#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

typedef struct NoAVL \*PtrNoAVL;

typedef struct NoAVL

{

PtrNoAVL dir, esq;

int chave;

int altura;

} NoAVL;

void iniciarArvoreAVL(PtrNoAVL \*raiz)

{

(\*raiz) = NULL;

}

bool estaVazia(PtrNoAVL \*raiz)

{

return ((\*raiz) == NULL);

}

void preOrdemArvoreAVL(PtrNoAVL \*no)

{

if ((\*no) == NULL)

return;

printf("%d ", (\*no)->chave);

preOrdemArvoreAVL(&(\*no)->esq);

preOrdemArvoreAVL(&(\*no)->dir);

}

void principalOrdemArvoreAVL(PtrNoAVL \*no)

{

printf("pre ordem = {");

preOrdemArvoreAVL(no);

printf("}");

}

int alturaArvoreAVL(PtrNoAVL \*no)

{

if ((\*no) == NULL)

return 0;

else

{

return ((\*no)->altura);

}

}

int atualizaAltura(PtrNoAVL \*no)

{

int ae, ad;

ae = alturaArvoreAVL(&(\*no)->esq);

ad = alturaArvoreAVL(&(\*no)->dir);

if (ae > ad)

return (ae + 1);

else

{

return (ad + 1);

}

}

// entra se rola desbalanceamento +2 || -2

void aplicaRotacoes(PtrNoAVL \*no)

{

// dir ou esquerda

int ae, ad, fatorbalanceamento;

ae = alturaArvoreAVL(&(\*no)->esq);

ad = alturaArvoreAVL(&(\*no)->dir);

fatorbalanceamento = ad - ae;

PtrNoAVL u;

int aUdir, aUesq;

if (fatorbalanceamento == 2)

{

// rotação para esquerda simples ou dupla

u = (\*no)->dir;

aUdir = alturaArvoreAVL(&u->dir);

aUesq = alturaArvoreAVL(&u->esq);

if (aUdir>=aUesq)

{

/\* rotação simples \*/

rotacaoSimplesEsquerda(&(\*no));

}

else{

//rotação dupla

rotacaoDuplasEsquerda(&(\*no));

}

}

else //==-2

{

// rotação prar direita simples ou dupla

u = (\*no)->esq;

aUdir = alturaArvoreAVL(&u->dir);

aUesq = alturaArvoreAVL(&u->esq);

if (aUesq>=aUdir)

{

/\* rotação simples\*/

rotacaoSimplesDireita(&(\*no));

}else{

rotacaoDuplaDireita(&(\*no));

}

}

}

bool insereArvore(PtrNoAVL \*no, int valor)

{

if ((\*no) == NULL)

{

(\*no) = malloc(sizeof(NoAVL))

(\*no)

->chave = valor;

(\*no)->dir = NULL;

(\*no)->esq = NULL;

(\*no)->altura = 1;

return true;

}

if ((\*no)->chave == valor)

return false;

// chamada recursiva

bool ret;

if (valor > (\*no)->chave)

{

// rec dir

ret = insereArvore(&(\*no)->dir, valor);

}

else

{

/\* rec esq \*/

ret = insereArvore(&(\*no)->esq, valor);

}

// AGORA É A MUDANÇA DA IMPLEMENTAÇÃO AVL

if (!ret)

return false;

// passou do if aconteceu inserção e pode ter desbalanceamento

int alturaDir, alturaEsq, fatorBalanceamento;

alturaDir = alturaArvoreAVL(&(\*no)->dir);

alturaEsq = alturaArvoreAVL(&(\*no)->esq);

fatorBalanceamento = alturaDir - alturaEsq;

if (fatorBalanceamento > 1 || fatorBalanceamento < -1)

{

aplicarRotacoes(&(\*no));

}

(\*no)->altura = atualizaAltura(&(\*no)->esq, &(\*no)->dir);

return true;

}

void RSD(PtrNoAVL \*p){

PtrNoAVL v = (\*p)->esq;

(\*p)->esq = v->dir;

v->dir = (\*p);

(\*p)->altura = atualizaAltura(&(\*p)->esq, &(\*p)->dir);

v->altura = atualizaAltura(&(v->esq),&(v->dir));

(\*p)=v;

}

void RSE(PtrNoAVL \*p){

PtrNoAVL v = (\*p)->dir;

(\*p)->dir = v->esq;

v->esq = (\*p);

(\*p)->altura = atualizaAltura(&(\*p)->esq, &(\*p)->dir);

v->altura = atualizaAltura(&(v->esq),&(v->dir));

(\*p)=v;

}

void RDD(PtrNoAVL \*p){

}

int main(int argc, char const \*argv[])

{

PtrNoAVL raiz;

iniciarArvoreAVL(&raiz);

if (estaVazia(&raiz))

{

printf("mudinha ainda");

}

return 0;

}