**UNIVERSITY OF MASSACHUSETTS LOWELL**
**Department of Electrical and Computer Engineering**

**Program 8** **16.322 Data Structures** **Fall 2011**

# Binary Search Trees and Class Templates
# Word Counter Application

The object of this assignment is to write a program that will read in a text file, and count the number of occurrences of each word in the file. The list of words will be stored in an alphabetically ordered binary search tree, which can then be displayed in any of four different orders:

1. alphabetical order (inorder traversal)
2. preorder
3. postorder
4. level-by-level (breadth first search)

The program will also allow adding new words to the tree from the keyboard, as well as deleting words from the tree.

The starter files for this assignment are available on the class web page. Your job is to provide two modules:

1. a module called "BST_T.h," which gives a definition of a class *template* for a generic binary search tree, and
2. a module called "Queue_T.h" which gives the definition of a class *template* for a queue. You should construct this module by converting your Queue class from Program 6 into a class template. The queue will be used in the level-by-level display.

Note that you need not provide destructor functions, copy constructors, or overloaded assignment operators.

The application is menu driven, supporting the following commands:

| **Command** | **Description** |
|---|---|
| A | Show the tree on the screen in alphabetical order. |
| B | Show the tree on the screen in preorder. |
| D *word* | Delete *word* from the tree, maintaining correct alphabetical order. |
| I *word* | Insert *word* into the tree in alphabetical order. |
| L | Show the tree on the screen in level-by-level order. |
| O *filename* | Open and read the text file from *filename*, storing words and their number of occurrences in the tree. |
| P | Show the tree on the screen in postorder. |
| Q | Terminate the application. |
| X *command filename* | Execute the sequence of commands in the file "*command filename*". |

FILES

The following source files are required.

| FILES | | CONTENTS |
|---|---|---|
| Prog8.cpp | Provided – do not modify. | The main program file. |
| BST_T.h | You supply this module. | Defines the class template for class "BST". |
| Queue_T.h | You supply this module. | Defines the class template for class "Queue". |
| WordCounter.h | Provided, do not change. | Defines a class "WordCount", whose objects store a word string along with an unsigned integer that counters the number of occurrences of that word. |
| Utility.h<br>Utility.cpp | Provided, do not change | The Utility module provides functions for controlling the cursor position in the console window, and for delaying execution of the program. |

**SAMPLE RUN**

```
>O WORDFILE.TXT
>A
all=6
brown=3
come=10
doggie=9
for=5
fox=4
good=7
is=2
jumped=5
lazy=8
men=8
now=1
over=6
party=13
quick=2
the=23
time=4
to=20
>B
now=1
is=2
for=5
all=6
come=10
brown=3
doggie=9
good=7
fox=4
men=8
jumped=5
lazy=8
the=23
party=13
over=6
quick=2
time=4
to=20
>P
brown=3
doggie=9
come=10
all=6
fox=4
good=7
for=5
lazy=8
jumped=5
men=8
is=2
over=6
quick=2
party=13
to=20
time=4
the=23
now=1
```

```
>L
now=1
is=2
the=23
for=5
men=8
party=13
time=4
all=6
good=7
jumped=5
over=6
quick=2
to=20
come=10
fox=4
lazy=8
brown=3
doggie=9
>D NOW
>A
all=6
brown=3
come=10
doggie=9
for=5
fox=4
good=7
is=2
jumped=5
lazy=8
men=8
over=6
party=13
quick=2
the=23
time=4
to=20
>I THEN
>A
all=6
brown=3
come=10
doggie=9
for=5
fox=4
good=7
is=2
jumped=5
lazy=8
men=8
over=6
party=13
quick=2
the=23
then=1
time=4
to=20
>Q
Press any key to continue . . .
```

