CS 124 Test 4 – Take Home Spring 2008

Name: ____ Due: **In class on 4/29/08**

1. (12 points) Draw the Binary Tree that results from inserting the following items in the order shown.

25, 57, 19, 65, 67, 90, 74, 3, 38, 53, 5, 91

2. (12 points) For the Binary Tree from question #1:

What is the height after all of the insertions are made? ______________________

What is the capacity of a tree of that height? ______________________

When searching this tree for a particular item what is the maximum number of comparisons that might be required? ____________________________

Would the insertion of the data item 92 into the tree change the height of the tree? _____________

3. (12 points) For the binary tree from question #1: give the pre-order and post-order traversals.

Pre-order:

Post-order

4. (20 points)What is output by the following program where everything in class Tree and class Node is as usual EXCEPT for fun1 and doFun1 which are as shown below? Fully explain what is going on.

```
bool Tree::fun1( ) { if (root)return doFun1(root); else return false; }

bool Tree::doFun1(Node *ptr ) {
if(ptr->left)  doFun1(ptr->left);
if(ptr->right) doFun1(ptr->right);
if(ptr->left)  doFun1(ptr->left);
ptr->data *=7;
if(ptr->data > 500) cout << .5*ptr->data<< endl;
else if(ptr->data < 50) cout << 5*ptr->data<< endl;

return true;}

int main( ) {
      Tree t;
      t.insert(14);t.insert(13);t.insert(31);t.insert(27);t.insert(12);
      t.fun1();
return 0;}
```

5. (20 points)What is output by the following program where everything in class Tree and class Node is as usual EXCEPT the overloaded output operators as shown below? Fully explain what is going on.

```
ostream& operator<<(ostream& os, const Node& n) {
      if (n.left) os << *(n.left);
      if (n.right) os << *(n.right);
      os << n.data << " ";
      if (n.left) os << *(n.left);
      if (n.right) os << *(n.right);
      return os;}

ostream& operator<<(ostream& os, const Tree& t) {
      if (t.root) os << *(t.root);
      return os;}

int main( ) {
Tree t;
t.insert(6);t.insert(10);t.insert(13);t.insert(7);t.insert(14);
cout << t << endl;
return 0;}
```

6. (24 points) For class Tree we see below a method called **LessThanNinety** that returns the number of nodes in the tree whose data is less than 90. It relies on another method to do most of the work. Complete the helper function **doLessThanNinety** that is started below which helps **LessThanNinety** do its task correctly.

```
int Tree:: LessThanNinety(void){

int count=0;

if(!root) return count;

      else return doLessThanNinety(root, count);}


int Tree:: doLessThanNinety(Node* ptr, int& count){


if(ptr->left) {







}

if(ptr->data < 90) {







}

if(ptr->right)  {







}

return            }
```