```c
AVLTree *rotacao_esquerda(AVLTree *arvore) {
  AVLTree *dir = arvore->dir;
  arvore->dir = dir->esq;
  dir->esq = arvore;
  return dir;
}

AVLTree *rotacao_direita(AVLTree *arvore) {
  AVLTree *esq = arvore->esq;
  arvore->esq = esq->dir;
  esq->dir = arvore;
  return esq;
}
```

```c
int maior(int a, int b) {
  return a > b ? a : b;
}
```

```c
void inserir_ordenadoV2(AVLTree **arvore, int dado) {
  if(!*arvore) { // *arvore == NULL
    *arvore = novaArvore(dado);
    return;
  }
  // exceção novo dado = dado do nó
  if(dado == (*arvore)->dado) return;
  // inserir dado de forma ordenada
  if(dado > (*arvore)->dado) inserir_ordenadoV2(&(*arvore)->dir, dado);
  else inserir_ordenadoV2(&(*arvore)->esq, dado);
  // atualizar altura
  (*arvore)->altura = maior(height((*arvore)->esq), height((*arvore)->dir)) + 1;
  // verificar fator de balanceamento
  // balancear se estiver desbalanceado
  int bf = balancingFactor(*arvore);
  if(bf <= -2 && (*arvore)->dir->dir) { // rotacionar para esquerda
    *arvore = rotacao_esquerda(*arvore);
    AVLTree *esq = (*arvore)->esq;
    esq->altura =  maior(height(esq->esq), height(esq->dir)) + 1;
  } else if(bf >= 2 && (*arvore)->esq->esq) { // rotacionar para direita
    *arvore = rotacao_direita(*arvore);
    AVLTree *dir = (*arvore)->dir;
    dir->altura = maior(height(dir->esq), height(dir->dir)) + 1;
  } else if(bf >= 2) { // rotação dupla esquerda-direita
    (*arvore)->esq = rotacao_esquerda((*arvore)->esq);
    *arvore = rotacao_direita(*arvore);
    AVLTree *dir = (*arvore)->dir;
    dir->altura = maior(height(dir->esq), height(dir->dir)) + 1;
    AVLTree *esq = (*arvore)->esq;
    esq->altura =  maior(height(esq->esq), height(esq->dir)) + 1;
  } else if(bf <= -2) { // rotacao dupla direita-esquerda
    (*arvore)->dir = rotacao_direita((*arvore)->dir);
    *arvore = rotacao_esquerda(*arvore);
    AVLTree *dir = (*arvore)->dir;
    dir->altura = maior(height(dir->esq), height(dir->dir)) + 1;
    AVLTree *esq = (*arvore)->esq;
    esq->altura =  maior(height(esq->esq), height(esq->dir)) + 1;
  } else return;
  (*arvore)->altura = maior(height((*arvore)->esq), height((*arvore)->dir)) + 1;
}
```

```c
void exibir_em_ordem_fb(AVLTree *tree) {
    if(!tree) return;
    exibir_em_ordem_fb(tree->esq);
    printf("fb(%d) = %d\n", tree->dado, balancingFactor(tree));
    exibir_em_ordem_fb(tree->dir);
}

void exibir_pre_ordem_fb(AVLTree *tree) {
    if(!tree) return;
    printf("fb(%d) = %d\n", tree->dado, balancingFactor(tree));
    exibir_pre_ordem_fb(tree->esq);
    exibir_pre_ordem_fb(tree->dir);
}
```

```c
#ifndef TADAVL2_H
#define TADAVL2_H

typedef struct avltree AVLTree;

struct avltree {
    AVLTree *esq;
    AVLTree *dir;
    int altura;
    int dado;
};

AVLTree *novaArvore(int dado);
int balancingFactor(AVLTree *);
int height(AVLTree *avltree);
AVLTree *rotateLeft(AVLTree *avltree);
AVLTree *rotateRight(AVLTree *avltree);
AVLTree *insertNode(AVLTree *avltree, int dado);
AVLTree *deleteNode(AVLTree *avltree, int dado);

// exibir
void preOrder(AVLTree *avltree);
void inOrder(AVLTree *avltree);
void posOrder(AVLTree *avltree);

#endif
```