Data Structures 2 - Lab 2

Name : Ahmed Khaled

ID : 9

Requirements :

-Implementing AVL Tree data structure supporting :
Searching , Deletion , Insertion and printing the height of the tree .

-Implementing English dictionary supporting :
-Loading words of dictionary from a file .
-inserting a non-repeated word in the dictionary .
-Looking-up for a word in the dictionary.
-Printing the size of the dictionary .
-Looking up for a file of words .
-Deleting a file of words from the dictionary .

Implementation details :

1-Searching :

```
Find(Node n , Object o)
{
   If(n=null)
         Return null; //Not Found !

   If(o<n.element)
      Find(n.left,o);
   Else if (o>n.element)
     Find(n.right,o)
  Else // the element is found
      Return n;
}
```

2-Insertion :

```
Insert(Node root , Object o )
{
    If(root=null)
          Return new Node(o)  // insert here
   If(o<root.element)
       Root.left =Insert(root.left,o)
  Else if(o>root.element)
       Root.right = Insert(root.right,o)
  Else
      Throw  Exception("Duplicate")

 updateHeight(n)
 return balance(n)

}
```

3- Deletion :

```
Delete(Node root , Object o)
  {
     If(root=null)
         Throw Exception("Not Found")  // insert here
     If(o<root.element)
       Root.left =Delete(root.left,o)
    Else if(o>root.element)
       Root.right = Delete (root.right,o)

   Else  // delete this node
     {
        If(has no children)
            Return   Root = null
       else If(has one child)
              Root = its child
       Else // has two nodes
            Node successor = getSuccessor(root)
            Replace(root , successor);


     }

  updateHeight(root);
  return balance(root);

  updateHeight(n)
  return balance(n)
}
```

4- Balancing :

```
Balance(Node n)
{
   bf=balanceFactor(n)

  if(bf<-1) // right
 {
        Bf=balanceFactor(n.right)

        If(bf<0) //right right case
            Return rightSingleRotation(n)
        Else //right left case
            Return rightDoubleRotation(n)

 }
Else if(bf > 1)  // left
{
     Bf = balanceFactor(n.left)
    If(bf <0) //left right case
        Return leftDoubleRotation(n)
    Else   // left left case
         Return leftSingleRotation(n)
}
```

Double Rotations :

```
rightDoubleRotation(n)
{
    n.right = leftSingleRotation(n.right)

    return  rightSingleRotation(n)
}
```

```
leftDoubleRotation(n)
{
    n.left = rightSingleRotation(n.left)

     return  leftSingleRotation(n)
}
```

Single Rotations :

```
rightSingleRotation()
{
      Right = n.right ;
      n.right = right.left
      right.left = n

   updateHeight(n)
   updateHeight(right)

    return right ;
}
```

```
leftSingleRotation()
{
      left = n.left ;
      n.left = right.right
      left.right = n

   updateHeight(n)
   updateHeight(left)

    return left ;

}
```