NAME: _____ NETID: _____

CS2110 Fall 2010 Prelim 1
October 14, 2010

*The exam is closed book and closed notes. Do not begin until instructed. You have 90 minutes. Good luck!*

*Start by writing your name and Cornell netid on top! There are 8 questions on 13 numbered pages, front and back. Check now that you have all the pages. When you hand in your exam, make sure your booklet is still stapled together. If not, please use our stapler to reattach all your pages!*

*We have scrap paper available, so you if you are the kind of programmer who does a lot of crossing out and rewriting, you might want to write code on scrap paper first and then copy it to the exam, just so that we can make sense of what you handed in!*

*Write your answers in the space provided. Ambiguous answers will be considered incorrect. You should be able to fit your answers easily into the space we provided, so if something seems to need more space, you might want to skip that question, then come back to your answer and see if you really have it right.*

*In some places, we have abbreviated or condensed code to reduce the number of pages that must be printed for the exam. This does not mean that it's good style.*

We've included one five-point extra credit question, so your total score can reach 105 points. However, P1 contributes Math.Min(P1-score, 100) towards your final grade.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | XC | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| Max | 16 | 20 | 12 | 12 | 10 | 10 | 10 | 10 | 5 | 105 |
| Score | | | | | | | | | | |
| Grader | | | | | | | | | | |

*Extra credit question: For 3 points. Write code to swap two integers X and Y without using any additional memory (e.g. no extra variables). For 2 more points: Do it two ways, once using only bitwise operators, and once using non bitwise operations.*

One way (arithmetic operators):

```
X = X+Y;
Y = X-Y;
X = X-Y;
```

The other way (bitwise operators. ^ is the Java XOR (exclusive or) operator):

```
X = X^Y;
Y = X^Y;
X = X^Y;
```

1. (16 points) Tell us what the following two Java programs will print if each is executed with "63" as an argument. (Hint: `Integer.parseInt` (s), for a string s, converts s to an integer).

(Part a: 8 points)

```
public class TheHulk {

	public static void main(String[] args) {
		int n = Integer.parseInt(args[0]);
		smash(2, n);
		System.out.println();
	}

	public static void smash(int a, int b) {
		if(a <= b) {
			if(b%a == 0) {
				System.out.print("  " + a);
				smash(a, b/a);
			}
			else {
				a = a+1;
				smash(a, b);
			}
		}
	}
}
```

**(8 pts) Output of TheHulk.main():**

**This is a kind of obscure program for factoring its input. For input 63 it will output**

**3 3 7**

(Part b: 8 points)

```
public class Node {
	char content;
	myNode next;

	public myNode(char c, myNode tail) { content = c; next = tail; }
	public void Add(char c) { next = new myNode(c, next); }
}

public class CharList {
	public static Node myList = new Node((char) 0, null);

	public static void main(String[] args) {
		for(String s: args) {
			for(int i = 0; i < s.length(); i++) {
				int c = Integer.parseInt(s.substring(i, i+1));
				for(int n = c; n > 0 ; n--) {
					myList.Add(s.charAt(i));
				}
			}
		}

		for(myNode nd = myList.next; nd != null; nd = nd.next) {
			System.out.print(nd.content + " ");
		}
		System.out.println();
	}
}
```

**(8 pts) Output of CharList.main():**

**This program makes a list in which each character in its input strings appears the same number of times as the character itself (it assumes the character is a decimal number). So, for 63, it adds six '6' characters to the list, and then 3 '3' characters. However, the Add function for this particular list puts the newest character at the front of the list (stack-style). So since the 3's were added last they end up at the front and it prints:**

**3 3 3 6 6 6 6 6 6**

2. (20 points) True or false?

