CS 2150 Exam 1

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 10 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!

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.

_____

_____

_____

_____

_____

The Tao that is seen
Is not the true Tao,
until You bring fresh toner.

(the bubble footer is automatically inserted into this space)

Page 2: C++

1. [3 points] The question you all knew was coming: other than syntax, what are the differences between pointers and references in C++?

2. [3 points] What purpose to templates serve in C++?

3. [3 points] We know that the `#ifndef FOO_H` / `#define FOO_H` / `#endif` pre-processor commands prevent include loops. But how and why does this work?

4. [3 points] Give a compelling use of a union.

Page 3: More C++

5. [3 points] One of the differences between arrays and pointers is that array base names are constant pointers. Why is this the case?

6. [3 points] When writing a C++ class, why do we split the class definition up into a header file and an implementation file?

7. [3 points] Why would we want to pass an object parameter by constant reference?

8. [3 points] What is the purpose of the destructor?

Page 4: Lists

9. [4 points] List two benefits and two drawbacks for implementing a queue data type via arrays.

10. [4 points] What is an abstract data type (ADT)? Why would we use them?

11. [4 points] In the `vector` class, if the vector's internal array (that holds the values) fills up, then the vector has to double the size of that array. How does this occur in C++, when array base names are constants and array lengths are fixed?

Page 5: Numbers

12. [3 points] Convert 234 in base 7 to a base 5 number.

13. [3 points] Write a one line C++ expression to determine the number of *bits* in the `int` type on an arbitrary machine (not necessarily 32-bit machine).

14. [6 points] Consider the statement `int x = -29;`. If you were to examine this statement in a debugger on an x86 machine, what hexadecimal value would be stored in x?

Page 6: More numbers

15. [6 points] Determine the maximum floating point value that can be stored in an IEEE 754 single-precision floating point number. You may leave your answer as a formula (i.e. using multiplication, exponentiation, etc.), as long as all the parts of the formula are actual numbers.

16. [6 points] Convert the value 17.625 ($17\frac{5}{8}$) to IBM floating point notation. IBM floating point notation uses 7 bits for the exponent and 24 for the mantissa, but otherwise encodes just like the IEEE 754 floating point format. You must specify your answer in *little*-endian!

Page 7: Smartlists

While this is a lot to read, you get a free 6 points for reading it.

Your co-worker, Pat Pareto, noticed an interesting characteristic about data when kept in lists: the majority of the accesses were to a minority of the data. For example, 75% of the time the accesses were to 25% of the data, and the other 25% of the time the accesses were to the rest of the 75% of the data. This wasn't an invariant, of course – meaning it didn't always hold. But often it did hold.

So Pat designed a *smartlist*, which is a that re-orders itself at each access. In other words, when an element was accessed, that element was moved somewhere else in the list. Pat came up with a few different means of re-ordering the list:

1. Step forward: after an element is accessed, it is swapped with it's predecessor in the list.
2. Set to front: after an element is accessed, it is moved to the front of the list (the old front of the list becomes the second element in the list).
3. Order by accesses: the number of accesses is recorded, and the list is kept in a sorted order, but sorted by the number of accesses. Thus, frequently accessed elements are nearer to the front of the list.

In each case, the purpose of the movement of the accessed element is to make it "quicker" to access next time, as it is "closer" to the beginning of the list, thus allowing a successive search to take fewer steps.

Your task, for the next few questions, is to analyze this data structure.

17. [6 points] Give one benefit and one drawback of each of the three re-ordering methods.

Page 8: Smartlists, page 2

18. [4 points] Which method will yield the best performance? Why?

19. [4 points] What are the big-Theta running times for the basic list operations (insert, delete, search) for each of the re-ordering methods? Assume the list is a linked list (and not an array). Briefly explain why in each case.

20. [4 points] Will these methods speed up the *average-case* running times for any of the basic list operations? If so, briefly explain which operations, why they are sped up, and by how much.

Page 9: Miscellaneous

21. [3 points] List all the gdb commands that you know, and give a (brief!) explanation as to what they mean. If you hit 8 commands, you're good.

22. [3 points] What does the `chmod` command do? In your answer, it must be clear *why* we would need to do whatever the `chmod` command does.

23. [3 points] Who is your favoriate super hero? No, this is not a trick question.

24. [3 points] Briefly xplain what the UNIX pipes do. There are four of them: '<', '>', '>>', and '|'.

Page 10: Back cover

This page unintentionally left blank.

Figure 1: http://xkcd.com/379/

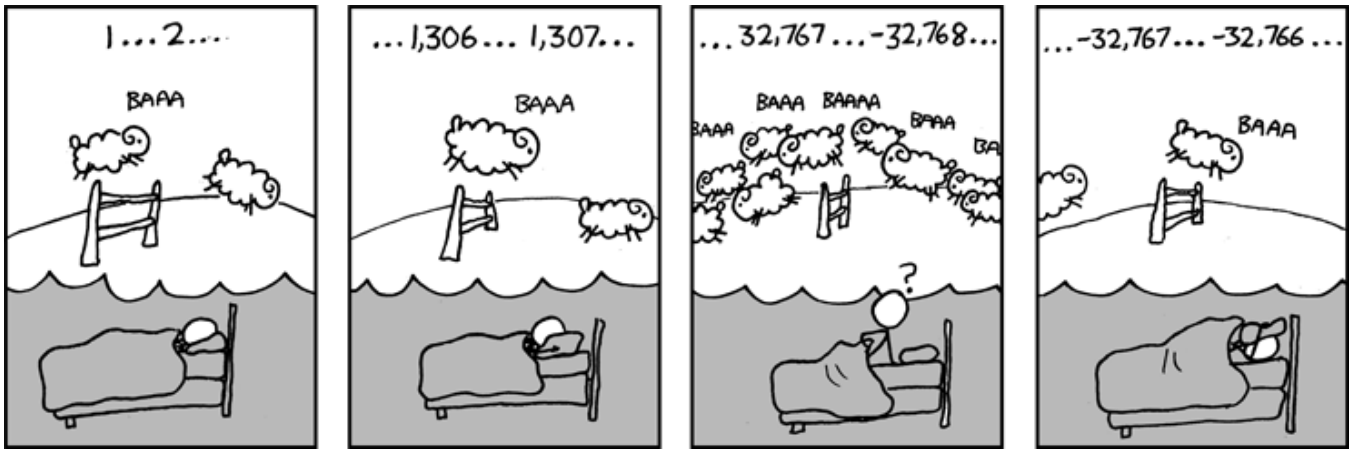

Figure 2: http://xkcd.com/571/