template<typename key,typename info>

class Dictionary{

…

};

class Dictionary is implemented as an AVL(self-balancing) tree.

The private part includes a **node** pointer to **root** of tree and unsigned integer variable **size,** a struct **node** and helper methods:

- **struct node{**
  key nkey;
  info ninfo;
  int height;
  node* right;
  node* left;
  };
- **int height(node* N)const;** // Returns the heights of the node
- **int getBalance(node *N);** // Returns the balance factor of the node (left height - right height)
- **node* FindMin(node* root);** // Returns pointer to the node with minimum key
- **node* rotateRight(node *n);** // Right rotation
- **node* rotateLeft(node *n);** // Left rotation
- **node* balance(node* nptr);** // Tree balancing method
- **node* insert(node* nptr,const key& k,const info& i);** // Helper method for insert
- **void recInsert(node *nptr);** // Helper method for recursive insert, uses the above method
- **node* deleteNode(node* nptr, const key& k);** // Helper method for deleting a node
- **void preOrder(node * root)const;** // Helper method for pre order print
- **void inOrder(node *root)const;** // Helper method for in order print
- **void postOrder(node * root)const;** // Helper method for post order print
- **node* find(node* nptr,const key& k)const;** // Returns a pointer to the node if its key is found
- **bool search(node* nptr,const key& k)const;** // Helper method for searching, uses the method above
- **bool search(const key& k)const;** // Returns true if given key found, false other wise
- **node* copy(node * orig)const;** // Helper method for copying all children of the node provided
- **void clearAll(node* nptr);** // method for deleting all nodes from tree, used in destructor
- **bool compare(node *n) const;** // Comparison method, used in comparison operator
- **void help_print(ostream &out_s, node *r)const;** // Helper method for print
- **void print(ostream& out_s)const;** // Helper method for print

Public part:

- **Dictionary();** // Constructor
- **~ Dictionary;** // Destructor
- **Dictionary (const Dictionary <key,info>& rhs);** // Copy Constructor
- **Dictionary & operator=(const Dictionary <key,info>& rhs);** // Assignment operator
- **void insert(const key& k, const info& i);** // Inserts a node, error if key already exists
- **void remove(const key& nkey);** // Removes the given key, error if key not found
- **const Info getInfo(const key& k) const;** //Returns the info associated with the given key, error if key not found
- **Dictionary& operator +=(const Dictionary& rhs);** // Operator adds the nodes of rhs tree into current, if rhs and current are same then an unchanged tree is returned
- **Dictionary operator +(const Dictionary& rhs)** // Operator adds the nodes of current and rhs tree into a new tree which is returned
- **bool operator == (const Dictionary& rhs)const** // Comparison operator, only compares the keys of the nodes
- **bool operator !=(const Dictionary& rhs)const** // Comaprison operator
- **enum Order {pre,in,post};** // Global enum decleration used in print function
- **void print(int Order)const** // Order; 0=preOrder, 1=inOrder, 2=postOrder displaying of the tree
- **friend ostream& operator << (ostream &out_s,const Dictionary &t)** // output operator which displays the tree in preOrder