**Nomes:**

**Henrique Sartori Siqueira 19240472**

**Rafael Silva Barbon 19243633**

**EX1:** Faça um programa que permita testar todas as funcionalidades de implementação de uma árvore binária solicitadas pela profa. Lúcia:

- criar

- vazia

- inserir

- remover

- buscar

- liberar

- pre\_order (impressão)

- in\_order (impressão)

- pos\_order (impressão)

Não há necessidade de solicitor valores para inclusão, remoção e consulta ao usuário (todos esses valores podem ser incluídos diretamente no código). Lembre-se que é um problema apenas para teste das funções solicitadas.

**Resposta:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

typedef struct arvore{

int valor;

struct arvore \*dir;

struct arvore \*esq;

}Arv;

Arv\* CriaA(){

return NULL;

}

bool VaziaA(Arv \*A){

if(A == NULL)

return true;

return false;

}

void InsereA(Arv \*\*A, int n){

/\*Arv \*novo = (Arv\*)malloc(sizeof(Arv));

novo->valor = n;

novo->dir = NULL;

novo->esq = NULL;

if(VaziaA(\*A)) //caso seja o primeiro elemento na arvore

(\*A) = novo;

else{

Arv \*pai = (\*A);

bool flag = true;

while(flag){

if(pai->valor < v){ //se o valor inserido for menor que o pai

if(pai->dir == NULL){ //caso não haja folha na direita

pai->dir = novo; //novo nó se situa na direita

flag = false;

}

else //atualiza o valor do pai para chegar em uma folha

pai = pai->dir;

}

else{ //senão o valor inserido é maior que o pai

if(pai->esq == NULL){ //caso não haja folha na esquerda

pai->esq = novo; //novo nó se situa na esquerda

flag = false;

}

else //atualiza o valor do pai para chegar em uma folha

pai = pai->esq;

}

}

}\*/

if(VaziaA(\*A)){

(\*A) = (Arv\*)malloc(sizeof(Arv));

if(VaziaA(\*A)) return;

(\*A)->valor = n;

(\*A)->esq = (\*A)->dir = NULL;

}

else{

if(n < (\*A)->valor)

InsereA(&(\*A)->esq,n);

else if(n > (\*A)->valor)

InsereA(&(\*A)->dir,n);

}

}

bool BuscaA(Arv \*A, int n){

/\*while(A != NULL){

if(A->valor < n) //direciona para a direita caso o número seja maior que o valor da partição da árvore

A = A->dir;

else if(A->valor > n) //direciona para a esquerda caso o número seja menor que o valor da partição da árvore

A = A->esq;

else //caso não seja nem menor e nem maior o número é igual

return true;

}

return false;\*/

if(!A) return false;

if(n == A->valor) return true;

if(n < A->valor)

return (BuscaA(A->esq,n));

return (BuscaA(A->dir,n));

}

Arv \*MaDir(Arv \*\*Aesq){

if((\*Aesq)->dir != NULL) //percorre a arvore ate chegar no maior valor da esquerda (ultimo valor da direita)

return MaDir(&(\*Aesq)->dir);

else{

Arv \*aux = (\*Aesq);

if((\*Aesq)->esq != NULL) //verifica se tem valor na esquerda

(\*Aesq) = (\*Aesq)->esq; //se tiver atualiza para o da esquerda

else

(\*Aesq) = NULL;

return aux;

}

}

void RemoveA(Arv \*\*A,int n){

if(\*A){

Arv \*aux;

if(n < (\*A)->valor){ //se o valor for menor ele se dirige a esquerda

aux = (\*A)->esq;

if((aux->dir != NULL) || (aux->esq != NULL))//verifica se o prox da esquerda possui filhos

RemoveA(&(\*A)->esq,n);

else{

if(aux->valor == n){//se nao possuir ele confere se o valor eh igual e desaloca

free(aux);

aux = NULL;

(\*A)->esq = NULL;

}

}

}

else if(n > (\*A)->valor){//se o valor for maior ele se dirige a direita

aux = (\*A)->dir;

if((aux->dir != NULL) || (aux->esq != NULL))//verifica se o prox da direita possui filhos

RemoveA(&(\*A)->dir,n);

else{

if(aux->valor == n){//se nao possuir ele confere se o valor eh igual e desaloca

free(aux);

aux = NULL;

(\*A)->dir = NULL;

}

}

}

else{//caso o no que sera desalocado possua filhos

aux = (\*A);

if((\*A)->dir == NULL){//se possuir filho na esquerda

(\*A) = (\*A)->esq;

free(aux);

}

if((\*A)->esq == NULL){//se possuir filho na direita

(\*A) = (\*A)->dir;

free(aux);

}

else{//possui filhos

aux = MaDir(&(\*A)->esq); //funçao que retorna o modulo que sera trocado (maior valor da esquerda com seu filho da direita se existir)

aux->esq = (\*A)->esq; //(\*A)->esq contem o maior nº da subarvore esquerda ou NULL

aux->dir = (\*A)->dir; //anexa o da direita fazendo o ramo da arvore "cair"

free(\*A);

(\*A) = aux;

}

}

}

}

void ImpressaoPre(Arv \*A){

printf("%d ",A->valor);

if(A->esq != NULL)

ImpressaoPre(A->esq);

if(A->dir != NULL)

ImpressaoPre(A->dir);

}

void ImpressaoIn(Arv \*A){

if(A->esq != NULL)

ImpressaoIn(A->esq);

printf("%d ",A->valor);

if(A->dir != NULL)

ImpressaoIn(A->dir);

}

void ImpressaoPos(Arv \*A){

if(A->esq != NULL)

ImpressaoPos(A->esq);

if(A->dir != NULL)

ImpressaoPos(A->dir);

printf("%d ",A->valor);

}

Arv\* LiberaA(Arv \*A){

if(A->dir != NULL)//verifica se há informações na direita

LiberaA(A->dir);

if(A->esq != NULL)//verifica se há informações na esquerda

LiberaA(A->esq);

free(A);//chega em uma folha e libera a informação

return NULL;

}

int main(){

Arv \*A = CriaA();

InsereA(&A,50);

InsereA(&A,25);

InsereA(&A,75);

InsereA(&A,10);

InsereA(&A,35);

InsereA(&A,5);

InsereA(&A,29);

printf("\nImpressao pre order:\n");

ImpressaoPre(A);

printf("\nImpressao in order:\n");

ImpressaoIn(A);

printf("\nImpressao pos order:\n");

ImpressaoPos(A);

printf("\n\nRemovendo os elemntos 50, 25 e 5:");

RemoveA(&A,25);

RemoveA(&A,50);

RemoveA(&A,5);

printf("\n\nImpressao pre order:\n");

ImpressaoPre(A);

printf("\nImpressao in order:\n");

ImpressaoIn(A);

printf("\nImpressao pos order:\n");

ImpressaoPos(A);

printf("\nBuscando pelo elemento 10:");

if(BuscaA(A, 10))

printf("\nElemento encontrado\n");

else

printf("\nElemento nao encontrado\n");

A = LiberaA(A);

return 0;

}