Lab 6: Binary Search Trees/AVL Trees

(and a little bit of linked lists)

(100 pts, due Nov 15 at midnight)

The game: WordHub

The idea: A player gets a set of letters of size x (you pick) and then generates as many words as possible with those letters only. The words are then checked for validity or not, and a player gets a final score that is the total number of words that used only the letters and occur in a dictionary versus the words that either use invalid letters or don't occur in the dictionary.

The crux of this: Reading in a dictionary file into either a Binary Search Tree or a BST modified using AVL (an AVL tree). For this, I used a flag to set whether, as words are being inserted into the binary search tree, the tree is also balanced and updated using AVL techniques or not.

Things I didn't worry about: caps versus small letters.

Code you'll be writing:

For this lab you will be creating the header file and the definition file for both an AVL tree and a simple linked list. With minor adjustments (and one more method) the linked list you wrote for the last lab should work fine, so the focus of this lab will be the AVL Tree header and definition files.

Note that an AVL tree is a binary search tree, with the caveat that it has been adjusted to be balanced as nodes are inserted. So make sure your AVL tree class has a flag that is true when you want to do the necessary check balances and rotations to create an AVL balanced binary search tree, and false when you just want a regular balanced binary search tree. You can set the flag in the main function, when creating a Game object.

Binary Search Tree/AVL Tree Portion:

You must write the header and the definitions for the binary search tree/AVL tree.

For the BST/AVL Tree, you should have at a minimum the following fields:

NodeT *root; // holds the root of the bst
bool AVLflag; // flag for whether to adjust bst to be an avl tree

And your methods should be

(Note: the below methods are necessary for both the Binary Search Tree and an AVL Binary Search Tree, with no difference in how they're written)

AVLTree(bool flag); //constructor
bool findWord (string s, NodeT *n); // finds whether s is in the bst. Returns true (if found) and false otherwise. Note: I called this with the root so that this method could be recursive. It doesn't have to be. If you prefer to write this iteratively, you won't need the NodeT pointer as a parameter.
void addNode(string s,NodeT *r); Adding s to the tree. Again, if done recursively, you'll need a NodeT address as a parameter. If not, you won't.
void printIO(NodeT *root);
void printPre(NodeT *root);
void printPost(NodeT *root);

Note: the below methods aren't necessary for a plain Binary Search Tree, but they adjust the heights of nodes, which makes debugging easier. In addition, if the flag is set, adjustHeights would call the methods that determine whether the balance is okay, and if not, then it determines and calls the appropriate rotations

void adjustHeights(NodeT *n); //starting with the node you just inserted, adjust the heights of its parents/grandparents/great… until a great… grandparent node's height doesn't change. If the AVLTree flag is set,

this method also checks balances and, if a node is unbalanced, calls the appropriate rotation(s) and re-adjusts heights and checks balances from that node up. To adjust heights at any moment, you get the max of the height of the left child and the height of the right child, and add 1. MAKE SURE you attach the newly rotated top node to the parent above it.
int getMax(NodeT *n); // This method is a little helper method that determines the max height between the left child and the right child and returns that height.

Note: the below methods are just for AVL Trees and are only called when the AVL flag is set to true.
NodeT *rotateRight(NodeT *n); // This method does an AVL right rotation, returning the new parent
NodeT * rotateLeft(NodeT *n); // This method does an AVL left rotation, returning the new parent
int getDiff(NodeT *n); // This method gets the difference between the left and the right child.

For the AVLTree class, you can write other methods if you want. Equally, the helper methods (getMax, getDiff) aren't necessary, but I liked having them.

Linked List Portion:

The linked list portion is for the words the user types in after s/he has received his/her letters. Each word is added to the linked list. These are the words that are checked with the dictionary (which is in the form of a binary search tree) in the Game oject's methods. Your job is to write the header file and the definitions for a Linked List. Because we're just adding words to an empty linked list, the methods required are a subset of the methods you wrote for lab 5. The only new field and method is the score field/ getScore method. The score is initially set to 0. The getScore method traverses the list and adds the score of each word/Node in the list, then sets the score to be that total.

For the Linked List, you should have at a minimum the following fields:

NodeL *first;
NodeL *last;
int size;
int score; // This field is for the game's score, and will be set using a getScore method, below. It should be initialized in the constructor to 0.

