CS 2150 Exam 1

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.

Each page is worth 12 points. Questions are worth varying points depending on the question length. The first and last pages 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 a total of 100 points of questions on this exam, and 1 hour 35 minutes (95 minutes) to take the exam, which means you should spend about one minute per question point.

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 below.

____

____

____

____

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

C/C++, page 1

1. [3 points] Why do we separate out files into separate files: interface (such as Rational.h), implementation (such as Rational.cpp) and the file with the `main()` method (such as TestRational.cpp)?

2. [3 points] Why would you pass an `int*` as a reference to a subroutine?

3. [3 points] Given a class Foo with a single `ListNode* list` field. What is wrong with this constructor?

```
Foo() {
      ListNode* list = new ListNode();
}
```

4. [3 points] Assume the same class (a class Foo with a single field, called list, of type ListNode*). What is wrong with the following constructor?

```
Foo() {
      ListNode temp();
      list = &temp;
}
```

C++, page 2

5. [3 points] Other than syntax, what are the three main differences between references and pointers?

6. [3 points] The copy constructor and the `operator=()` method are similar, but they are invoked at different times. When does C++ invoke the copy constructor, and when does C++ invoke the `operator=()` method?

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

8. [3 points] We have talked about the need to have the #ifndef/#define/#endif pre-processor directives in all of our *.h files. Give an example of a situation that would cause problems if you did NOT include those directives. This can be explained in English or C++ code, your choice.

Lists

9. [3 points] What is an abstract data type?

10. [3 points] Why would you write *method* template? Note that this is not a function template, but a method template.

11. [3 points] Why are list iterators necessary?

12. [3 points] What are the three major operations on a stack, and what are their running times for both an array-based implementation and a linked-list-based implementation?

Deques

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

Consider the data structure called a *deque* or double-ended queue. It is similar to a queue, but allows elements to be added to, or removed from, either the front *or* the back of the list. This is different than a linked list, as a linked list also allows inserts in and removals from the middle of the list as well. A deque only allows inserts and removals from either end.

Implementing a deque as a linked list is easy – it's basically what you implemented for lab 2, only we would have to restrict the inserts and deletes to the ends of the list (the head and tail), and not allow them in the middle. We would keep the `insertAtTail()` method, add an analogous `insertAtHead()` method, and remove both the `insertBefore()` and `insertAfter()` methods.

In addition to the standard methods we would expect in any list (`size()`, `isEmpty()`, etc.), we have two insert methods (`push_front()` and `push_back()`), two removal methods (`pop_front()` and `pop_back()`), and two examination methods (`top_front()` and `top_back()`). There is no way to iterate through the deque, since all operations only occur only on one side or the other.

With a linked list implementation, all of these methods are constant time. Instead, for this example, we are going to analyze deques that are implemented based on an array implementation. You will have to fill in the details of this implementation, as indicated by the questions below.

13. [3 points] Give the big-theta running times for the two insert operations (`push_front()` and `push_back()`), and briefly describe why.

14. [3 points] Give the big-theta running times for the two removal operations (`pop_front()` and `pop_back()`), and briefly describe why.

(the bubble footer is automatically inserted in this space)

Deques, continued

15. [3 points] Give the big-theta running times for the two examination operations (`top_front()` and `top_back()`), and briefly describe why.

16. [3 points] Imagine that a use of your data structure had 4 billion calls to `push_back()` interleaved with 4 billion calls to `pop_front()`. How would you prevent your implementation from running out of space on a 32-bit machine with 4 Gb of RAM (i.e. 4 billion spots in memory) from a large array size? You can assume that no more than 1 million (or so) elements will be in the array at any given time, and that the system can easily hold 1 million elements in memory.

17. [3 points] Considering again the situation in the last question (4 billion calls to `push_back()` interleaved with 4 billion calls to `pop_front()`). Your answer to the previous question was how you would prevent your data structure from running out of space. This question is a bit tougher: how would you do so *efficiently*?

18. [3 points] Given all the problems with the array-based implementation of deques, why would anybody bother with such an implementation over linked lists? Give two reasons.

