```
struct AVLNode{
int key;
int bf;//bf = height(right subtree) – height(left subtree)
int size; //Total number of nodes in the subtree
struct AVLNode *leftChild;
struct AVLNode *rightChild;
};
typedef struct AVLNode AVLNode;
typedef struct AVLNode * AVLNodePtr;
void createAVLTree(AVLNodePtr *root)*root = NULL;}
AVLNodePtr getAVLTreeNode();
[5] void displayAVLTree(AVLNodePtr root, char *fileName);
[80] void AVLInsert(AVLNodePtr *root, int data); //Implement non recursively
[5] void deleteTree(AVLNodePtr *root); //Delete the tree
[10] commenting and good programming practices
/* Note: You can’t calculate the height of the tree. Also need to update size while doing rebalancing – all calculations
should be local.Your code should be commented heavily so that an evaluator can understand the logic. No marks for
partial implementation. */
```

```
struct AVLNode{
int key;
int bf;//bf = height(right subtree) – height(left subtree)
int size; //Total number of nodes in the subtree
struct AVLNode *leftChild;
struct AVLNode *rightChild;
};
typedef struct AVLNode AVLNode;
typedef struct AVLNode * AVLNodePtr;
void createAVLTree(AVLNodePtr *root)*root = NULL;}
AVLNodePtr getAVLTreeNode();
[5] void displayAVLTree(AVLNodePtr root, char *fileName);
[80] void AVLInsert(AVLNodePtr *root, int data); //Implement non recursively
[5] void deleteTree(AVLNodePtr *root); //Delete the tree
[10] commenting and good programming practices
/* Note: You can’t calculate the height of the tree. Also need to update size while doing rebalancing – all calculations
should be local.Your code should be commented heavily so that an evaluator can understand the logic. No marks for
partial implementation. */
```

```
struct AVLNode{
int key;
int bf;//bf = height(right subtree) – height(left subtree)
int size; //Total number of nodes in the subtree
struct AVLNode *leftChild;
struct AVLNode *rightChild;
};
typedef struct AVLNode AVLNode;
typedef struct AVLNode * AVLNodePtr;
void createAVLTree(AVLNodePtr *root)*root = NULL;}
AVLNodePtr getAVLTreeNode();
[5] void displayAVLTree(AVLNodePtr root, char *fileName);
[80] void AVLInsert(AVLNodePtr *root, int data); //Implement non recursively
[5] void deleteTree(AVLNodePtr *root); //Delete the tree
[10] commenting and good programming practices
/* Note: You can’t calculate the height of the tree. Also need to update size while doing rebalancing – all calculations
should be local.Your code should be commented heavily so that an evaluator can understand the logic. No marks for
partial implementation. */
```

```
struct AVLNode{
int key;
int bf;//bf = height(right subtree) – height(left subtree)
int size; //Total number of nodes in the subtree
struct AVLNode *leftChild;
struct AVLNode *rightChild;
};
typedef struct AVLNode AVLNode;
typedef struct AVLNode * AVLNodePtr;
void createAVLTree(AVLNodePtr *root)*root = NULL;}
AVLNodePtr getAVLTreeNode();
[5] void displayAVLTree(AVLNodePtr root, char *fileName);
[80] void AVLInsert(AVLNodePtr *root, int data); //Implement non recursively
[5] void deleteTree(AVLNodePtr *root); //Delete the tree
[10] commenting and good programming practices
/* Note: You can’t calculate the height of the tree. Also need to update size while doing rebalancing – all calculations
should be local.Your code should be commented heavily so that an evaluator can understand the logic. No marks for
partial implementation. */
```