**REQUIRED CLASS TEMPLATE DEFINITION – BST_T.h (Incomplete)**

```
template <typename NodeData>
class BST
{
private:
   // Tree node class definition
   struct Node
   {
     // Constructors
        Node() : left(0), right(0) {}
        Node(const NodeData &d) : data(d), left(0), right(0) { }
     // Data Members
     NodeData    data;    // The "contents" of the node
     Node        *left;   // Link to the left successor node
     Node        *right;  // Link to the right successor node
   };
public:
   // Constructor
   BST() : root(0), current(0) { }
   // True if the tree is empty
   bool Empty() { return root == 0;}
   // Search for an entry in the tree. If the entry is found,
   // make it the "current" entry. If not, make the current entry
   // NULL. Return true if the entry is found; otherwise return false.
   bool Search(NodeData &d);
   // Add a new node to the tree.
   void Insert(NodeData &d);
   // Delete the current node.
   void Delete();
   // Output the tree to the "os" in the indicated sequence.
   void OutputInOrder(ostream &os) const;    // Output inorder
   void OutputPreOrder(ostream &os) const;   // Output preorder
   void OutputPostOrder(ostream &os) const;  // Output postorder
   void OutputByLevel(ostream &os); const;   // Output by level
   // Retrieve the data part of the current node.
   NodeData Current() { return current->data; }
   // Show the binary tree on the screen.
   void ShowTree() const;
private:
   Node *root;      // Points to the root node
   Node *current;   // Points to the current node
   Node *parent;    // Points to current node's parent

   // Recursive Search
   bool RSearch(Node *subTree, NodeData &d);
   // Recursive Insert
   void RInsert(Node *&subTree, NodeData &d);

   // Recursive Traversal Functions
   void ROutputInOrder(Node *subTree, ostream &os) const;
   void ROutputPreOrder(Node *subTree, ostream &os) const;
   void ROutputPostOrder(Node *subTree, ostream &os) const;

   // Find the parent of leftmost right successor of the current node.
   Node *ParentOfLeftMostRightSucc(Node *node, Node *parent) const;
   // Show the binary tree on the screen.
   void RShowTree(Node *subTree, int x, int y) const;
};

template <typename NodeData>
bool BST<NodeData>::Search(NodeData &d)
{
   parent = 0;
   return RSearch(root, d);
}
       …MISSING MEMBER FUNCTION DEFINITIONS GO HERE…
```

## GENERAL REQUIREMENTS

1. Your binary search tree must be implemented as class template in the file “BST_T.h”, and must be completely generic, having no knowledge of the structure of the data part of a node.
2. Your queue module must be implemented as class template in the file “Queue_T.h”, and must be completely generic, having no knowledge of the structure of the data part of a node.
3. The operations search, and insert node, as well as the inorder, postorder, and preorder traversals must be implemented using recursion – no loops are allowed. This will necessitate the use of auxiliary functions so that the root pointer may be passed from level to level.
4. You may not define any additional data members.
5. Use the provided Prog8.cpp starter file without change.
6. Your program's output must appear exactly as shown in the sample run, producing exactly the same output format.
7. Use const definitions to define symbolic constants rather than hard-coding constants into your program. For example specify the array dimensions (size) with a symbolic constant. Then, it is easy to change the sizes of all of the arrays. Do not use #define to name constants.
8. Use descriptive names throughout your program. Strive to make your code so readable that is self-explanatory.
9. Use a consistent indentation scheme to help show the structure of your program.
10. Use a consistent capitalization convention to distinguish variables from constants. The preferred convention is that constant names, type names, class names, and function names have the first letter of each word capitalized (e.g., MaxCurvePoints). Variable names, data members, and parameter names, have the first letter of every word except the first capitalized (e.g., thePoint, numPoints). Using this convention, names representing values that may change (L-values) start with lower case. If a name starts with a capital, its value cannot be changed.
11. Include a sufficient number of comments to explain your program.
12. Do not use goto statements.
13. Do not use global variables.
14. No function definition may be longer than one page (60 lines).
15. Do not use redundant code. If you need the same procedure in more than one place, make it a function.
16. Keep your functions simple. If your program seems overly complex, it probably is. This means that you did not spend enough time designing, before you started typing code. KEEP IT SIMPLE!
17. Eliminate all warning messages.
18. Do not use C-style input-output functions (stdio) such as printf(), scanf(), getchar(), putchar(), etc.

## SUBMISSION REQUIREMENTS

Hand in the following items:

1. A computer printout of your include files, “BST_T.h.” and “Queue_T.h”.

2. A computer printout of test run(s) of your using showing at least two examples of your program’s operation. Make sure that you include the sample run shown above.

3. Zip up the source files `BST_T.h and Queue_T.h`. Name the zip file *LASTNAME*.zip, and submit it via blackboard. The files submitted must be identical to the printout from item 1. If you make changes, then take back your submission and submit the changed version. **Make sure that your zip file includes only the requested files and no folders.**

   **NOTE: Programs received after 12:00 noon on the due date will be considered late.**