```
typedef struct AVLTreeNode *AVLTree;
typedef struct AVLTreeNode{
    ElementType Data;
    AVLTree Left;
    AVLTree Right;
    int Height;
};

AVLTree AVL_Insertion ( ElementType X, AVLTree T )
{ /* 将 X 插入 AVL 树 T 中，并且返回调整后的 AVL 树 */
    if ( !T ) { /* 若插入空树，则新建包含一个结点的树 */
        T = (AVLTree)malloc(sizeof(struct AVLTreeNode));
        T->Data = X;
        T->Height = 0;
        T->Left = T->Right = NULL;
    } /* if (插入空树) 结束 */

    else if (X < T->Data) { /* 插入 T 的左子树 */
        T->Left = AVL_Insertion(X, T->Left);
        if (GetHeight(T->Left) - GetHeight(T->Right) == 2 )
            /* 需要左旋 */
            if (X < T->Left->Data)
                T = SingleLeftRotation(T);          /* 左单旋 */
            else
                T = DoubleLeftRightRotation(T); /* 左-右双旋 */
    } /* else if (插入左子树) 结束 */

    else if (X > T->Data) { /* 插入 T 的右子树 */
        T->Right = AVL_Insertion(X, T->Right);
        if (GetHeight(T->Left) - GetHeight(T->Right) == -2 )
            /* 需要右旋 */
            if (X > T->Right->Data)
                T = SingleRightRotation(T);         /* 右单旋 */
            else
                T = DoubleRightLeftRotation(T); /* 右-左双旋 */
    } /* else if (插入右子树) 结束 */

    /* else X == T->Data，无须插入 */

    T->Height = Max(GetHeight(T->Left),GetHeight(T->Right))+1;
    /*更新树高*/

    return T;
}
```

```
AVLTree SingleLeftRotation ( AVLTree A )
{     /* 注意：A 必须有一个左子结点 B */
      /* 将 A 与 B 做如图 4.35 所示的左单旋，更新 A 与 B 的高度，返回新的根结点 B */

      AVLTree B = A->Left;
      A->Left = B->Right;
      B->Right = A;
      A->Height = Max(GetHeight(A->Left), GetHeight(A->Right))+1;
      B->Height = Max(GetHeight(B->Left), A->Height)+1;

      return B;
}


AVLTree DoubleLeftRightRotation ( AVLTree A )
{ /* 注意：A 必须有一个左子结点 B，且 B 必须有一个右子结点 C */
   /* 将 A、B 与 C 做如图 4.38 所示的两次单旋，返回新的根结点 C */

   A->Left = SingleRightRotation(A->Left); /*将 B 与 C 做右单旋，C 被返回*/

   return SingleLeftRotation(A); /*将 A 与 C 做左单旋，C 被返回*/
}
```