//Drzewo BST

//Rafal Gawlik

//www.algorytm.org

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

//definicja wezla

struct wezel

{

int wartosc; //wartosc przechowywana w wezle

struct wezel \*rodzic; //wskaznik na rodzica

struct wezel \*l\_syn; //wskaznik na lewe dziecko

struct wezel \*p\_syn; //wskaznik na prawe dziecko

};

struct wezel \*root; //wskaźnik na root'a

//funkcja zwraca wskaznik elementu o najmniejszej wartosci (najbardziej na lewo)

struct wezel\* naj\_lewo(struct wezel \*start)

{

if(start->l\_syn != NULL)

return naj\_lewo(start->l\_syn);

else

return start;

}

//funkcja zwraca wezel o podanej wartosci, badz NULL, gdy taki wezel nie istnieje

struct wezel\* szukaj(struct wezel \*start, int wartosc)

{

//jezeli wezel ma szukana wartosc to odnalezlismy go

if (start->wartosc == wartosc) return start;

//jezeli szukana wartosc jest mniejsza to szukamy w lewym poddrzewie o ile istnieje

else if (wartosc < start->wartosc && start->l\_syn != NULL) return szukaj(start->l\_syn, wartosc);

//jezeli szukana wartosc jest wieksza to szukamy w prawym poddrzewie o ile istnieje

else if (wartosc > start->wartosc && start->p\_syn != NULL) return szukaj(start->p\_syn, wartosc);

return NULL;

}

//dodaje wezel o podanej wartosci n, do drzewa o korzeniu start

int dodawanie(int n, struct wezel \*start)

{

//jezeli drzewo jest puste to dodaj korzen

if (root == NULL)

{

root = (wezel\*)malloc(sizeof \*root);

root->wartosc = n;

root->l\_syn = NULL;

root->p\_syn = NULL;

root->rodzic = NULL;

}

//jezeli zadana wartosc jest mniejsza od korzenia idz do lewego poddrzewa

else if(n < start->wartosc)

{

//jezeli lewe poddrzewo istnieje wywolaj dla niego ta funkcje rekurencyjnie

if(start->l\_syn != NULL)

{

dodawanie(n,start->l\_syn);

}

//jezeli lewe poddrzewo nie istnieje dodaj nowy wezel o zadanej wartosci

else

{

wezel \*nowy = (wezel\*)malloc(sizeof \*root);

nowy->wartosc = n;

nowy->l\_syn = NULL;

nowy->p\_syn = NULL;

nowy->rodzic = start;

start->l\_syn=nowy;

}

}

//jezeli zadana wartosc jest wieksza lub rowna korzeniowi idz do prawego poddrzewa

else

{

//jezeli prawe poddrzewo istnieje wywolaj dla niego ta funkcje rekurencyjnie

if(start->p\_syn!=NULL)

{

dodawanie(n,start->p\_syn);

}

//jezeli prawe poddrzewo nie istnieje dodaj nowy wezel o zadanej wartosci

else

{

wezel \*nowy = (wezel\*)malloc(sizeof \*root);

nowy->wartosc = n;

nowy->l\_syn = NULL;

nowy->p\_syn = NULL;

nowy->rodzic = start;

start->p\_syn=nowy;

}

}

return 0;

}

//usun wezel start

void kasowanie(struct wezel \*start)

{

//jezeli wezel nie ma dzieci

if(start->l\_syn==NULL && start->p\_syn==NULL)

{

//jezeli wezel jest korzeniem

if(start->rodzic==NULL)

{

root=NULL;

}

//jezeli wezel jest po lewej stronie rodzica,

else if(start->rodzic->l\_syn==start)

{

//usun wezel z lewej strony wezla rodzica

start->rodzic->l\_syn=NULL;

}

else

{

//usun wezel z prawej strony wezla rodzica

start->rodzic->p\_syn=NULL;

}

delete start;

}

//jezeli wezel ma tylko jedno dziecko

else if(start->l\_syn==NULL || start->p\_syn==NULL)

{

//jezeli po lewej stronie nie ma dziecka

if(start->l\_syn==NULL)

{

//jezeli wezel jest korzeniem

if(start->rodzic==NULL)

{

root=start->p\_syn;

}

//jezeli wezel jest po lewej stronie rodzica

else if(start->rodzic->l\_syn==start)

{

//przyczep z lewej strony rodzica wezel bedacy po prawej stronie usuwanego wezla

start->rodzic->l\_syn=start->p\_syn;

}

else

{

//przyczep z prawej strony rodzica wezel bedacy po prawej stronie usuwanego wezla

start->rodzic->p\_syn=start->p\_syn;

}

}

else

{

//jezeli wezel jest korzeniem

if(start->rodzic==NULL)

{

root=start->l\_syn;

}

//jezeli wezel jest po lewej stronie rodzica

else if(start->rodzic->l\_syn==start)

{

//przyczep z lewej strony rodzica wezel bedacy po lewej stronie usuwanego wezla

start->rodzic->l\_syn=start->l\_syn;

}

else

{

//przyczep z prawej strony rodzica wezel bedacy po prawej stronie usuwanego wezla

start->rodzic->p\_syn=start->l\_syn;

}

}

delete start;

}

else

{

//wstaw w miejsce usuwanego elementu - najmniejsza wartosc z prawego poddrzewa

struct wezel \*temp;

temp=naj\_lewo(start->p\_syn);

start->wartosc = temp->wartosc;

kasowanie(temp);

}

}

//przejdz drzewo w kolejnosci zaczynajac od wezla start

void in\_order\_tree\_walk(struct wezel \*start)

{

if(start->l\_syn != NULL) //jezeli ma dzieci po lewej stronie wywolaj funkcje rekurencyjnie

in\_order\_tree\_walk(start->l\_syn);

printf("%d\n", start->wartosc); //wypisz wartosc

if(start->p\_syn != NULL) //jezeli ma dzieci po prawej stronie wywolaj rekurencyjnie

in\_order\_tree\_walk(start->p\_syn);

}

//lsouje wartosc w przedziale od a do b

int losowanie(int a, int b)

{

if(a < b)

return a + (int)((b-a+1.0)\*(rand()/(RAND\_MAX+1.0)));

else

{

fprintf(stderr, "złe wartości");

return -1;

}

}

//przklad uzycia drzewa BST

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

{

int i;

//pobierz rozmiar drzewa z parametru wejsciowego

int a,k,size=atoi(argv[1]);

root = NULL;

struct timezone tz;

struct timeval tv;

gettimeofday(&tv, &tz);

srand(tv.tv\_usec);

//losuj wartosc elementow

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

{

a=losowanie(1,100);

dodawanie(a, root);

}

printf("\n");

//przejdz drzewo w kolejnosci

in\_order\_tree\_walk(root);

//usun wartosc z drzewa

printf("Wartość węzła do usunięcia: \n");

scanf("%d", &k);

kasowanie(szukaj(root,k));

printf("\n\n");

//przejdz drzewo w kolejnosci

in\_order\_tree\_walk(root);

return 0;

}