ICOM 4035 – Data Structures
Exam III
May 1, 2002

Name: ________________________________

Student Number: _______________________

Section: _________________________________

Instructions:

1. Write your name on all pages of this exam.
2. You have two hours to complete this exam. Use your time wisely.
3. This exam is worth 100 points, but it contains six problems totaling 110 points. Do as many problems as you can.
4. Read each question carefully, and show all the work you used to generate your answer.
5. To receive partial credit, you must show all the work you used to generate your answer.

GOOD LUCK!

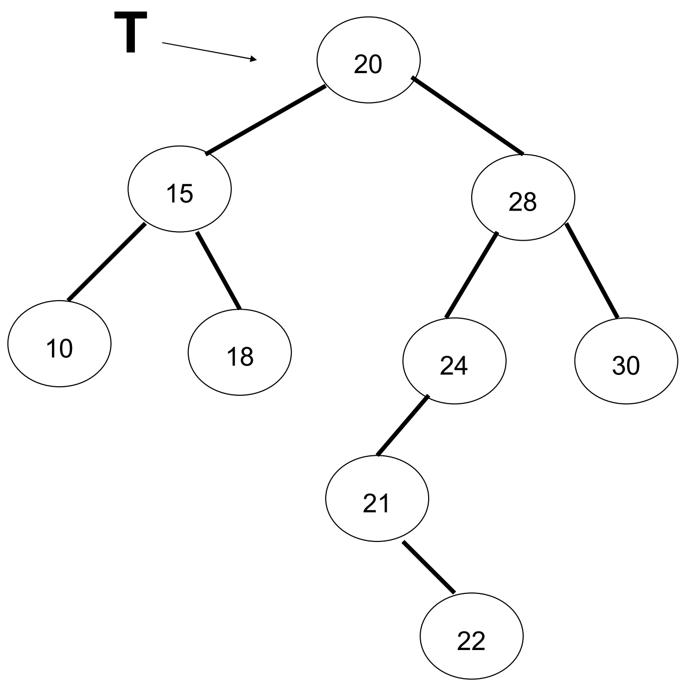
Name: _____________________________ Section: ___________

SCORE

1	/15
2	/10
3	/10
4	/15
5	/30
6	/30
TOTAL	/100

Name: ____________________________ Section: ___________

Problem 1. (15 points) Understanding of the Binary Search Tree
Use the following Binary Search Tree T, storing integers, to answer the following questions:



a) (5 pts) What is the height of the tree T?

Name: ____________________________ Section: ___________

Problem 1 (Continuation)

b) (5 pts) Draw the resulting BST after applying the operation T.delete(20) to the tree T.

c) (5 pts) Draw the resulting BST after applying the operation T.insert(21) to the original tree T.
(**NOTE: Assume the operation in b) was not executed!**)

Name: ____________________________ Section: ___________

Problem 2. (10 points) True or false about general course concepts

Use the Binary Search Tree T from problem 1 to determine whether each of the following statements is true or false. For those that you declare as false, you must explain your answer.

a) (5 pts) In a post-order tree traversal of tree T, the nodes will be visited in the following order: 10, 18, 15, 22, 21, 24, 30, 28, 20.

b) (5 pts) In a pre-order tree traversal of tree T, the nodes will be visited in the following order: 20, 15, 28, 10, 18, 24, 30, 21, 22

Name: ___________________________ Section: ___________

Problem 3. (10 pts) Big-Oh notation

Use Big-Oh notation to determine the complexity of the running time for each of the following code fragments. Briefly explain you answer.

a) (5 pts)

```
// Assume BST definition as in project 4
template <typename BSTData, typename Key>
int BinarySarchTree<BSTData,Key>::num_nodes
(BSTNode<BSTData> *node) const {
      if (node == NULL){
            return 0;
      }
      else {
            return 1 + num_nodes(node->left_child) +
                  num_nodes(node->right_child);

      }
}
```

b) (5 pts)

```
// Assume BST definition as in project 4
void print_data(const BinarySearchTree<int>& T) {
      int data[100], i=0, len = 100;
      for (i=0; i<len; ++i){
            data[i] = i;
      }
      for (i=0; i < len; ++i){
            if (T.erase(i)){
                  cout << "Number: "<< i << " was found" << endl;
            }
            else {
                  cout << "Number: "<< i << " was not found<< endl;
            }
      }
}
```

Name: ____________________________ Section: ___________

Problem 4. (15 points) Understanding of Binary Search Tree Container Class.
Trace the execution of the following operations on an instance of a Binary Search Tree container class (as in project 4) of `string`. Write your answer on the next page of this exam.

```
BinarySearchTree<string,string>;

T.insert(-1);
T.insert(10);
T.insert(-2);
T.insert(0);
T.erase(-1);
T.insert(3);
Pre_order_Iterator<string,string> = T.find(0);
cout << T.size() << endl;
T.insert(0);
T.erase(0);
T.insert(-4);
T.erase(10);
```

Name: _____________________________ Section: ___________

Problem 4. (Continuation)

Name: ____________________________ Section: ___________

Problem 5. (30 points) Usage of the Binary Search Container class.

