/\* Napisati program koji omogucava rad s binarnim stablom pretrazivanja. Treba omoguciti unosenje novog elementa u stablo, ispis elemenata, brisanje i pronalazenje nekog elementa. \*/

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct TreeNode;

typedef struct TreeNode \* Node;

Node MakeEmpty(Node);

Node Find(int, Node);

Node FindMin(Node);

Node FindMax(Node);

Node Insert(int, Node);

Node Delete(int, Node);

void Print(Node);

struct TreeNode

{

int Element;

Node Left;

Node Right;

};

void main()

{

Node root = NULL, temp;

int odabir = 0, X;

while (odabir != 9)

{

system("cls");

printf("\n\tOdaberi akciju:");

printf("\n\t\t 1 - unos novog elementa");

printf("\n\t\t 2 - ispis stabla");

printf("\n\t\t 3 - trazenje elementa");

printf("\n\t\t 4 - brisanje elementa");

printf("\n\t\t 5 - trazenje min. elementa");

printf("\n\t\t 6 - trazenje max. elementa");

printf("\n\t\t 7 - brisanje stabla");

printf("\n\t\t 9 - kraj\n\t");

scanf(" %d", &odabir);

switch(odabir)

{

case 1:

printf("\n\tUnesi element: ");

scanf(" %d", &X);

root = Insert(X, root);

break;

case 2:

system("cls");

Print(root);

break;

case 3:

printf("\n\tKoji element trazis? ");

scanf(" %d", &X);

temp = Find(X, root);

if (temp != NULL)

printf ("Element %d se nalazi na adresi %d", temp->Element, temp);

break;

case 4:

printf("\n\tKoji element zelis izbrisat? ");

scanf(" %d", &X);

root = Delete(X, root);

break;

case 5:

temp = FindMin(root);

printf("Minimalni element je %d", temp->Element);

break;

case 6:

temp = FindMax(root);

printf("Maximalni element je %d", temp->Element);

break;

case 7:

root = MakeEmpty(root);

case 9:

break;

default:

printf("\n\tKrivi unos!!!");

}

printf("\nPress any key to continue!");

getch();

}

}

Node Find(int X, Node T)

{

if (NULL == T)

{

printf("\n\tElement ne postoji!!");

return NULL;

}

else if(X < T->Element)

return Find(X, T->Left);

else if (X > T->Element)

return Find(X, T->Right);

return T;

}

Node FindMin(Node T)

{

while(T->Left != NULL)

T = T->Left;

return T;

}

Node FindMax(Node T)

{

while(T->Right != NULL)

T = T->Right;

return T;

}

Node Insert(int X, Node T)

{

if(NULL == T)

{

T = (Node)malloc(sizeof(struct TreeNode));

T->Left = NULL;

T->Right = NULL;

T->Element = X;

}

else if(X > T->Element)

T->Right = Insert(X, T->Right);

else if(X < T->Element)

T->Left = Insert(X, T->Left);

return T;

}

Node Delete(int X, Node T)

{

Node temp;

if(NULL == T)

printf("Nema ga!!!");

else if(X < T->Element)

T->Left = Delete(X, T->Left);

else if (X > T->Element)

T->Right = Delete(X, T->Right);

else

{

if(T->Left && T->Right)

{

temp = FindMin(T->Right);

T->Element = temp->Element;

T->Right = Delete(T->Element, T->Right);

}

else

{

temp = T;

if(NULL == T->Left)

{

T = T->Right;

}

else if(NULL == T->Right)

{

T = T->Left;

}

free(temp);

}

}

return T;

}

void Print(Node T)

{

static int n = 0;

int i;

if(-1 == n)

n = 0;

if(NULL != T)

{

n++;

Print(T->Right);

printf("\n");

for(i = 0; i < n; i++)

printf("\t");

printf(" %d", T->Element);

printf("\n");

n++;

Print(T->Left);

}

n--;

}

Node MakeEmpty(Node T)

{

if(NULL != T)

{

MakeEmpty(T->Left);

MakeEmpty(T->Right);

free(T);

}

return NULL;

}