Briefly explain why the function you write fails to do so.

10. [3 points] What is the worst-case Big-Theta runtime of *rehashing* a hash table that already contains n elements. Briefly, why? Assume your collision resolution strategy is *quadratic probing*.

11. [6 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 linear probing as your collision resolution strategy. For removes, draw a single line through the item that has been removed.

```
tableSize = 5
hash(k) = 3*k + 2

insert 3
insert 8
insert 10
insert 1
remove 8
insert 5
```

Index	Data
0	
1	
2	
3	
4	

Page 5: IBCM and Assembly

12. [3 points] Consider the following *IBCM* program that multiplies two numbers together and stores the result in the first number. Briefly describe the bug(s) in this code AND fix the code on the left to correct the bug(s).

```
//multiply x and y, store result in x
loop:
  load y
  je done
  load x
  add y
  store x
  load y
  sub 1
  store y
done:
  halt
```

13. [3 points] Implement the following x86 instruction using other x86 commands: *mov rax, rsi*. You may *ONLY* use the following instructions in your solution: push, pop, lea, add, sub, imul, idiv, shr

14. [3 points] Explain in your own words the difference between *lea* and *mov*.

15. [3 points] Consider the x86 method below. The function signature is *bool func(long x);*. What does the method, implemented below, return? *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:
  mov rax, 1;
loop:
  cmp rdi, 1
  jl done
  xor rax, −1
  and rax, 1
  jmp loop
done:
  ret
```

Page 6: Miscellaneous

16. [3 points] Why does the x86 family of processors use a little Endian representation for storing integers, instead of big Endian? In other words, list at least one advantage of using little Endian to store integers.

17. [3 points] Write an appropriate Makefile target (meaning the target name and various commands) to compile foo.cpp (which has a `main()`) and bar.s (which has an assembly subroutine called by `main()`) into an executable named `qux`. Do not use any Makefile variables for this.

18. [3 points] GDB and LLDB contain commands called *up* and *down* that move up and down *frames*. Briefly describe what these commands do. Discuss your answer in the context of the x86 calling convention.

19. [3 points] When using Unix, we often use *pipes*. Briefly discuss what a pipe does.