| | | | |
|---|---|---|---|
| a | T | F | After executing "`int x = 1; int y = ++x;`" the final values of **x** and **y** are 2 and 1, respectively. |
| b | T | F | The Java "x **instanceof T**" operator tests to see if the type of **X** is a subtype of **T**. |
| c | T | F | When a class implements an interface, it needs to define every method in the interface. |
| d | T | F | If a recursive method fails to check for the base case(s), it might call itself repeatedly, triggering a stack overflow. |
| e | T | F | If we have a class **Animal** for animals, and then a subclass **Dog** for dogs, the **Dog** class can define additional fields and methods not defined in **Animal**. |
| f | T | F | If we have a **Dog** object and **Dog** is a subclass of **Animal**, then all the methods defined in the **Animal** class will also be defined for the **Dog** class. |
| g | T | F | We use anonymous classes in GUI design as a convenient way to put the code for some operation, like a button click, close to where we created the button. |
| h | T | F | If a class extends some other existing class, then all its methods must be static. |
| i | T | F | If a static method needs to access a static field in an object from some other class, it would be best to qualify it by the class name, as in "`SomeClass.some_field`." |
| j | T | F | If a static method needs to access a static field in some object, the field name should be qualified by the class name, as in "`SomeClass.SOME_FIELD`." |
| k | T | F | Static type checking not only takes time when you compile a program, but also makes Java programs slow when you execute the compiled code. |
| l | T | F | If a method is declared to be private, it can be accessed from outside the class if you first cast the class to public, as in "`x = (public)foo.somePrivateMethod();`" |
| m | T | F | If **Animal** is an interface, and you call "`Animal a = new Animal("Biscuit");`" a new Animal object will be created and the constructor will use ``Biscuit'' to initialize the fields. |
| n | T | F | If a list ends with a null pointer and your code dereferences each node on the list without checking for null pointers, Java will throw a null pointer exception when you reach the end of the list. |
| o | T | F | If Dog is a class and you write "`Dog Biscuit = null;`" then **Biscuit** will be a **Dog** object with every field initialized to null, zero, or the corresponding "empty" value. |
| p | T | F | Suppose Dog is a subclass of Animal, and we execute:<br>`Dog m = new Dog("Midnight"); Animal a = (Animal)m; Dog d = (Dog)a;`<br>The last line of the sequence will fail, because once we turn **m** into an Animal in the second line, we can't change our minds and reverse the transformation this way. |
| q | T | F | Searching for an element in a balanced binary search tree will require exactly log(N) "compare" operations. |
| r | T | F | If it takes exactly $3n^2 + 2n - 1$ comparison operations to compute something, we can say that the complexity of the method, measuring comparisons, is $O(n^2)$ |
| s | T | F | If we are given two methods for computing something, and one has complexity $O(n^2)$ and the other has complexity $O(2^n)$, the $O(n^2)$ will always be faster than the $O(2^n)$ version . |
| t | T | F | If you want to use a "`for (T elem: container)`" statement to iterate over the members of `container`, `container` must either implement the `Iterable<T>` interface or be an array of type U[], where U is a subclass of T. |

3. (14 points) Your new boss at Backus & Naur Foundries Publishing Co. has asked you to produce a million pages of nonsense text that can be used as filler for page layout mockups by the graphic design department. You devise a grammar for generating meaningless sentences that resemble Latin:

**Noun** = *ipsum*| *felis*| *lectus*| *amet*
**Verb** = *sit* |*metus*| *libero*
**Adjective** = *dolor* | *elit* |*sapien* | *a* **NounPhrase**
**Qualifier** = *in*| *sed*| *lorem*
**NounPhrase** = **Noun Adjective** | **Noun** | **Qualifier NounPhrase**
**VerbPhrase** = **Verb NounPhrase** | **Verb**
**Sentence** = **NounPhrase VerbPhrase ,** *et* **NounPhrase VerbPhrase** .
| **NounPhrase VerbPhrase** .
| **VerbPhrase** ?

You feed this into a program you found on the web that takes a grammar as its input, and then prints all the 1 word sentences (if any), then all the 2 word sentences, then all the 3 word sentences, etc. The program stops if it ever manages to print every possible sentence. It prints in a normal 12-point font.

(a) (2 points) Is this grammar likely to be able to fill a million pages without repeating any sentences? Why or why not?