(the bubble footer is automatically inserted in this space)

Numbers, page 1

19. [3 points] Convert 25_6 (i.e. 25 in base 6) to base 13.

20. [3 points] In Java, the `byte` type is an 8-bit two's complement integer type. What is the maximum value (in decimal) that it can store? What is the minimum value?

21. [3 points] Why did the Patriot missile interceptor fail to intercept the incoming Scud missile on February 25, 1991? What could have been done to prevent this?

22. [3 points] IEEE standard 754 defines a quad-precision floating point type (128 bits, 15 for the exponent, 112 for the mantissa, and about 34 decimal digits of accuracy). But nobody seems to use it or implement in hardware. Why not?

Numbers, page 2

While this is a lot to read, you get free 4 points just for reading it.

A very well-known machine in computing history, the PDP-11, was neither big-endian nor little-endian; instead it was middle-endian. Whereas a big-endian machine would write a number as 1234, and a little-endian machine 4321, a middle-endian machine would write it as 2143.

So one day the PDP-11 met the MIPS processor (MIPS is big-endian). And they couldn't really talk to each other very well over a network. The PDP-11 would encode an IEEE 754 single-precision floating point number in big-endian format, but store the number in middle-endian format. This middle-endian format got transmitted to the MIPS, but the bytes were NOT moved around. So the value was still stored in middle-endian, and was read as if it were big-endian.

In other words, imagine if the value that the value of an encoded floating point number was 0x01020304 in big-endian. The PDP-11 would store it in middle-endian as 0x02010403. When this is interpreted on a big-endian machine (i.e. the MIPS), it would read the same value – 0x02010403 – as a big-endian value, and try to decode that value.

This may sound like a esoteric example. But it is a real problem facing network protocols – in what endian format do you send data from one computer to another?

23. [8 points] If the PDP-11 machine was sending the floating point value for -6.0625 (which is -6 1/16), what value did the x86 machine receive? Show your work!

Arrays & Big-Oh

24. [3 points] In what way(s) can you NOT treat an array base name as a pointer?

25. [3 points] Describe how arrays are stored in memory. An English explanation is fine for this.

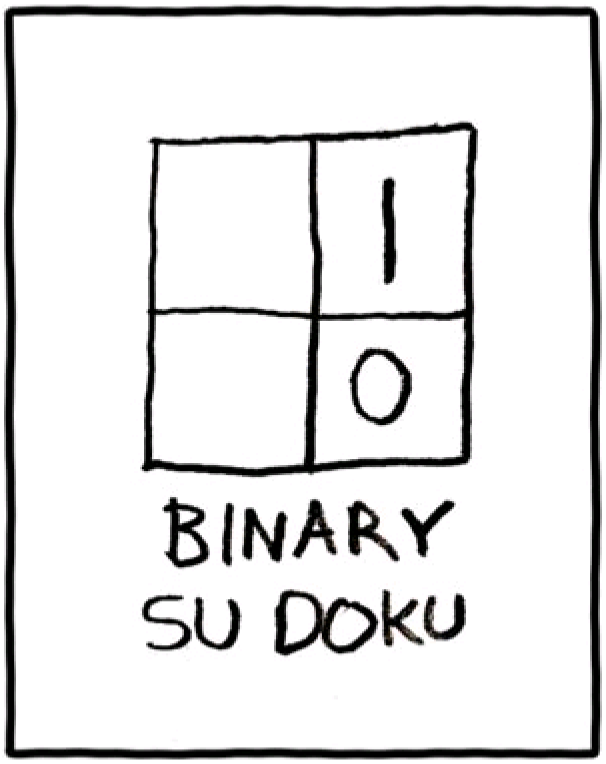
26. [3 points] Describe all you know about little-theta.

27. [3 points] Why do we like big-theta over big-oh and big-omega?

(the bubble footer is automatically inserted in this space)

This page unintentionally left blank.

But you get two points for bubbling in the form at the bottom of this page. Woo-hoo!



Can you finish the binary Sudoku? It's not required, but it's quite a challenge!

(the bubble footer is automatically inserted in this space)