#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

// https://portaldoprofessor.fct.unesp.br/projetos/cadilag/apps/structs/arv\_binaria.php

typedef struct NoArvore \*PtrNoArvore;

typedef struct NoArvore

{

int chave;

PtrNoArvore esquerda;

PtrNoArvore direita;

} NoArvore;

// iniciar

void iniciaArvore(PtrNoArvore \*arvore)

{

\*arvore = NULL;

}

// esta vazia

bool estaVaziaArvore(PtrNoArvore \*arvore)

{

return (\*arvore == NULL);

}

// destruir

// pesquisa

bool pesquisarArvore(PtrNoArvore \*no, int valor)

{

if ((\*no) == NULL)

{

return false;

}

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

{

return true;

}

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

{

return pesquisarArvore(&(\*no)->esquerda, valor); // acessar subarvore esquerda

}

else

{

return pesquisarArvore(&(\*no)->direita, valor); // acessar subarvore direita

}

}

// inserir elem

bool inserirArvore(PtrNoArvore \*no, int x)

{

if ((\*no) == NULL) // ponto de inserção

{

(\*no) = malloc(sizeof(NoArvore));

(\*no)->direita = NULL;

(\*no)->esquerda = NULL;

(\*no)->chave = x;

return true; // quando deu bom

}

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

{

return false; // não inserir numero igual

}

// PERCORRER ARVORE COM RECURSÃO

if (x < (\*no)->chave) // subarvore esquerda

{

return (inserirArvore(&(\*no)->esquerda, x));

}

else // x> acessar subarvore direita

{

return (inserirArvore(&(\*no)->direita, x));

}

}

// remover elem

PtrNoArvore funcaoAuxiliar(PtrNoArvore \*no)//RECURSIVA

{

while ((\*no)->direita!=NULL)

{

(\*no)=(\*no)->direita;

}

PtrNoArvore ret = (\*no);

(\*no)=ret->esquerda;

return ret;

}

bool removeArvore(PtrNoArvore \*no, int valor)

{

/\*

pode encontrar valor null

encontrou valor

caso 1: folha

no pra null

caso 2: subarvore esqueda

no recebe no esquerda

caso 3: subarvore direita

no recebe no direito

caso 4:

2 possibilidade

A maior elem subarvore3 esq

B maior elem subarvore3 dir

função auxiliar

procura maxmo na subarvore esquerda

encontra elemento cujo filho da direita é null

no=

\*/

if ((\*no) == NULL)

{

return false;

}

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

{

PtrNoArvore tmp;

if (((\*no)->direita || (\*no)->esquerda) == NULL)

{

(\*no) = NULL;

}

else

{

if ((\*no)->direita == NULL && (\*no)->esquerda != NULL)

{

(\*no)=(\*no)->esquerda;

}

else if ((\*no)->direita != NULL && (\*no)->esquerda == NULL)

{

(\*no)=(\*no)->direita;

}

else

{

tmp = funcaoAuxiliar(&(\*no)->esquerda);

(\*no)->chave = tmp->chave;

}

}

free(tmp);

}

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

{

return (removeArvore(&(\*no)->esquerda, valor));

}

}

// pesquisa

// em ordem

void emOrdem(PtrNoArvore \*no)

{

if (\*no == NULL)

{

return;

}

emOrdem(&(\*no)->esquerda);

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

emOrdem(&(\*no)->direita);

}

// pre ordem

void preOrdem(PtrNoArvore \*no)

{

if (\*no == NULL)

{

return;

}

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

preOrdem(&(\*no)->esquerda);

preOrdem(&(\*no)->direita);

}

// pos ordem

void posOrdem(PtrNoArvore \*no)

{

if (\*no == NULL)

{

return;

}

posOrdem(&(\*no)->esquerda);

posOrdem(&(\*no)->direita);

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

}

//////////////////////////////////////////////////

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

{

PtrNoArvore raiz; // COMEÇAR UMA ARVORE

int toinsert[] = {2, 10, 5, 7, 6, 14, -1, -12, 8, 10};

int consult[] = {5, 7, 3, 12};

iniciaArvore(&raiz);

if (estaVaziaArvore(&raiz))

{

printf("Arvore vazia\n");

}

for (size\_t i = 0; i < 10; i++)

{

inserirArvore(&raiz, toinsert[i]);

}

preOrdem(&raiz);

printf("\n");

emOrdem(&raiz);

printf("\n");

posOrdem(&raiz);

printf("\n");

for (size\_t i = 0; i <= 3; i++)

{

if (pesquisarArvore(&raiz, consult[i]))

{

printf("Valor %d foi encontrado :)\n", consult[i]);

}

else

{

printf("Valor %d nao foi encontrado :(\n", consult[i]);

}

}

return 0;

}