**Answer:** ***Yes, because NounPhrase -> Adjective and Qualifier -> NounPhrase can recurse indefinitely. One point for saying "No, because you can only form a finite number of sentences out of a given set of noun/verb phrases", because it's good reasoning that misses the recursion.***

(b) (4 points) For each of the following, say whether it is a sentence generated by the grammar. Ignore capitalization and whitespace.

| | | | |
|---|---|---|---|
| (i) | Y | N | Lorem ipsum dolor sit amet. |
| (ii) | Y | N | In felis a lectus libero, et lectus sit. |
| (iii) | Y | N | Ipsum sapien, et metus amet? |
| *(iv)* | *Y* | N | Lorem lectus a lorem ipsum a lorem felis sit. |

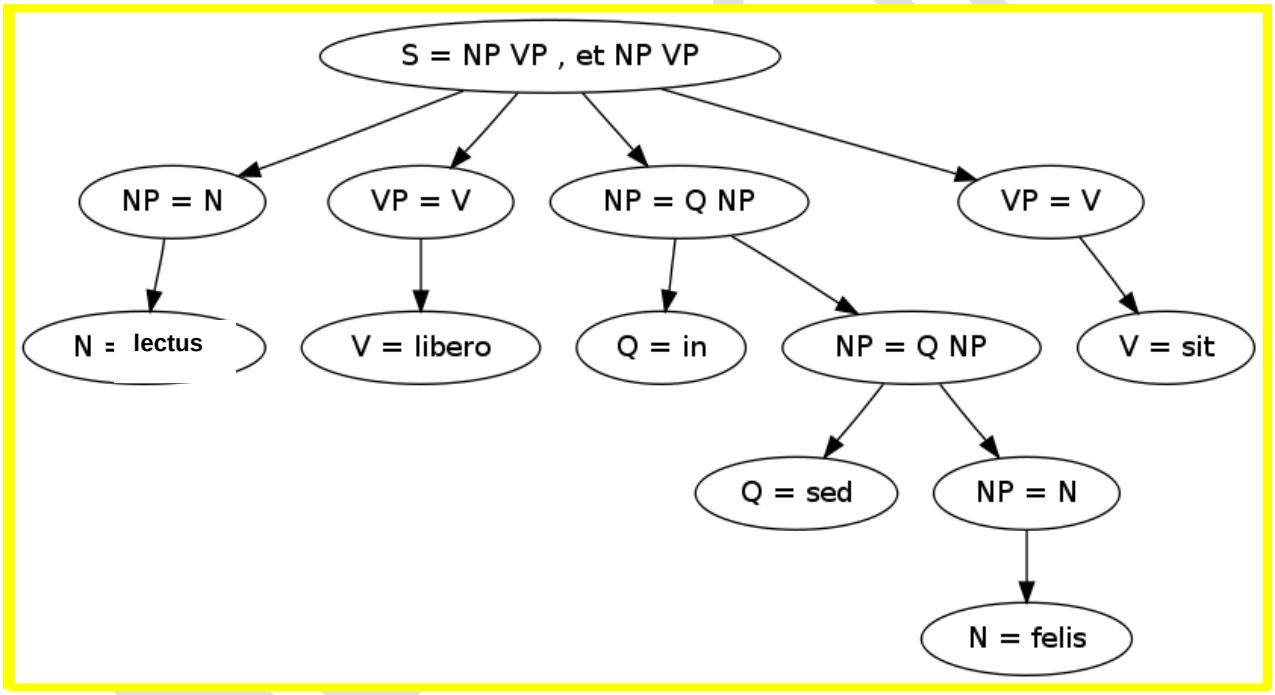
(c) (2 points) Our grammar would be more expressive if we added a new term **Clause** and modified the definition of **Sentence** as follows:

**Clause** = **NounPhrase VerbPhrase** | **Clause ,** *et* **Clause**
**Sentence** = **Clause .** | **VerbPhrase** ?

Could you use a recursive descent parser for the grammar with these new rules? Explain.

