```cpp
// McBuster.cpp
#include "McBuster.h"
#include <iomanip>
#include <vector>
using namespace std;

bool McBuster::checkInsertion(ostream& Log) {
    Log << "Checking insertion of root to empty tree..." << endl;
    BSTreeT<int> T1;
    int rootValue = rand() % 1000;
    T1.Insert( rootValue ); // tree with root only
    BinNodeT<int>* rootAddress = T1.Root;
    if ( rootAddress == NULL ) {
        Log << " Failed to build tree with one node." << endl;
        Log << " Aborting insertion test." << endl;
        return false;
    } else if ( T1.Root->Element != rootValue ) {
        Log << " Failure: root stores wrong value." << endl;
    } else {
        Log << " Insertion of root was successful." << endl;
    }

    Log << "Checking insertion of two children..." << endl;
    int leftValue = T1.Root->Element - 10;
    int rightValue = T1.Root->Element + 10;
    T1.Insert( leftValue ); // insert left child
    T1.Insert( rightValue ); // insert right child
    BinNodeT<int>* leftChildAddress = T1.Root->Left;
    BinNodeT<int>* rightChildAddress = T1.Root->Right;
    if ( rootAddress != T1.Root ) {
        Log << " Failure: insertion messed up root address." << endl;
        Log << " Aborting insertion test." << endl;
        return false;
    } else if ( leftChildAddress == NULL || rightChildAddress == NULL ) {
        Log << " Failure: child pointers are messed up." << endl;
        Log << " Aborting insertion test." << endl;
        return false;
    } else if ( leftChildAddress->Element != leftValue ) {
        Log << " Failure: left child stores wrong value." << endl;
    } else if ( rightChildAddress->Element != rightValue ) {
        Log << " Failure: right child stores wrong value." << endl;
    } else {
        Log << " Insertion of child nodes was successful." << endl;
    }
    Log << endl;

    Log << "Beginning general insertion test..." << endl;
    BSTreeT<int> stuTree;
    int stuSize = fillBST(stuTree, Log);
    const unsigned int numToDisplay = 10;
    vector<TraceElemT<int> > Trace;
    traceBST(stuTree, Trace);

    unsigned int Size = Trace.size();
    unsigned int stepSize = Size / numToDisplay;
    if ( stepSize < 1 ) stepSize = 1;
    Log << "Checking locations of some random elements..." << endl;

    Log << "
```
```cpp
Log << "Level: Value" << endl;
int Count = 0;
for (unsigned int Pos = 0; Pos < Size; Pos = Pos + stepSize) {
    Count++;
    Log << setw(5) << Trace[Pos].Level << ": " << setw(6) << Trace[Pos].Data << endl;
}
return true;
}

bool McBuster::checkPass(BSTreeT<int> Copy, BSTreeT<int>& Original, ostream& Log) {
    // check for different root addresses
    if (Copy.Root == Original.Root) {
        Log << "Root of copy has same address root as of original." << endl;
        Log << "Aborting test NOW..." << endl;
        exit(1);
        return false;
    }
    else {
        Log << "Root of copy has different address from root of original." << endl;
    }

    vector<TraceElemT<int>> traceCopy;
    vector<TraceElemT<int>> traceOriginal;
    traceBST(Copy, traceCopy); // build models of original
    traceBST(Original, traceOriginal); // and of original to compare

    if (!compareTraces(traceCopy, traceOriginal, Log)) {
        Log << "Differences found between copy and original!" << endl;
        return false;
    }
    else {
        Log << "Traces of copy and original match." << endl;
    }

    return true;
}

int McBuster::fillBST(BSTreeT<int>& T, ostream& Log, int Size) {
    if (Size == INT_MAX)
        Size = 50 + rand() % 51;
    Log << "Creating BST with " << Size << " random integer values" << endl;
    for (int Count = 1; Count <= Size; Count++) {
        T.Insert( rand() % 1000);
    }
    return Size;
}
```