Design

```
#include <iostream>
#define COUNT 10//used for creating space in printColumn() and other similar private
//methods. Also used for main for loop
#define RANDM 100     //used for testing for random
#include <time.h>   //using time(NULL)//
#include <stdlib.h> //using srand and rand//

using namespace std;

template <typename key, typename info>

class Dictionary{

    private:

        struct AVLnode

        {

            key ID;

            info Data;

            int bfactor; //balance factor

            AVLnode *llink;//link to the left

            AVLnode *rlink;//link to the right

    };

AVLnode *root;


int height(AVLnode *p) const;//returns the maximum of the height of the left and right
subtree


void copyTree(AVLnode* &copied, AVLnode* other);//recursive private method used to copy
subtree

void insertIntoAVL(AVLnode* &root,AVLnode *newNode, bool& isTaller);

//recursive private method used to insert node to AVL Tree, uses balanceRight and
balanceLeft to correct the subtree after insertion of node. Also acts as append function
because duplicates are not allowed but it can write over a key with new info. root is root
of the subtree, newNode is the added node, isTaller is updated for the switch-case
functions and it checks if the subtree became bigger or not after the insertion of newNode.


void removeFromAVL(AVLnode* &curRoot, key & k, AVLnode* & toSwapWith, AVLnode* & newLink,
bool & isSmaller, bool & wasDeleted);

//recursive private method used to remove node from AVL tree, uses balanceRight and
balanceLeft to correct the subtree after removal of node. curRoot is root of the current
subtree, k is the key to be removed, toSwapWith is the node that copies the current node,
newLink is the node copies either the right link or left link of the current node,
isSmaller is updated for the switch-case functions and it checks if the subtree became
smaller or not after the removal of the node, wasDeleted is true when a node is deleted
from the subtree.

//continued in next page
```

```
//extra methods with no use aside from utility
AVLnode * minValueNode(AVLnode* p);//returns leftmost value in tree
AVLnode * maxValueNode(AVLnode* p);//returns rightmost value in tree
int max(int x, int y) const;//utility method that returns larger of x and y, used for int height
int nodeCount(AVLnode *p) const;//returns number of nodes in the tree that p points to
int leavesCount(AVLnode *p) const;//returns number of leaves in the tree that p points to


//used for destroyer
void destroy(AVLnode *p);//destroys subtree



//recursive private printing methods
void inorder(AVLnode *p) const;//prints subtree inorder
void preorder(AVLnode *p) const;//prints subtree preorder
void postorder(AVLnode *p) const;//prints subtree postorder
//rotating methods
void rotateRight(AVLnode * &root);//right subtree of left subtree of root becomes left subtree of root
void rotateLeft(AVLnode* &root);//left subtree of right subtree of root becomes right subtree of root
//balancing methods
void balanceLeft(AVLnode* &root)//uses rotateLeft and rotateRight to balance trees by checking balance factor of left link of root
void balanceRight(AVLnode* &root);//uses rotateLeft and rotateRight to balance trees by checking balance factor of right link of root
//print functions used for public graphical print
void printRow(AVLnode* p, int level);//recursive method used for printVert
void printRowInfo(AVLnode* p, int level);//recursive method used for printVertInfo
void printColumn(AVLnode *p, int space);//recursive method used for printHori
void printColumnInfo(AVLnode *p, int space);//recursive method used for printHoriInfo
void printColumnDetail(AVLnode *p, int space);//recursive method used for printHoriDetail
void printHoriDetail();////prints all nodes of tree with key,info,balance factor in horiztonal way
//end of private methods
```

```
	public:


const Dictionary<key,info>& operator=(const Dictionary<key,info>& D);//overloading of assignment operator

bool isEmpty() const;//returns 1 if tree is empty, 0 if not

void inorderTraversal() const;//prints tree inorder by using inorder private method

void preorderTraversal() const;//prints tree preorder by using preorder private method

void postorderTraversal() const;//prints tree postorder by using postorder private method

int treeHeight() const;//returns height of tree by using private height method

int treeNodeCount() const;//returns number of nodes in tree by using private nodeCount method

int treeLeavesCount() const;//returns number of leaves by using private leavesCount method

void destroyTree();//deallocates memory space occupied by AVL tree, uses private destroy method and works identically to destructor

Dictionary(const Dictionary<key,info> &D);//copy constructor

Dictionary();//default constructor

~Dictionary();//destructor using private destroy function


bool search(const key& item) const;//searches tree for key, returns 1 if found, returns 0 if not.


void insert(const key &newItem, const info &newData);//using isTaller = false, it inserts/appends tree using the private method insertIntoAVL.


void remove(key k);//using isSmaller = false, wasDeleted = false, it removes node from the tree containing key by using the private method removeFromAVL


//five graphical print functions


void printVert();//prints key in vertical way

void printHori();//prints key in horizontal way

void printVertInfo();//prints key along with info, in vertical way

void printHoriInfo();//prints key along with info, in horiztonal way

void printDetail();//prints all nodes of tree with key,info,balance factor in horiztonal way and also prints height of tree.

//end of public methods

};//end of class Dictionary

int rndom(int r = RANDM){return (rand()%r);}//utility function for main
```

Implementation

The AVL tree has mostly private methods that are recursive and operate on a subtree. The public methods use these private methods and operate on the root (meaning the entire tree instead of just a subtree). There are 3 types of print methods, horizontal print, vertical print, linear print. Horizontal is the main method of printing, with three separate variations (printing just key, key and info, and key, info, balance factor, height, number of nodes and leaves). Vertical is not a good method, but included anyway for another representation, and has two variations (printing just key and printing both key and info). Linear printing is just printing inorder, preorder, postorder.

Testing

```
int main()
{
    srand(time(NULL));//initializing random
    srand(0);//can also use values that remain the same throughout
    Dictionary<int,int> A;
    int i = 0;//counter
    int j,k;//stores random value to show if anything gets appended or not
    for(i;i<COUNT;i++){//count = 10
        j = rndom();//random number from 0 to 100
        k = rndom();//random number from 0 to 100
        A.insert(j,k);//inserts random int key, info
        cout<<i+1<<"\tnode inserted "<<j<<" and "<<k<<endl;
    }
      cout<<endl<<endl;
      A.printDetail();


      A.remove(j);//removes last node added to show remove function
      A.printDetail();
      return 0;
}
```