 Name: _____ Email ID: _____

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 do not do this, you will receive a zero for that 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.

Pages 2-7 of this exam contain a combination of medium answer questions (worth 6 points each) and short answer questions (worth 3 points each). The short answer questions should not take more than a sentence or two to answer. The last 3 pages of this exam are the long answer questions, worth 12 points each. There are 99 points worth of questions on this test – you get the extra point for filling out the bubble form at the bottom of this first page, to make 100 points.

This exam is CLOSED text book, closed-notes, **closed-calculator**, 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:

_____

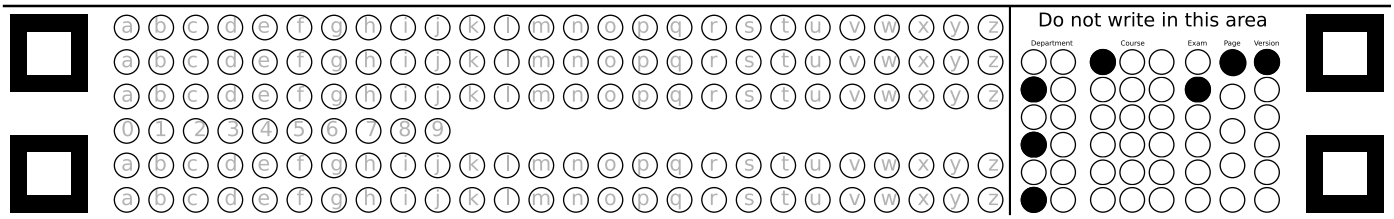
_____

_____

_____

_____

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



 Name: _____ Email ID: _____

Old stuff

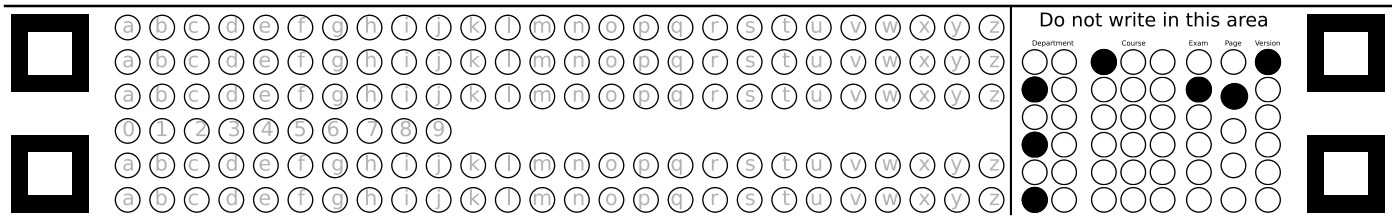
1 [6 points] Consider the following C++ code segment:

```
union foo {
   float f;
   int x;
};
foo bar;
bar.x = 0x40a80000;
cout << bar.f << endl;
```

What is printed? Show your work for partial credit!

2 [3 points] Why isn't there a little-theta?

3 [3 points] Encode -14 as a two's complement binary integer (of 8 bits)



 Name: _____ Email ID: _____

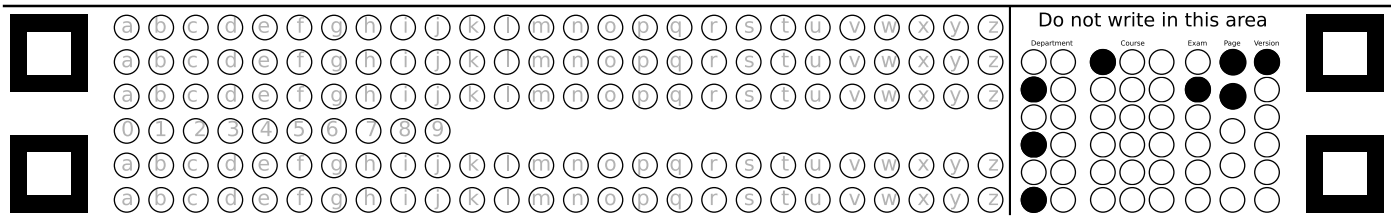
Trees

4 [6 points] Fill in the big-Oh running times in the table below for the following operations on the following trees.