A collection of one or more independent trees is called a **forest**. One mechanism used to represented a forest is simply as an array of trees and an integer that provides the number of trees in the forest. Suppose that you have forest of Binary Search Trees (BST) storing C++ `strings`, and that each tree has the same interface as in project 4. Answer the following questions:

a) **(10 pts)** Write a function `count_copies()` that returns the total number of times that a given string `str` appears in the forest.
Hint: Think about the in-order iterator for BST.

```
// returns number of copies of string str stored in the forest
int count_copies(BinarySearchTree<string,string> forest[],
                 int forest_size, const string& obj){
```

Name: ____________________________ Section: ___________

Problem 5. (Continuation)

b) **(10 pts)** Write a function `delete_from_forest()` that removes all copies of a element string `str` from the forest. **After completion, all copies of str are removed from all trees in the forest.**

```
void delete_from_forest
(BinarySearchTree<string,string>[] forest, int forest_size,
 const string& str){
```

Name: ____________________________ Section: ___________

Problem 5. (Continuation)

c) **(10pts)** Write a function `count_different()` that counts the number of different strings that are stored on a given tree *S,* where *S* provides the index in the array for the target tree.
Hint: Use a queue to store all the elements of tree *S*, and the find the different ones.

```
int count_different(BinarySearchTree<string,string> forest[],
                    int forest_size, int S){
```

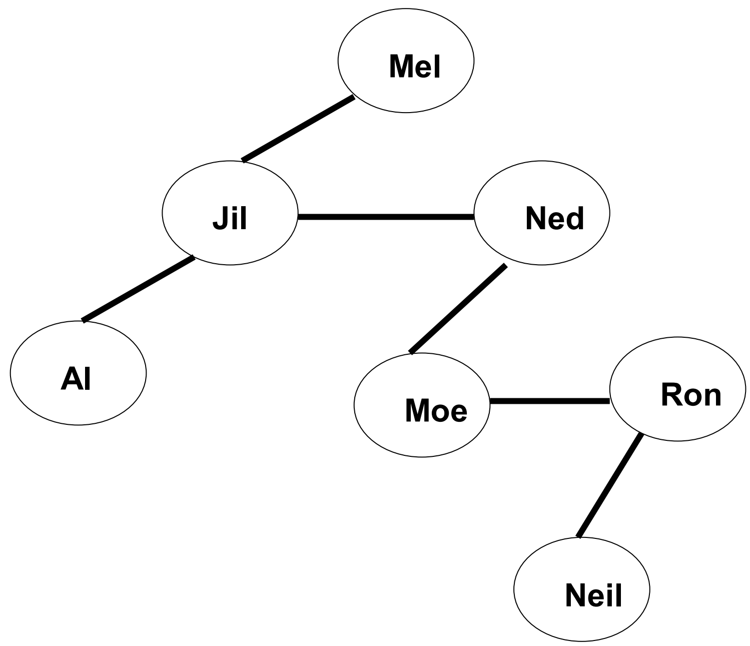
Name: ___________________________ Section: ___________

Problem 6. (30 points)

In this course, we have implemented Binary Search Tree using nodes that have pointers to the left child and the right child of the given node. An alternative implementation is by having each node have two pointers:

1. A pointer to its left child
2. A pointer to its right sibling

The following diagram illustrates this scheme:



We can declare the structure for the BSTNode from project 4 as follows:

```
// BST Node
template <typename BSTData>
    struct BSTNode {
      // Data stored in the BST
      BSTData data;
      // Left child
      BSTNode *left_child;
      // Right sibling
      BSTNode *right_sibling;
    };
```

With this information answer the following questions.

Name: ____________________________ Section: ___________

Problem 6 (Continuation)

a) (**10 pts**) Write the function `find_aux()` with finds a pointer to the first element in the binary search tree with a given key `K`.

```
template <typename BSTData, typename DataKey>
BSTNode<BSTData>* BinarySearchTree<BSTData,DataKey>::find_aux
(const DataKey& key, BSTNode<BSTData> *node) const {
```

Name: ____________________________ Section: ___________

Problem 6. (Continuation)

b) (10 pts) Write the function `insert_aux()` which inserts a new element in the Binary Search Tree. This function must follow the Binary Search Tree order property:

For any node N in the Binary Search Tree T, the key for node N is greater than the key of all the nodes in its left subtree, and is also smaller or equal than the key of all nodes on its right subtree

```
template <typename BSTData, typename DataKey>
void BinarySearchTree<BSTData,DataKey>::insert_aux
(const BSTData& obj, BSTNode<BSTData> * & node){
```

Name: ___________________________ Section: ___________

Problem 6. (Continuation)

c) (10 pts) Implement a new function `print_pre_preorder()` which visits all the nodes on a tree in pre-order, **printing the key** of each node as it visits them. Each key is printed on a different line.

```
// out - the output stream
// node - the root of the tree currently being visited in
// pre-order
template <typename BSTData, typename DataKey>
void BinarySearchTree<BSTData,DataKey>::print_pre_order
(ostream& out, BSTNode<BSTData> *node) {
```