CS 2150 Exam 2

You MUST write your name and e-mail ID on EACH page and bubble in your userid at the bottom of EACH page – including this page.

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 forms. So please do that first. Sorry to have to be strict on this.

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

There are 10 pages to this exam – once the exam starts, please make sure you have all 10 pages.

Questions are worth 3, 6, or 12 points, depending on the question length. The first and last page are worth 2 points each – if you fill in the bubble footer, you get those points. The three point questions on this exam should not take more than a line or two to answer – **your answer should not exceed about 20 words.** There are 100 points of questions and 1 hour 45 minutes (105 minutes) to take the exam, which means you should spend one minute per question point – the other 5 minutes are to fill out the bubble footers.

If you do not bubble in a page, you will not receive credit for that page!

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 here:

_____

_____

_____

_____

There are 10 types of people in the world –
those that understand binary and those that don't.

(the bubble footer is automatically inserted in this space)

First midterm stuff

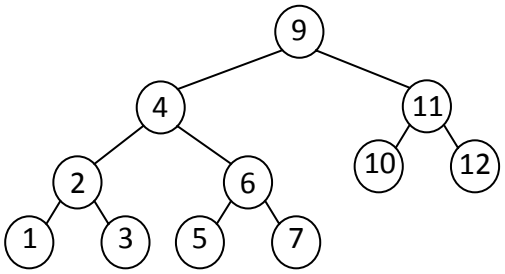
1. [3 points] Other than syntax, what are the differences between pointers and references?

2. [3 points] List everything you know about little-theta.

3. [6 points] Encode -170 as a 32-bit little Endian two's complement integer (note the negative sign on that number). Show you work for partial credit!

Trees

4. [3 points] Insert 8 into the AVL tree below. Show the resulting data structure.



5. [3 points] For the *original* tree listed in the previous question (i.e. not the tree after you inserted a value into it), print out the pre-order, in-order, and post-order transversal of the tree. Label which is which!

6. [3 points] A splay tree runs in $\Theta(\log n)$ *amortized* running time. What does this mean (i.e. the 'amortized' part)?

7. [3 points] What is the algorithm for a removal of a node with two children in a binary search tree?

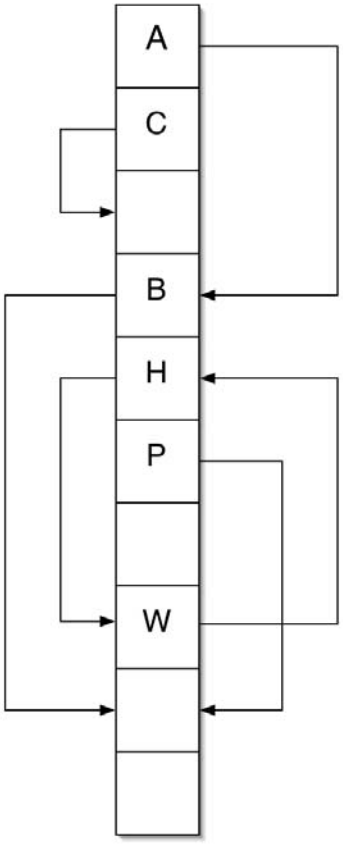
Hash Tables

This page presents a new collision resolution protocol for hash tables. You still get a free 12 points for this page (as long as you bubble in the footer on this page), as this will help you plan your minutes-spent-per-page time appropriately.

Consider a new collision resolution protocol called *Cuckoo hashing*. It is an open addressing scheme – meaning, like the probing strategies, a collision causes the key to be inserted into a different spot in the table rather than into an auxiliary data structure.