And your methods you'll need for this are:

LL();
void push(string s);
void addFirst(string s);
void printList();
void getScore(); // This is the new method – each word already has a score. This method just traverses the linked list from the first to the last node and keeps a running total of the wscore of each node. Then the score field is set to that total.

To Test:

1. (25 pts) Set the flag in main to false, and use the testdict.txt file. At first, comment out the call to the startGame method in the main function. Make sure your output looks like my output (included below) for the pre-order , in-order, and post-order printing of the tree.
2. (25 pts) Uncomment Out the startGame method in the main, and run the program. Check to see if you have output similar to my game output below (Note: to do this properly, test with the commondict.txt file, which has about 2000 of the most common English words. This isn't as good as the scrabble dictionary, also included, but on the other hand the scrabble dictionary will take forever to read in, so for testing purposes, you probably want to stick with the commondict.txt file.

    a. Make sure you try a word with characters not in your list. Make sure at least one word is in the dictionary, and one isn't. Check to make sure the words are scored correctly and the final score is correct.
3. (35 pts) In Main, go back to using testdict.txt and change the flag to true. This should turn on the AVL Tree flag so that the tree created uses the AVL techniques to balance the tree. Recomment out the startGame method. Check to see that your ouput for pre, in, and postorder printouts look like mine, below.
4. (15 pts) Uncomment Out the startGame method in the main, and run the program. Make sure everything still works.
    a. Make sure you try a word with characters not in your list. Make sure at least one word is in the

Extra Credit:

(5 pts) iInstead of signaling that the user is done entering possible words by entering a -1, use a timer so that the user is limited to entering words only with an allotted amount of time.

(10 pts) Instead of just scoring words with 1 or -1, give each letter a value, so that words using more difficult letters will get higher scores than those using less difficult letters (this is scrabblish – you'd probably want to rank the actual words themselves at a difficulty level and then give valid words scores based on

---

Output

(Pt 1) Testdict with AVLflag turned off to false (so just a regular binary search tree):

```
Adding: babe
Made root
Adding: rest
Inserting to right of babe

Adding: wanton
Looking right of babe
Inserting to right of rest

Adding: shawn
Looking right of babe
Looking right of rest
Inserting to left of wanton

Adding: dunce
Looking right of babe
Inserting to left of rest

Adding: fabric
Looking right of babe
Looking left of rest
Inserting to right of dunce

Adding: deject
Looking right of babe
Looking left of rest
Inserting to left of dunce

Adding: chars
Looking right of babe
Looking left of rest
Looking left of dunce
```

```
Inserting to left of deject

Adding: master
Looking right of babe
Looking left of rest
Looking right of dunce
Inserting to right of fabric

Adding: lose
Looking right of babe
Looking left of rest
Looking right of dunce
Looking right of fabric
Inserting to left of master

Adding: believe
Looking right of babe
Looking left of rest
Looking left of dunce
Looking left of deject
Inserting to left of chars

Adding: grange
Looking right of babe
Looking left of rest
Looking right of dunce
Looking right of fabric
Looking left of master
Inserting to left of lose

Adding: juice
Looking right of babe
Looking left of rest
Looking right of dunce
Looking right of fabric
Looking left of master
Looking left of lose
Inserting to right of grange

Adding: super
Looking right of babe
Looking right of rest
Looking left of wanton
Inserting to right of shawn

Adding: sort
Looking right of babe
Looking right of rest
Looking left of wanton
Looking right of shawn
Inserting to left of super

Adding: hand
Looking right of babe
Looking left of rest
Looking right of dunce
Looking right of fabric
Looking left of master
Looking left of lose
Looking right of grange
Inserting to left of juice
```

```
Adding: frisbee
Looking right of babe
Looking left of rest
Looking right of dunce
Looking right of fabric
Looking left of master
Looking left of lose
Inserting to left of grange

Adding: running
Looking right of babe
Looking right of rest
Looking left of wanton
Inserting to left of shawn

Adding: older
Looking right of babe
Looking left of rest
Looking right of dunce
Looking right of fabric
Inserting to right of master

Adding: tint
Looking right of babe
Looking right of rest
Looking left of wanton
Looking right of shawn
Inserting to right of super

Adding: appease
Inserting to left of babe

Adding: might
Looking right of babe
Looking left of rest
Looking right of dunce
Looking right of fabric
Looking right of master
Inserting to left of older

Printing in order:
|1:appease|
|9:babe|
|1:believe|
|2:chars|
|3:deject|
|7:dunce|
|6:fabric|
|1:frisbee|
|3:grange|
|1:hand|
|2:juice|
|4:lose|
|5:master|
|1:might|
|2:older|
|8:rest|
|1:running|
|3:shawn|
|1:sort|
```

```
|2:super|
|1:tint|
|4:wanton|

Printing Preorder:
|9:babe|
|1:appease|
|8:rest|
|7:dunce|
|3:deject|
|2:chars|
|1:believe|
|6:fabric|
|5:master|
|4:lose|
|3:grange|
|1:frisbee|
|2:juice|
|1:hand|
|2:older|
|1:might|
|4:wanton|
|3:shawn|
|1:running|
|2:super|
|1:sort|
|1:tint|

Printing Postorder:
|1:appease|
|1:believe|
|2:chars|
|3:deject|
|1:frisbee|
|1:hand|
|2:juice|
|3:grange|
|4:lose|
|1:might|
|2:older|
|5:master|
|6:fabric|
|7:dunce|
|1:running|
|1:sort|
|1:tint|
|2:super|
|3:shawn|
|4:wanton|
|8:rest|
|9:babe|

------------------------------------------------------------------------

(Pt 2) Whole thing with just BST:

How many letters do you want?
8
Your letters are:
o c f l y s v j
Start generating words:
fly
so
```

```
of
fool
soy
coy
joy
sly
toy
blug
-1
fly:0, so:0, of:0, fool:0, soy:0, coy:0, joy:0, sly:0, toy:0, blug:0,
fly is okay  letterwise
fly is in tree
so is okay  letterwise
so is in tree
of is okay  letterwise
of is in tree
fool is invalid letterwise
soy is okay  letterwise
soy NOT in tree
coy is okay  letterwise
coy NOT in tree
joy is okay  letterwise
joy is in tree
sly is okay  letterwise
sly NOT in tree
toy is invalid letterwise
blug is invalid letterwise
fly:1, so:1, of:1, fool:-1, soy:-1, coy:-1, joy:1, sly:-1, toy:-1, blug:-1,
Final Score is: -2
```

