CSCI 2170 Spring 2011
Review for test 4 (Monday, April 25th)

- **Stack**
  - Main characteristic
  - Basic stack operations
  - Array-based, pointer-based implementation
  - Client program using stack ADT

- **Queue**
  - Main characteristic
  - Basic Queue operations
  - Array-based implementation
  - Why do we use circular array, instead of regular array?
  - Client program using queue ADT
  - Client program that using queue and stack to solve problems

- **General Tree:** root, parent node, child node, ancestor, descendant, sub-tree, sibling, leaf

- **Binary tree**
  - Definition
  - level of a node
  - degree of a node
  - height of a tree
  - full binary tree
  - complete binary tree
  - number of nodes=2^H-1, height = ceiling($\log_2 N$)
  - balanced binary tree
  - In-order, pre-order, post-order traversal
  - Binary tree operations(i.e., build a new binary tree)

- **Binary search tree**
  - Definition
  - Insertion
  - Deletion
  - saving BST to file and restore BST from file
  - restore to the original tree
  - restore to the minimum height tree
  - binary search tree operations (insert, delete, pre-order / in-order / post-order traversal, copy tree, destroy tree, …), understand the code

- **AVL tree**
  - definition
  - build/maintain AVL tree using rotation (single rotation and double rotations) - depending on class schedule

Sample Test questions:

1. homework and closed lab questions
2. What is the characteristic of a stack? A queue?
3. Show the copy constructor of a pointer based implementation of the ADT Stack
4. Show the implementation of the Stack ADT member function "Pop" with array implementation.

5. Show the value of **front**, **back, count,** and the content of the circular array implementation of a queue, after the following statements are executed (assume MAXQUEUE_SIZE = 8)

```
int i, j;
Queue   Q;

for (i=0; i<5; i++)
     Q.EnQueue(i, success);
Q.DeQueue(success);
Q.DeQueue(success);
Q.DeQueue(success);
Q.DeQueue(j, success);
cout << j;
```

6. draw a complete binary tree with 15 nodes
7. what is an AVL tree?
8. why is it better to store large collection of records in a tree structure rather than a linked list?
9. what type of binary tree structure makes record insertion, deletion and retrieval most efficient?
10. understand the code that can be used to save a binary search tree and restore the tree, or rebuild the tree with minimum height.
11. Show how to build a binary search tree with records that have keys listed below: 40, 25, 8, 60, 48, 90, 31, 5, 17, 16, 29, 45, 46
12. what is the height of this tree?
13. Is this a balanced binary tree?
14. what is the level of the node with key 5?
15. Show the order of the nodes being visited (list the key values of the nodes visited) if pre-order traversal, in-order traversal, or post-order traversal method is used.
16. show the tree after the record with key 31 is deleted
17. show the tree after the record with key 40 is deleted