CS 302 – Assignment #05

Purpose: Learn concepts regarding balanced binary trees.
Due: Tuesday (10/10) → Must be submitted on-line before class.
Points: Part A → 150 pts, Part B → 50 pts

Assignment:

Part A:

Many web sites collect product reviews. Design and implement a C++ template class ***reviewData*** to implement a product review program using an AVL Tree[1] data structure. For this, we will implement two classes as follows:

- Design and implement a C++ template class, ***avlTree.h***, to implement an avlTree data structure. A main will be provided that performs a series of tests using the avlTree class with different data types.
- Design and implement a C++ class, ***reviewData***, that will inherit from the ***avlTree*** class to store the product reviews.

A main will be provided that performs the user interface and uses the ***reviewData*** object.

Source: http://xkcd.com/1497/

Part B:

When completed, create and submit a write-up (PDF format) not too exceed ~500 words including the following:

- Name, Assignment, Section
- Summary of the AVL tree data structure.
- Compare using an AVL tree to a binary search tree.
  - Include advantages and disadvantages of each implementation approach.
- Big-O for the various AVL tree operations.

Visualization

The following web site provides a visualization for AVL Trees, including the rotate operations.
https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

It should be noted that there are many implementation variations on these data structures and algorithms. These are the algorithms that must be implemented. Copying code from the net will result in a zero for the assignment and referral to the Office of Student Conduct.

Submission:

- Part A → Submit a compressed zip file of the program source files, header files, and makefile via the on-line submission by 23:50.
- Part B → A copy of the write-up including the chart. Must use PDF format.

All necessary files must be included in the ZIP file. The grader will download, uncompress, and type **make** (so you must have a valid, working *makefile*).

1 For more information, refer to: http://en.wikipedia.org/wiki/AVL_tree

Class Descriptions

- AVL Tree Class
  The AVL tree template stack class will implement functions specified below.
  We will use the following node structure definition.

```
template <class myType>
struct nodeType {
      myType             keyValue;
      double             score;
      unsigned int       count;
      nodeType<myType>   *left;
      nodeType<myType>   *right;
};
```

avlTree<myType>
-nodeType<myType> *root
+avlTree()
+~avlTree()
+destroyTree(): void
+countNodes() const: int
+height() const: int
+search(myType, double &, unsigned int &) const: bool
+printTree() const: void
+insert(myType, double): void
+findMaxReview(string &, double &, unsigned int &): bool
-destroyTree(nodeType<myType> *): void
-countNodes(nodeType<myType> *) const: int
-height(nodeType<myType> *) const: int
-search(myType, nodeType<myType> *) const: nodeType<myType> *
-findMaxScore(nodeType<myType> *, string &, double &, unsigned int &): void
-printTree(nodeType<myType> *) const: void
-insert(myType, double, nodeType<myType> *): nodeType<myType> *
-rightRotate(nodeType<myType> *): nodeType<myType> *
-leftRotate(nodeType<myType> *): nodeType<myType> *
-getBalance(nodeType<myType> *) const: int

Function Descriptions

- The *avlTree()* constructor should initialize the tree to an empty state.
- The *~avlTree()* destructor should delete the tree by calling the private *destroyTree()* function.
- The public *destroyTree()* function should delete the tree by calling the private *destroyTree()* function.
- The private *destroyTree()* function should recursively delete the tree (including releasing all the allocated memory). Must be recursive.
- The public *countNodes()* function should return the total count of nodes in the tree by calling the private *countNodes()* function.

