The University of Chicago
Department of Computer Science

CMSC 15200 – Introduction to Computer Science 2
Summer Quarter 2005
Homework #6 (08/12/2005)
Due: 08/17/2005

Name:

Student ID:

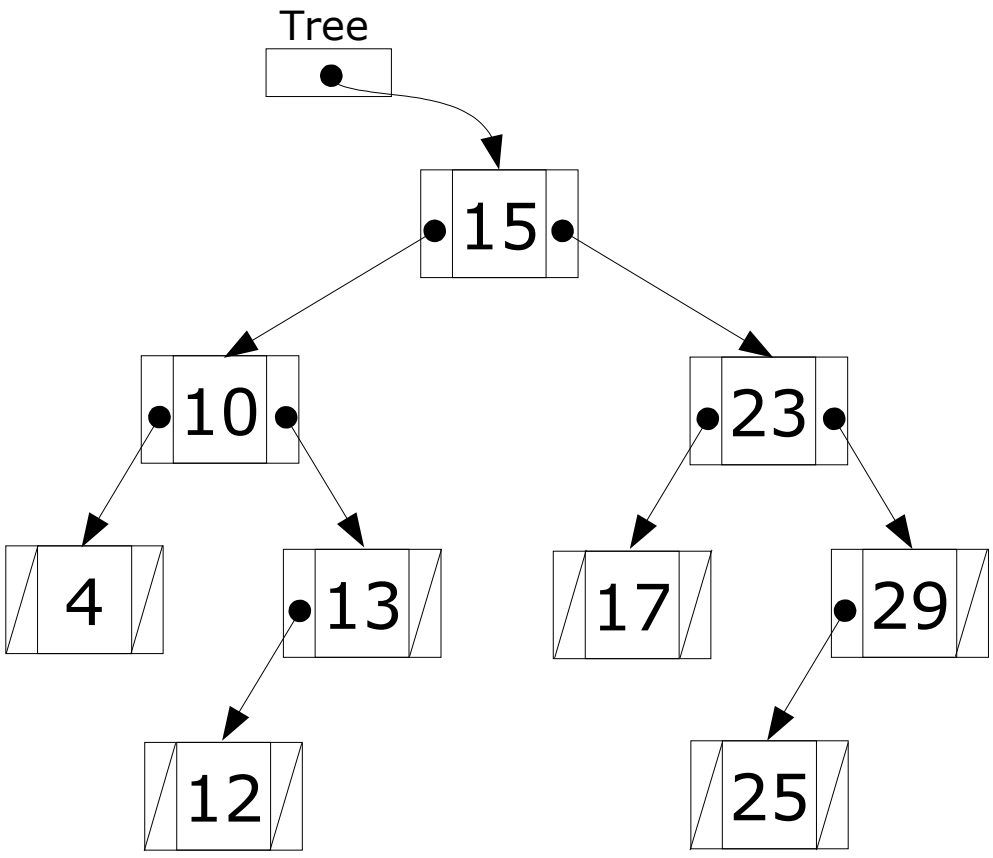
Instructor: Borja Sotomayor

Do not write in this area

1	2	3	4	5	6	**TOTAL**

Maximum possible points: 75 + 20

In this homework assignment you will implement a binary search tree (as described in class).



The structure and function declarations are the following (tree.h in the homework files):

```
struct TreeNode
{
	TreeNode* left;
	int value;
	TreeNode* right;
};

typedef TreeNode* Tree;

void createTree(Tree &t);

bool insert(Tree &t, int value);
void inorder(Tree &t);
void preorder(Tree &t);
void postorder(Tree &t);
bool find(Tree &t, int value);
bool remove(Tree &t, int value);
void breadth(Tree &t);
```

To test your list implementation, a main.cpp is provided in the homework files. Running this program with a correct BST implementation should yield the following:

```
Could not insert value 15. Already in tree.
Could not insert value 17. Already in tree.
Could not insert value 23. Already in tree.
4 10 12 13 15 17 23 25 29
15 10 4 13 12 23 17 29 25
4 12 13 10 17 25 29 23 15
Value 10 is contained in the tree.
Value 29 is contained in the tree.
Value 48 is NOT contained in the tree.
4 10 12 13 15 17 23 25 29
4 10 12 13 15 23 25 29
4 10 12 15 23 25 29
4 10 12 23 25 29
15 10 23 4 13 17 29 12 25
```

When debugging your program, take into account that the tree used in the main.cpp program is exactly the same as the one shown in the previous page.

Exercise 1 <<5 points>>

Implement the `createTree` function:

```
void createTree(Tree &t);
```

Exercise 2 <<15 points>>

Implement the `insert` function:

```
bool insert(Tree &t, int value);
```

This function inserts a new value in the tree, using the naïve algorithm seen in class (insert the value as a new leaf node). The function returns `true` if the insertion was successful and `false` if the tree already contains the specified value.

Exercise 3 <<10 points>>

Implement the following functions:

```
void inorder(Tree &t);
void preorder(Tree &t);
void postorder(Tree &t);
```

Each function does a different traversal of the tree (inorder, preorder, and postorder). During the traversal, the function will simply write out the value of each node.

Exercise 4 <<15 points>>

Implement the `find` function:

```
bool find(Tree &t, int value);
```

This function determines if the specified value is contained in the tree. You must implement this function recursively.

Exercise 5 <<30 points>>

Implement the `remove` function:

```
bool remove(Tree &t, int value);
```

This function removes the node with the specified value using the algorithm seen in class (i.e. you do not have to worry about rebalancing the tree). Remember that you will have to consider three cases: the node to remove has no children, one child, or two children. The function will return `true` if the removal was successful and `false` if no such value exists in the tree.

Exercise 6 <<20 points>> (Extra credit)

Implement the `breadth` function:

```
void breadth(Tree &t);
```

This functions does a breadth-first traversal of the tree. This algorithm has not been explained in class, so you will have to look it up elsewhere. Note that most sources show the algorithm for traversing cyclic graphs. The algorithm for breadth-first traversal of a tree is slightly simpler. Hint: You will not need to make any modifications to the TreeNode struct. Hint #2: You will need an additional data structure to perform the traversal.