(Pt 3) Testdict with AVLflag turned on to true:

```
Adding: babe
Made root
Adding: rest
Inserting to right of babe

Adding: wanton
Looking right of babe
Inserting to right of rest

Rotating left around babe
Height of node before: babe:3
Height of right child before: rest:2
height of babe is now 1height of rest is now 2
rest is new root
Adding: shawn
Looking right of rest
Inserting to left of wanton

Adding: dunce
Looking left of rest
Inserting to right of babe

Adding: fabric
Looking left of rest
Looking right of babe
Inserting to right of dunce

Rotating left around babe
Height of node before: babe:3
```

```
Height of right child before: dunce:2
height of babe is now 1height of dunce is now 2
resetting rest's left to dunce
Adding: deject
Looking left of rest
Looking left of dunce
Inserting to right of babe

Adding: chars
Looking left of rest
Looking left of dunce
Looking right of babe
Inserting to left of deject

Rotating left around babe
right rotate first deject
Height of node before: deject:2
Height of right child before: chars:1
height of deject is now 1height of chars is now 2
Height of node before: babe:3
Height of right child before: chars:2
height of babe is now 1height of chars is now 2
resetting dunce's left to chars
Adding: master
Looking left of rest
Looking right of dunce
Inserting to right of fabric

Adding: lose
Looking left of rest
Looking right of dunce
Looking right of fabric
Inserting to left of master

Rotating left around fabric
right rotate first master
Height of node before: master:2
Height of right child before: lose:1
height of master is now 1height of lose is now 2
Height of node before: fabric:3
Height of right child before: lose:2
height of fabric is now 1height of lose is now 2
resetting dunce's right to lose
Adding: believe
Looking left of rest
Looking left of dunce
Looking left of chars
Inserting to right of babe

Rotating right around rest
Height of node before: rest:5
Height of right child before: dunce:4
height of rest is now 3height of dunce is now 4
dunce is new root
Adding: grange
Looking right of dunce
Looking left of rest
Looking left of lose
Inserting to right of fabric

Adding: juice
```

```
Looking right of dunce
Looking left of rest
Looking left of lose
Looking right of fabric
Inserting to right of grange

Rotating left around fabric
Height of node before: fabric:3
Height of right child before: grange:2
height of fabric is now 1height of grange is now 2
resetting lose's left to grange
Adding: super
Looking right of dunce
Looking right of rest
Looking left of wanton
Inserting to right of shawn

Rotating right around wanton
Left rotate first shawn
Height of node before: shawn:2
Height of right child before: super:1
height of shawn is now 1height of super is now 2
Height of node before: wanton:3
Height of right child before: super:2
height of wanton is now 1height of super is now 2
resetting rest's right to super
Adding: sort
Looking right of dunce
Looking right of rest
Looking left of super
Inserting to right of shawn

Adding: hand
Looking right of dunce
Looking left of rest
Looking left of lose
Looking right of grange
Inserting to left of juice