	Insert	Delete	Search	PrintAll
Binary search tree				
AVL tree				
Splay tree				
Red-black tree				

5 [3 points] How does the splay operation of a splay tree work? In other words, how does the splay() routine go about modifying the tree to splay a given value?

6 [3 points] After an AVL tree insert, if the resulting tree is unbalanced, the insertion routine must balance the tree by a series of rotations. How many rotations are needed at a maximum? Why?



 Name: _____ Email ID: _____

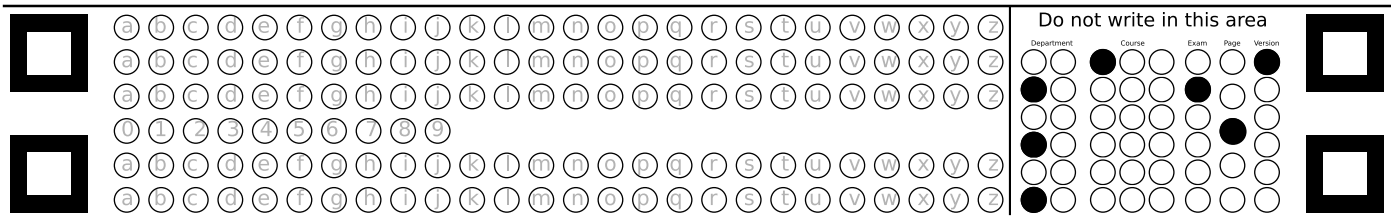
Hash Tables

7 [6 points] Insert the values 13, 84, 75, 97, and 23 into the following hash tables, where the collision resolution method is shown at the top of each table. Let the hash function be $h(x) = x \bmod 10$, and the secondary hash function (if needed) be $i(x) = x \text{ div } 10$.

	Linear probing	Quadratic probing	Double hashing	Separate chaining
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				

8 [3 points] What is the load factor for a hash table? Why is it important? When do we use it?

9 [3 points] What is the big-Oh running time for the following hash table operations: insert, find (a value when given a key), and printAll?



 Name: _____ Email ID: _____

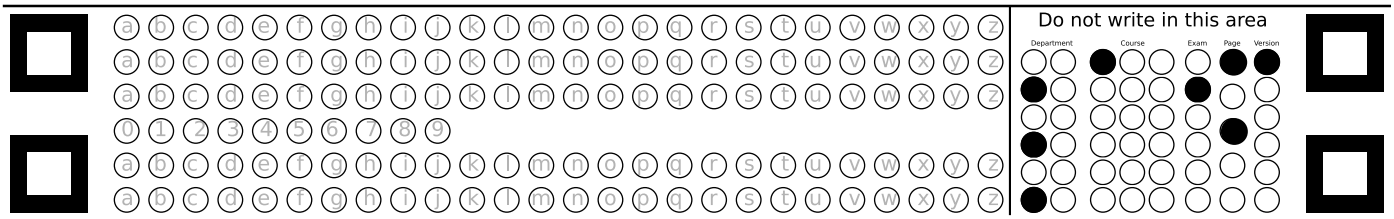
C++

10 [3 points] What are the differences between a reference and a pointer?

11 [3 points] What is the difference between static dispatch and dynamic dispatch? Give an example of when each is used.

12 [3 points] What do private and protected inheritance do? When would we use them over public inheritance?

13 [3 points] What problems exist in multiple inheritance that do not exist in single inheritance? How are they solved?



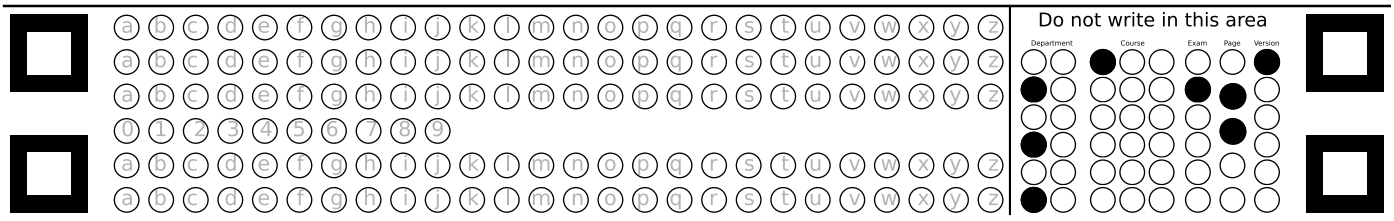
 Name: _____ Email ID: _____

Data structures

14 [3 points] When would we want to use a red-black tree instead of an AVL tree? When wouldn't we?

