CS 2150 Exam 2, spring 2016

Name _____

You MUST write your e-mail ID on **EACH** page and bubble in your userid at the bottom of this first 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 – even if you are still writing to fill in the bubble form. So please do that first. Sorry to have to be strict on this!

Other than bubbling in your userid at the bottom of this page, please do not write in the footer section of this page.

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.

If you do not bubble in this first page properly, you will not receive credit for the exam!

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.

_____

_____

_____

_____

Serious error.
All shortcuts have disappeared.
Screen. Mind. Both are blank.

(the bubble footer is automatically inserted into this space)

Page 2: Midterm 1 Stuffs

1. [6 points] Convert the *little-Endian* IEEE 754 value 0x000096c0 to it's base-10 floating point representation. Show your work!

2. [3 points] What is reason we might use call by (or pass by) constant reference?

3. [3 points] Write a snippet of C++ code (just a few statements) that creates a memory hole. Assuming the answer is correct, you will get more points for a shorter answer!

Page 3: C++, arrays, big-oh

4. [3 points] Why do we need n_0 when defining the set big-Oh? Give an example to illustrate this point.

5. [3 points] Explain why the insert method for vector is amortized constant time. Be succinct.

6. [3 points] Briefly, why do we need and use suffix rules in Makefiles? Briefly, what are Makefile targets?

7. [3 points] Suppose we want to add an additional register to IBCM. Briefly describe what modifications to IBCM would be necessary to accomodate two registers instead of one.

Page 4: Trees

8. [6 points] Pseudo-code a method called `evaluate(Node* ptr)` that, given a pointer to the head node of an expression tree, returns the value of the expression. You can assume internal nodes contain a public string called *op* that is either +, *, or - (negation, not subtraction) and leaf nodes contain a public member called "val", a `double` value amount. You may also assume that nodes contain `left` and `right` pointer fields, and that negation nodes contain their one child in the left pointer.

9. [3 points] What is the worst-case runtime of an AVL tree insert operation? Explain your answer.

10. [3 points] List the five properties of red-black trees.

Page 5: Hashes

11. [3 points] Consider the three required properties of a hash function, as discussed in lecture. Fill in the top row (the right-most three boxes, not the left-most one) with those properties (be brief here!)

12. [9 points] For each of the described hash functions below, indicated **by a single check-mark** whether the hash function fulfills each of those properties

Hashing method			
Just return 42 every time no matter what the input is			
Keep track of a global variable. Every time you need a new hash, multiply this value by 102,877 (a prime number) and take the remainder modulo 1,299,827 (another prime number). Update the global variable with this new value and then return the value as the hash. (For your purposes, you can assume that having only 1,299,827 different possible hash values is sufficient.)			
Run a well known hash algorithm (like MD5 or SHA-256) on your input 1,000 times (to get it really well mixed).			
Take the binary representation of the input and return the first 32 bits of this representation as an integer. (Do not worry about Endian-ness problems.)			
Every time you need to hash something, roll two six-sided dice and return their sum. (Note: this isnt a simulation. You physically roll two dice every time you need a hash.)			
Every time you see a particular input for the first time, generate a new unique identifier and store the result in a hash table. Then, when you see the input again, return the value stored in the hash table.			
Split your input into 8-bit chunks. Then return the sum of these 8-bit chunks (as an integer).			
Split your input into 8-bit chunks. Then initialize a variable to 0. For each chunk, multiply the variable by 31 and add the integer value of this 8-bit chunk. Return the final value of the variable as the hash. (Do not worry about endianness problems.)			

Page 6: x86

13. [3 points] What are the modifications to the stack done by the *caller prologue*?

14. [3 points] What are the modifications to the stack done by the *caller epilogue*?

15. [3 points] What are the modifications to the stack done by the *callee prologue*?

16. [3 points] What are the modifications to the stack done by the *callee epilogue*?