The idea behind Cuckoo hashing is that there are only two possible locations that a key can be inserted into: the primary location and a secondary location. Thus, two independent hash functions, which we will designate as $h_1(k)$ and $h_2(k)$, are needed for the location of the primary and secondary locations, respectively. If the primary location is full, then the key is inserted into the secondary location. If that spot is occupied, then the item that was in that spot is evicted, and moved to its 'other' location (which may be either its primary or secondary location). This process repeats until one of two cases occur: either (1) the most recently evicted item find an empty spot, or (2) this eviction cycle goes on for too long. In case (2), the entire table is re-hashed. The name of the hashing algorithm is named after the nesting habits of the European Cuckoos, which will evict others from their nests, causing a chain reaction of nest evictions.

The image to the right shows a hash table with a number of keys inserted into it. Each key has an arrow pointing to its 'other' location. There are 10 spots in the array, although they are not labeled with their index in the diagram. The arrows originate from the cell of the primary hash function for that given value, and terminate at the cell of the secondary hash function.



As an example of Cuckoo hashing, imagine that the primary hash function hashed a key to an occupied spot (it doesn't matter which). The secondary hash function hashes it to the spot occupied by 'A'. Thus, 'A' gets evicted (and in that spot goes the element we are trying to insert), and 'A' moves to its 'other' spot, which is occupied by 'B'. We then evict 'B' and move it to its 'other' spot, which is the second to bottom spot in the table. This illustrates case (1), above, as to how an insert can terminate.

Next, consider the case where the secondary hash function were map a key to the spot occupied by 'H' (we assume the primary hash function still hashed to an occupied spot, which is why the algorithm went to the secondary hash function). In this case, 'H' would be evicted, and move to where 'W' is. Then 'W' would get evicted, and move to where 'H' was (note that element we are trying to insert would be there now). This will form a loop. After a certain number of iterations, the insert fails, and the entire table is rehashed. This illustrates case (2), above, as to how an insert can terminate.

(the bubble footer is automatically inserted in this space)

Hash Tables

8. [3 points] What is the running time of an insert() operation on a hash table that uses Cuckoo hashing as it's collision resolution strategy? Briefly, why?

9. [3 points] What is the running time of an find() operation on a hash table that uses Cuckoo hashing as it's collision resolution strategy? Briefly, why?

10. [3 points] What are the advantages of Cuckoo hashing over the other probing strategies?

11. [3 points] What are the disadvantages of Cuckoo hashing over the other probing strategies?

IBCM

12. [12 points] Subroutines in IBCM are possible via the branch-and-link (`brl`) functionality. We will define an IBCM stack that will, as in x86, grow from the end of memory (location 4095) downward. We want a calling convention for subroutines in IBCM that uses the stack, allows parameter passing, and can properly return to the instruction *after* the `jmp` instruction to the subroutine (recall that `brl` puts the address of the `jmp` instruction into the accumulator). **In particular, your calling convention MUST support recursion!** You need to define this calling convention. As the push and pop operations (which you will presumably use on your stack) are not defined by IBCM, you should show how you would implement those as well. Any IBCM code can be left in opcode form – we aren't looking for machine language here.

---

x86

13. [3 points] What part of the x86 calling convention did Professor Bloomfield present incorrectly this semester?

14. [3 points] How does the C calling convention allow for buffer overflow attacks to occur?

15. [6 points] Draw a diagram (similar to the ones that were shown in lecture) showing a complete activation record for a subroutine. Some of the steps in the calling convention are optional if the subroutine or calling function does not need them – we will assume that, for this diagram, that all these steps are needed.

C++

16. [3 points] Write a **SMALL** code snippet that shows a call-by-reference function. Your function should utilize the feature(s) that the call-by-reference parameter passing concept was designed for.

17. [3 points] What is dynamic dispatch? When is it used?

18. [3 points] How, if at all, does Java support multiple inheritance?

19. [3 points] What is shared multiple inheritance versus replicated multiple inheritance?