15 [3 points] When would we want to use a splay tree instead of an AVL tree? When wouldn't we?

16 [3 points] When would we want to use a hash table instead of an efficient tree (AVL or red-black or splay)? When would we prefer an efficient tree?



Hash Tables (again)

17 [6 points] Consider a routine (function, method, whatever) that is passed a hash table full of values – we'll assume, for sake of argument, that the keys are integers and the values are strings (of a maximum length of a fixed, but small, size). Your job is to design this routine, which will do a number of searches through the *values* of the hash table (for these searches, we don't care about the key – we just are doing a search on the value). You can assume that any properties of the hash table (load factor, table size, etc.) are either provided or easily determined (they could be fields of the hash table object, for example). How would you go about writing the routine in as efficient a manner as possible? This should be a high-level overview of what you are going to do – meaning a few sentences of English to describe your algorithm. No C++ code is needed (or even desired) here. Assume that there are *n* elements in the hash table, its load factor is reasonable (say, between 0.5 and 0.75), and that you must perform *m* searches. We do not know if *n* is greater than, equal to, or less than *m*. What is the big-Oh running time of your algorithm (for all *m* searches)? Give your big-Oh answer in terms of *m* and *n*.

`

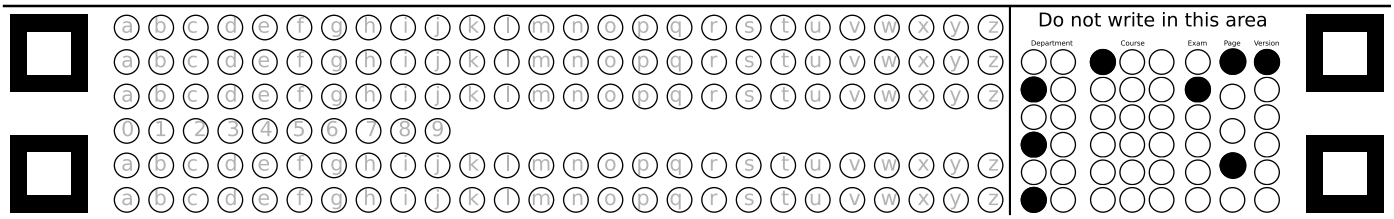
AVL Trees

18 [12 points] Draw an AVL tree that is unbalanced, and in need of a double rotation (either type) for proper re-balancing. Show the balance factors for the nodes, as well as the values in the nodes (they can be any values, but use integers). Show the tree after the first part of the double rotation (with node values and balance factors), as well as when completed (i.e. after the second part of the double rotation).

`

Calling Conventions

19 [12 points] You are familiar with the C calling convention for x86 assembly. For this problem, you will need to design a calling convention for IBCM. It will be used when a subroutine is called through the BRL instruction (branch-and-link, which stores the PC into the accumulator and then jumps to the address in the instruction). You should assume that there could be a return value from the subroutine, (the subroutine could also return void), and that there can be a maximum of 3 parameters. There can also be local variables in the subroutine. When the entire calling convention is completed, the accumulator should have the same value that it had before any of the calling convention started. Describe the four parts of the calling convention – both the prologue and epilogue for both the caller and callee. This can be a high-level overview – meaning you should explain (in a sentence or two) what each step does; you don't have to write any IBCM code. However, in case you want to review the IBCM instructions, they are listed on the following page.



IBCM

20 [12 points] Write a routine to perform subtraction of two integers using IBCM assembly. It should read in two hex values from input, and output the result. You should leave your code in IBCM instructions (i.e. 'jmp foo'), not machine code. Start by writing the high-level pseudocode for how this routine might work; you will get (some) credit just for the pseudocode. Work on the IBCM instructions from there. All of the IBCM instructions are listed to the right.

IBCM

0	halt
1	I/O
2	shifts
3	load
4	store
5	add
6	sub
7	and
8	or
9	xor
A	not
B	nop
C	jmp
D	jmpe
E	jmpl
F	brl