***Answer: Clause = Clause , et Clause is left recursion and cannot be handled by an LL / RD parser. The grammar could be modified to eliminate left recursion.***

(d) (6 points) Draw the parse tree generated by a recursive descent parser of "*Lectus libero, et in sed felis sit."* (When labeling nodes, you may abbreviate rule names as S, N, V, NP, etc.).




*Note: We'll accept any tree "identical" to this one; you don't have to use exactly the same notation as we did to get full credit. But your tree does have to have the same nodes and structure because this grammar can only parse the given expression in this specific way!*

4. (10 points) Choose the best alternative.

(a) If class B is a subclass of A, this *must* be fixed before you can compile and run a program:

A. Casting a variable of type A to (B)
B. Casting a variable of type B to (A)
C. Casting a variable of type A to (C), where C is unrelated to A or B

(b) The **`static`** keyword

A. is used to make values constant
B. causes local variables of a method to be reused in subsequent method calls
C. associates field values with a class instead of with an instance

(c) Unit tests

A. should be written as code is developed to test individual pieces
B. guarantee a program is correct
C. are a waste of time for good programmers

(d) **`Integer x = 5;`**

A. needs an explicit cast (`Integer`) because 5 is of primitive type `int`
B. is fine, because 5 is an `Integer`
C. is fine, because autoboxing will convert `int` to `Integer`

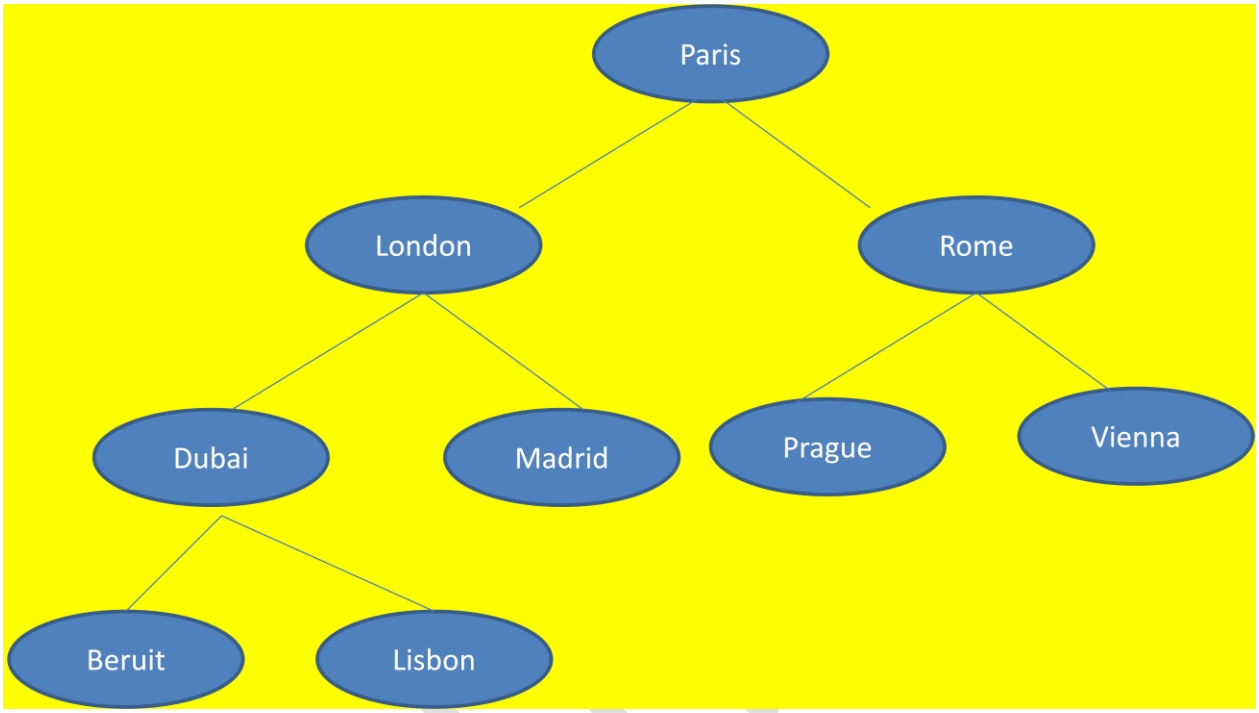
(e) A class might declare a private default constructor