- The private *countNodes()* function should recursively return the total count of nodes in the tree. Must be recursive.
- The public *height()* function should return the maximum height of the tree by calling the private *height()* function.
- The private *height()* function should recursively return maximum height of the tree. Must be recursive.
- The public *search()* function should call the private *search()* function to determine if the passed product is in the tree. If the product is found, the function should return true and the score and count. If not found, the function should return false.
- The private *search()* function should recursively search the tree for the passed product and, if found, return a pointer to the node. If not found, the function should return NULL. Must be recursive.
- The public *findMaxReview()* function should call the private *findMaxReview()* function to search the tree and find the product with the most reviews. If there is no tree, the function should return false. Otherwise, the function should find the product with the highest number of reviews and return true along with the product, score, and count true (via reference).
- The private *findMaxReview()* function should recursively search the tree for product with the most reviews. The product, score, and count true should be returned (via reference). Must be recursive.
- The public *printTree()* function should call the private *printTree()* function to print the tree in the order passed.
- The private *printTree()* function should recursively print the tree in post-order format. The product ID, score (fixed, showpoint, setprecision(2)), and count should be displayed (two spaces between each). Must be recursive.
- The public *insert()* function should call the private *insert()* function to insert the passed product ID and score into the tree. If the product is already in the tree, the score and count should be updated. The new score should be added to the existing score. The average score will be calculated later. If the product is not already in the tree, a new node should be added.
- The private *insert()* function should recursively insert the passed product ID and score into the tree. The function will use the private *leftRotate()*, *rightRotate()*, and *getBalance()* functions.
- The private *getBalance()* function should return the balance factor (left subtree height – right subtree height) of the passed node. This is used during the insert.
- The private *rightRotate()* function should perform a right tree rotate operation (as described in class, in the lecture notes, and in the text). This is used during the insert.
- The private *leftRotate()* function should perform a left tree rotate operation (as described in class, in the lecture notes, and in the text). This is used during the insert.

- <u>Review Data Class</u>
  The review data class ***must*** inherit from the ***avlTree*** class and will implement functions specified below.

reviewData public avlTree<string>
-totalReviews: int
+reviewData()
+~reviewData()
+readMasterReviewData(const string): bool
+getReviews(const string): bool

+showStats() const: void
+showMaxReview() const: void
+printAllReviews() const: void
+printProduct(const string, const double, const unsigned int): void

Function Descriptions

- The *reviewData()* constructor should initialize the class variables.
- The *~reviewData()* destructor should reset the class variables.
- The *readMasterReviewData()* function should read the passed master zip codes file and *insert()* the words in the Red-Black tree. If the master zip codes file read operations are successful, the function should return true and false otherwise.
- The *getReviews()* function should read the passed product ID's data file. If the data file read open/read operations are successful, the function should return true and false otherwise. For each product ID in the file, the function should check if a review exists and if so, print it using the *printProduct*() function. If not, it should display a formatted message "Product, <productID> not found." message. Refer to the example output for formatting.
- The *showStats()* function should display the data statistics (in the format shown in the examples) including the tree height via the *height*() function, total reviews via the *totalReviews* class variable, and total unique products via the *countNodes*() function. Refer to the example output for formatting.
- The *showMaxReviews*() function should show the product with the maximum number of reviews via the inherited *findMaxReview*() function.
- The *printAllProducts*() function should print all products in the tree. The is primarily used for debugging. Refer to the example output for formatting.
- The *printProduct*() function should print the passed product information in a formatted manner. The average score should be calculated based on the score total and the review count. This function is used by multiple other functions. Refer to the example output for formatting.

Refer to the example executions for output formatting. Make sure your program includes the appropriate documentation. See Program Evaluation Criteria for CS 302 for additional information. ***Note, points will be deducted for especially poor style or inefficient coding.***

AVL Tree Algorithms

The following is a summary of a couple AVL tree algorithms.

AVL Tree Balance Function

- AVL Tree Get Balance Function
  - return height(left) - height(right)

AVL Tree Height Function

- AVL Tree Height → Private Function
  - if node is NULL
    - return 0
  - else
    - recursively get left height
    - recursively get right height
    - return max left height or right height + one

AVL Tree Insertion Function

- AVL Tree Insertion → Private Function
  - recursively perform normal BST insertion
    - if NULL
      insert new node
      return node
    - else
      based on key, go left or right
  - get balance factor
  - check for possible cases for unbalanced
    - if (balance factor > 1 AND key < left node value) // left left case
      return right rotate
    - if (balance factor < -1 AND key > right node value) // right right case
      return left rotate
    - if (balance factor > 1 AND key > left node value) // left right case
      left node = left rotate
      return right rotate
    - if (balance factor < -1 AND key < right node value) // right left case
      right node = right rotate
      - return left rotate
  - return node (possibly unchanged)