UNIX

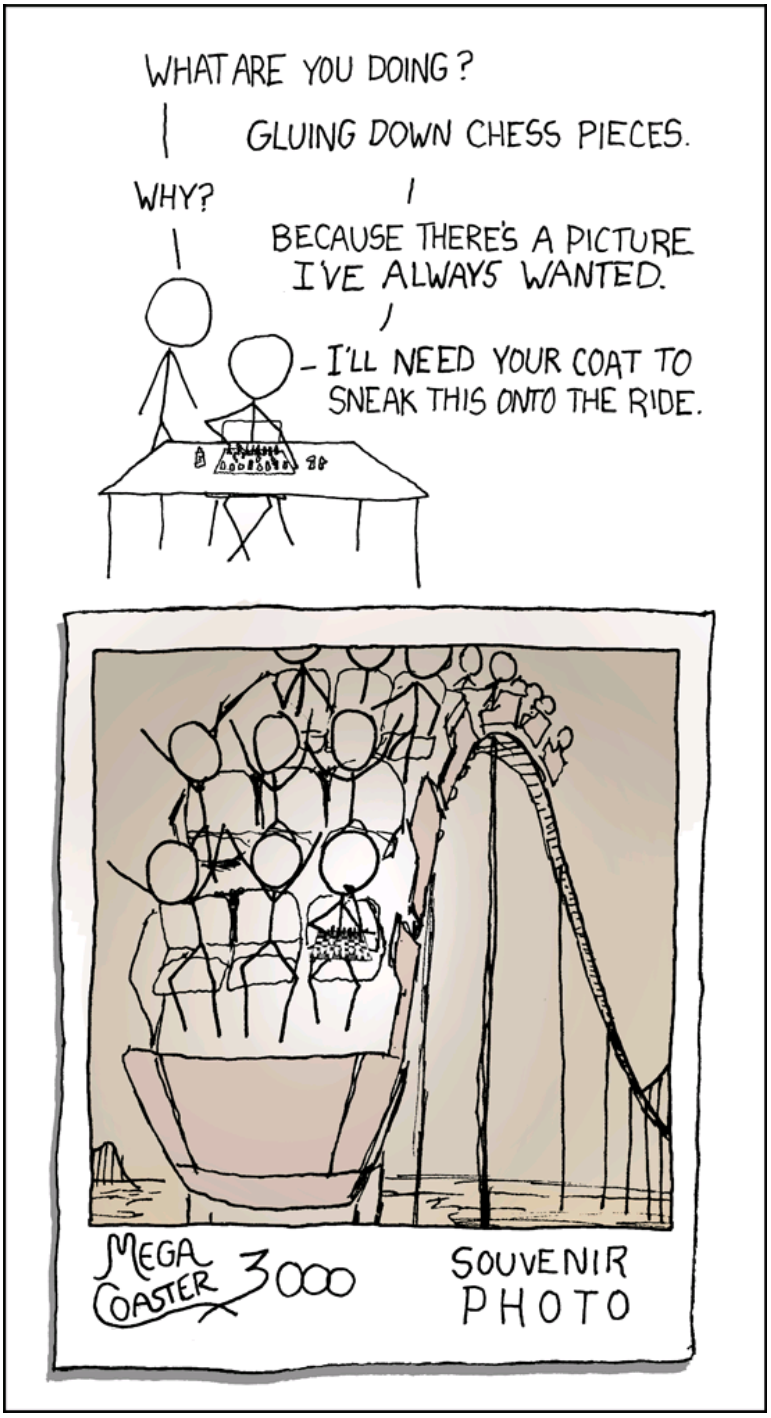
20. [3 points] List all the Emacs keyboard commands that you know, and give a BRIEF description of what each one does (a word or two is fine here). At this point in the semester, you should know at least 8.

21. [3 points] List all the GDB commands that you know, and give a BRIEF description of what each one does (a word or two is fine here). At this point in the semester, you should know at least 8.

22. [3 points] Define the following terms from Makefiles (and briefly describe what they do and/or what they are used for): suffix rules, dependencies, targets.

23. [3 points] What are shell scripts particularly good for? What are they particularly bad for?

You get 2 free points if you fill out the bubble footer on the bottom of this page.



(the bubble footer is automatically inserted in this space)