A. to allow the class to be instantiated only by a static method of the same class
B. to prevent the class from being instantiated entirely
C. A and B

(f) In assignment 3, an object of type **`HashMap<Species,String>`** could be used most elegantly and efficiently to:

A. store the list of genes associated with a particular species
B. remember which file a Species was read from
C. keep track of the species that share a given gene

5. (10 points) (a) Draw the binary search tree resulting from inserting the following elements in the following order: *Paris, London, Rome, Vienna, Dubai, Madrid, Lisbon, Prague, Beirut*. Assume that the normal lexicographic (dictionary) comparison ordering is employed.



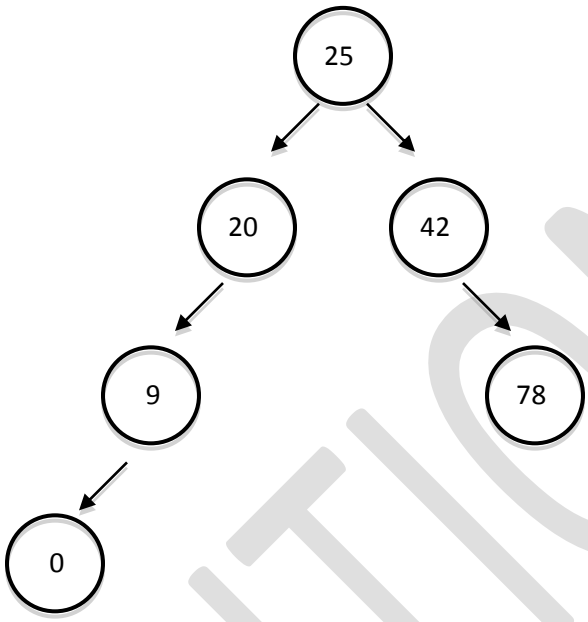

(b) Binary search trees allow us to perform lookups in O(log n) steps. To demonstrate that, write down the elements encountered (in order) while performing a lookup for "*Prague*".

Paris… Rome… Prague

(c) To drive home the difference, let's assume that instead of using a BST the same elements were stored in a list by adding them one by one in the order shown above. Write down the list of elements encountered (in order) while looking up "Prague" in the list created in this way.

*Paris, London, Rome, Vienna, Dubai, Madrid, Lisbon, Prague*

6. (10 points) Binary Search Trees. A BST is *balanced* when the distance between the root and any given leaf node is bounded on top by $\log_2(n)$, where n is the number of nodes in the tree. Balance is important because we can make sure that all of our operations on the tree take O(log n) time.




a. (2 point) Is this a valid binary search tree? Explain briefly why, or why not.
It is. The basic rule is that for each node, the left child can only contain items smaller than the node, and the right child items larger than the node. And indeed that property does hold for this tree.

b. (2 points) Is the tree balanced? Again, give a brief explanation.
No . In this tree the paths are both of length n/2, which is O(n), not O(log n).

c. (6 points) Write a method that returns `true` if the BST is balanced, and `false` otherwise. Hint: use some recursive helper methods. You won't need more than about 10 lines of code in total.

```
public  static Boolean isBalanced(Node n) {
       return maxDepth(n) <= Math.log2(countNodes(n));
}
public static int maxDepth(Node n)  {
      if(n == null) return 0;
      return Math.max(maxDepth(n.left), maxDepth(n.right))+1;
}
public static int countNodes(Node n) {
      if(n == null) return 0;
      return countNodes(n.left)+countNodes(n.right)+1;
}
```

7. (10 points) This question tests your knowledge of tree traversals.

a. (6 points, 3 each)

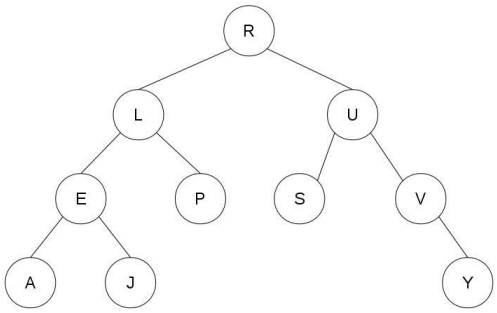
Write down the In-Order and Post-Order traversals of the binary tree in the given figure.