Example Execution:

Below is an example program execution for the tree test program.
Note, the **`ed-vm%`** is the prompt on my machine.

```
ed-vm%
ed-vm% ./avlTest
****************************************************************
CS 302 - AVL Tree Test Program

------------------------------
Test Set #0  (10)

Max Height: 4
Node Count: 10
Tree:
Complete Tree: (debug)
----------------------
a  1.00  1
answer  3.00  1
balloon  9.00  1
ball  8.00  1
any  4.00  1
bye  6.00  1
their  7.00  1
there  2.00  1
the  0.00  1
by  5.00  1


------------------------------
Test Set #1  (50)

Max Height: 6
Node Count: 50
Tree:
Complete Tree: (debug)
----------------------
aah  0.00  1
aahing  2.00  1
aahed  1.00  1
aal  4.00  1
aaliis  6.00  1
aalii  5.00  1
aahs  3.00  1
aardvark  8.00  1
aargh  10.00  1
aardwolf  9.00  1
aarrghh  12.00  1
aasvogel  14.00  1
aas  13.00  1
aarrgh  11.00  1
aals  7.00  1
aba  16.00  1
abacas  18.00  1
abaca  17.00  1
aback  20.00  1
abacuses  22.00  1
abacus  21.00  1
abaci  19.00  1
abaka  24.00  1
abalone  26.00  1
abakas  25.00  1
abamp  28.00  1
abamps  30.00  1
abampere  29.00  1
abalones  27.00  1
abaft  23.00  1
ab  15.00  1
abandons  32.00  1
abas  34.00  1
abapical  33.00  1
abased  36.00  1
abaser  38.00  1
abasedly  37.00  1
```

```
abase  35.00  1
abases  40.00  1
abashed  42.00  1
abash  41.00  1
abashing  44.00  1
abasias  46.00  1
abasia  45.00  1
abate  49.00  1
abatable  48.00  1
abasing  47.00  1
abashes  43.00  1
abasers  39.00  1
abandon  31.00  1


------------------------------
Test Set #2  (25)

Max Height: 5
Node Count: 18
Tree:
Complete Tree: (debug)
----------------------
a  20.00  2
ab  22.00  2
abaa  30.00  2
abab  32.00  2
aba  6.00  1
abad  15.00  1
abaf  17.00  1
abae  16.00  1
abc  8.00  1
abb  7.00  1
abac  14.00  1
abe  10.00  1
ac  24.00  2
abf  11.00  1
af  5.00  1
ae  28.00  2
ad  26.00  2
abd  9.00  1


------------------------------
Test Set #3  (30)

Max Height: 5
Node Count: 30
Tree:
Complete Tree: (debug)
----------------------
a  0.00  1
aaa  2.00  1
aa  1.00  1
aaaaa  4.00  1
aaaaaaa  6.00  1
aaaaaa  5.00  1
aaaa  3.00  1
aaaaaaaaa  8.00  1
aaaaaaaaaaa  10.00  1
aaaaaaaaaa  9.00  1
aaaaaaaaaaaaa  12.00  1
aaaaaaaaaaaaaaa  14.00  1
aaaaaaaaaaaaaa  13.00  1
aaaaaaaaaaaa  11.00  1
aaaaaaaa  7.00  1
aaaaaaaaaaaaaaaaa  16.00  1
aaaaaaaaaaaaaaaaaa  18.00  1
aaaaaaaaaaaaaaaaa  17.00  1
aaaaaaaaaaaaaaaaaaaa  20.00  1
aaaaaaaaaaaaaaaaaaaaa  22.00  1
aaaaaaaaaaaaaaaaaaaa  21.00  1
aaaaaaaaaaaaaaaaaa  19.00  1
aaaaaaaaaaaaaaaaaaaaaaa  24.00  1
aaaaaaaaaaaaaaaaaaaaaaaa  26.00  1
```

```
aaaaaaaaaaaaaaaaaaaaaaaa  25.00  1
aaaaaaaaaaaaaaaaaaaaaaaaaaaa  29.00  1
aaaaaaaaaaaaaaaaaaaaaaaaaaaa  28.00  1
aaaaaaaaaaaaaaaaaaaaaaaaaaaa  27.00  1
aaaaaaaaaaaaaaaaaaaaaaa  23.00  1
aaaaaaaaaaaaaaaa  15.00  1


------------------------------
Game Over, thank you for playing.
ed-vm%
ed-vm%
```

