Lab 11

Complete the missing parts in AvlTree.cpp file. Write a main method to test your code. Test if the insert method works properly.

You can find the following files on the online learning system.

- AvlTree.h
- AvlNode.h
- AvlTree.cpp

Node declaration for AVL trees

```
template <class T>
class AvlNode
{
   T element;
   AvlNode   *left;
   AvlNode   *right;
   int       height;

   AvlNode( const T & theElement, AvlNode *lt = NULL,
            AvlNode *rt = NULL, int h = 0 )
     : element( theElement ), left( lt ), right( rt ),
                height( h ) { }
};
```

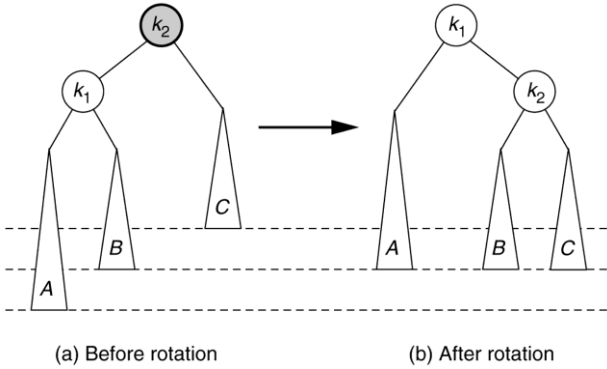
Height

```
template <class T>
int height( const AvlNode<T> *t)
{
   return t == NULL ? -1 : t->height;
}
```

Single right rotation

Case1: An insertion in the left subtree of the left child of X

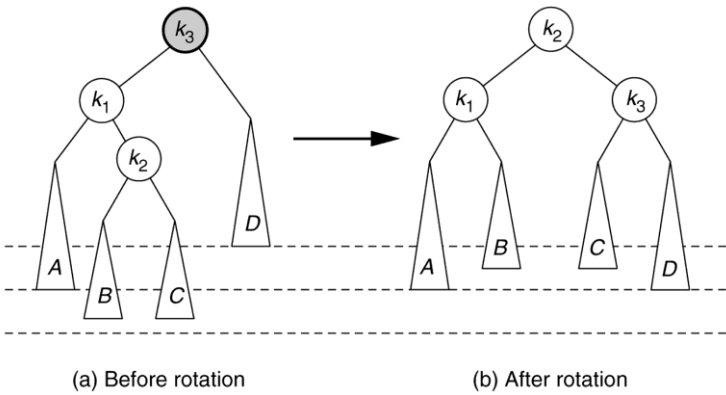
```
/**
 * Rotate binary tree node with left child.
 * For AVL trees, this is a single rotation for case 1.
 * Update heights, then set new root.
 */
template <class T>
void rotateWithLeftChild( AvlNode<T> *& k2 )
{
    AvlNode<T> *k1 = k2->left;
    k2->left = k1->right;
    k1->right = k2;
    k2->height = max( height( k2->left ), height( k2->right ))+1;
    k1->height = max( height( k1->left ), k2->height ) + 1;
    k2 = k1;
}
```



Double Rotation

Case 2: An insertion in the right subtree of the left child of X

```
/**
 * Double rotate binary tree node: first left child.
 * with its right child; then node k3 with new left child.
 * For AVL trees, this is a double rotation for case 2.
 * Update heights, then set new root.
 */
template <class T>
void doubleWithLeftChild( AvlNode<T> *& k3 )
{
    rotateWithRightChild( k3->left );
    rotateWithLeftChild( k3 );
}
```



AVL Trees -- Insertion

```
/* Internal method to insert into a subtree.
 * x is the item to insert; t is the node that roots the tree.
 */
template <class T>
void insert( const T& x, AvlNode<T> *& t )
{
    if( t == NULL )
      t = new AvlNode<T>(x);
    else if( x < t->element )
    {
      insert( x, t->left );
      if( height( t->left ) - height( t->right ) == 2 )
        if( x < t->left->element )
            rotateWithLeftChild( t );  // case 1
        else
            doubleWithLeftChild( t );  // case 2
    }
    else if( t->element < x )
    {
        insert( x, t->right );
        if( height( t->right ) - height( t->left ) == 2 )
           if( t->right->element < x )
              rotateWithRightChild( t );  // case 4
           else
              doubleWithRightChild( t );  // case 3
     }
     else
        ;  // Duplicate; do nothing
     t->height = max( height( t->left ), height( t->right ) ) + 1;
}
```