In-Order:

A E J L P R S U V Y

Post-order:

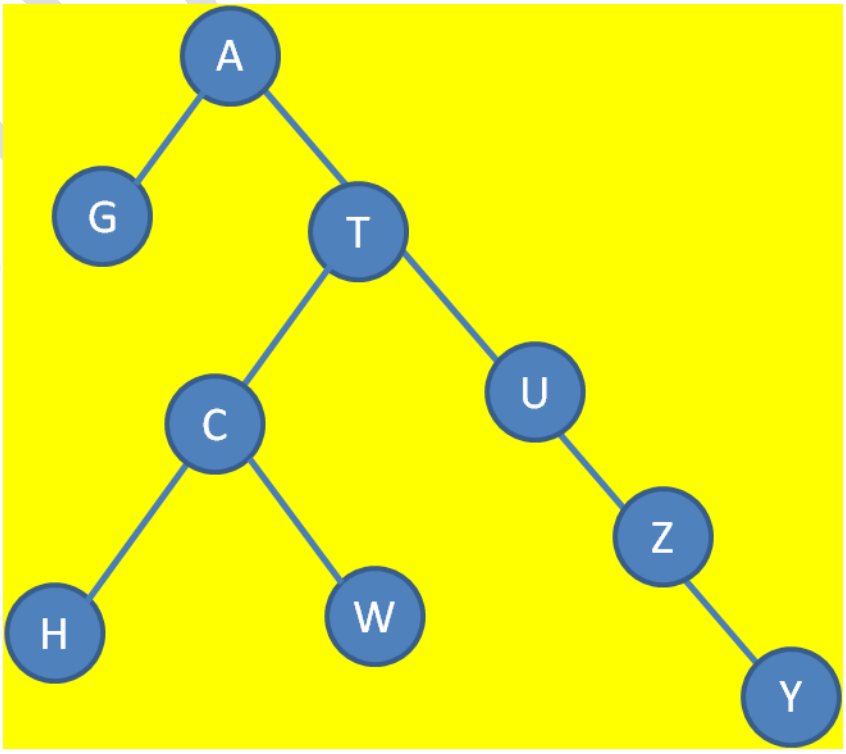
A J E P L S Y V U R




(b) (4 points) Draw a binary tree with its nodes correctly labeled such that it has the following two traversals:

Pre-Order: A G T C H W U Z Y

In-Order: G A H C W T U Z Y



8. (10 points) Write a method to implement simple letter substitution codes:

**`public static String Encode(String source, String from, String to)`**

that works as follows. The first argument is a message to encode. The second and third arguments are both strings of *k* characters: the first being a list of characters to encode and the second, their coded substitute. If a character isn't in the From list, it should be copied unchanged. So, for our example, if

**source="Dear Sally, I love you. Harry",**
**from="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz," and**
**to="wxyzABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuv",**

then the input string becomes **"zaWn OWhhu, E hkra ukq. DWnnu".**

```
public static String Encode(String source, String From, String To)
 {
       String result = "";
       for(int i = 0; i < source.length(); i++)
       {
             char c = source.charAt(i);    // Extract the i'th character
             boolean fnd = false;          // True if a mapping was found
             for(int j = 0; j < From.length(); j++)
                   if(From.charAt(j) == c){
                         // Found "c" in the From list, so map it
                         result = result + To.charAt(j);
                         fnd = true;
                         break;
                   }
             if(!fnd) // If the character wasn't mapped, just copy it
                   result = result+c;
       }
       return result;
 }
```

*Note: this is just one of many correct ways to write the Encode method. For example, there are predefined string methods can search for the first instance of one string in another string, and you can make your code shorter by using those. We'll award full credit for any correct solution that is coded in a clear way and has proper comments.*