Below is an example program execution for the check zips program.

```
ed-vm%
ed-vm% ./reviews -m foodsSm.txt
****************************************************************
CS 302 - Assignment #5
Amazon Review Checking Program

================================================
Select Option:
   'p' - process input file.
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> s
------------------------------------
Review Data Statistics:

Review Data Tree Stats:
   Tree Height: 7

Review Data Stats:
   Total Reviews: 222
   Unique Products: 55

================================================
Select Option:
   'p' - process input file.
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> m

--------------------
Product: B00374ZKQ0
  Avg Score: 2.93
  Reviews:   28

================================================
Select Option:
   'p' - process input file.
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> l
Enter Product: B003O5Q3KE

Product: B003O5Q3KE
  Avg Score: 3.00
  Reviews:   4


================================================
Select Option:
   'p' - process input file.
```

```
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> p
Enter input file: on0.txt
reviews: Error reading  input data file.
================================================
Select Option:
   'p' - process input file.
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> p
Enter input file: in0.txt
*******************************************

Reviews List:
-------------
prod: B001LR2CU2
Product: B001LR2CU2
  Avg Score: 5.00
  Reviews:   1

prod: B004I613EE
Product: B004I613EE
  Avg Score: 5.00
  Reviews:   2

prod: B001EO7N10
Product: B001EO7N10
  Avg Score: 4.50
  Reviews:   6


================================================
Select Option:
   'p' - process input file.
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> a
Complete Tree: (debug)
----------------------
B0000D16IP  6.00  2
B0002ARMS6  5.00  1
B0001WYNFA  5.00  1
B000H7K114  5.00  1
B000KOSIP0  20.00  6
B000JT45IA  5.00  1
B000H28ABW  64.00  14
B000LKVRQA  42.00  10
B000NY8O9M  5.00  1
B000NY4SAG  19.00  4
B0013Z0PTW  92.00  22
B000RHXJM2  5.00  1
B000P56I7Y  5.00  1
B000KOUOHU  5.00  1
B0018CLWM4  32.00  7
B001682QCK  15.00  5
B001EO7N10  27.00  6
B001ELL54Y  24.00  6
B001FPT1WM  36.00  8
B001F2GDJY  10.00  2
B001EQ5O6Y  71.00  18
B001BOAOLY  27.00  6
B001LR2CU2  5.00  1
B002OXLXLG  5.00  1
B001TGY7W6  5.00  1
B002XO3Q52  3.00  1
```

```
B002PXSGA6  10.00  2
B001IZHZJA  32.00  8
B0015V7GG4  24.00  5
B0039KE8Y2  4.00  1
B003IFB148  5.00  1
B003EMXLU2  18.00  4
B003NQMPYM  5.00  1
B003JHR4GE  15.00  3
B003S1WTCU  2.00  1
B003Q9VWUO  2.00  1
B003XUL27E  7.00  5
B004BRECP2  3.00  1
B003VWU7IE  28.00  6
B003O5Q3KE  12.00  4
B004BY23I8  12.00  3
B004CHDG44  5.00  1
B004I613EE  10.00  2
B004MZ4ODW  6.00  2
B004JLGEII  21.00  5
B004CZUOSM  2.00  1
B005ZC0RRO  5.00  1
B005OTVL8C  5.00  1
B006T7TKZO  5.00  1
B009GTIHG0  5.00  1
B006Z0Z6WG  4.00  1
B006JSPXZY  1.00  1
B005OCX5XI  1.00  1
B004BRECPW  6.00  2
B00374ZKQ0  82.00  28

==================================================
Select Option:
   'p' - process input file.
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> q

************************************************************
Game Over, thank you for playing.
```