Rotating right around lose
Left rotate first grange
Height of node before: grange:3
Height of right child before: juice:2
height of grange is now 2height of juice is now 3
Height of node before: lose:4
Height of right child before: juice:3
height of lose is now 2height of juice is now 3
resetting rest's left to juice
Adding: frisbee
Looking right of dunce
Looking left of rest
Looking left of juice
Looking left of grange
Inserting to right of fabric

Rotating left around dunce
right rotate first rest
Height of node before: rest:5
Height of right child before: juice:4
height of rest is now 4height of juice is now 5
Height of node before: dunce:6
```

```
Height of right child before: juice:5
height of dunce is now 4height of juice is now 5
juice is new root
Adding: running
Looking right of juice
Looking right of rest
Looking left of super
Inserting to left of shawn

Adding: older
Looking right of juice
Looking left of rest
Looking right of lose
Inserting to right of master

Rotating left around lose
Height of node before: lose:3
Height of right child before: master:2
height of lose is now 1height of master is now 2
resetting rest's left to master
Adding: tint
Looking right of juice
Looking right of rest
Looking right of super
Inserting to left of wanton

Adding: appease
Looking left of juice
Looking left of dunce
Looking left of chars
Inserting to left of babe

Adding: might
Looking right of juice
Looking left of rest
Looking right of master
Inserting to left of older

Printing in order:
|1:appease|
|2:babe|
|1:believe|
|3:chars|
|1:deject|
|4:dunce|
|2:fabric|
|1:frisbee|
|3:grange|
|1:hand|
|5:juice|
|1:lose|
|3:master|
|1:might|
|2:older|
|4:rest|
|1:running|
|2:shawn|
|1:sort|
|3:super|
|1:tint|
|2:wanton|
```

```
Printing Preorder:
|5:juice|
|4:dunce|
|3:chars|
|2:babe|
|1:appease|
|1:believe|
|1:deject|
|3:grange|
|2:fabric|
|1:frisbee|
|1:hand|
|4:rest|
|3:master|
|1:lose|
|2:older|
|1:might|
|3:super|
|2:shawn|
|1:running|
|1:sort|
|2:wanton|
|1:tint|

Printing Postorder:
|1:appease|
|1:believe|
|2:babe|
|1:deject|
|3:chars|
|1:frisbee|
|2:fabric|
|1:hand|
|3:grange|
|4:dunce|
|1:lose|
|1:might|
|2:older|
|3:master|
|1:running|
|1:sort|
|2:shawn|
|1:tint|
|2:wanton|
|3:super|
|4:rest|
|5:juice|
----------------------------------------------------------------------
(Pt 4) Whole thing with AVL Tree (NOTE: If this works, it should look just like the output
with the BST.  However, you still need to make sure it works):
How many letters do you want?
8
Your letters are:
o h h r x w m s
Start generating words:
so
mow
sow
or
```

```
show
mosh
ohm
room
sox
glug
-1
so:0, mow:0, sow:0, or:0, show:0, mosh:0, ohm:0, room:0, sox:0, glug:0,
so is okay  letterwise
so is in tree
mow is okay  letterwise
mow NOT in tree
sow is okay  letterwise
sow NOT in tree
or is okay  letterwise
or is in tree
show is okay  letterwise
show is in tree
mosh is okay  letterwise
mosh NOT in tree
ohm is okay  letterwise
ohm NOT in tree
room is invalid letterwise
sox is okay  letterwise
sox NOT in tree
glug is invalid letterwise
so:1, mow:-1, sow:-1, or:1, show:1, mosh:-1, ohm:-1, room:-1, sox:-1, glug:-1,
Final Score is: -4
```