#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

typedef struct tree

{

int num;

int high\_l;

int high\_r;

struct tree\* left;

struct tree\* right;

struct tree\* rev;

}tree;

void Update\_Upper\_Nodes(tree\* t1, tree\* t2)

{

if (t1->num < t2->num)

t1->high\_r = (t2->high\_l > t2->high\_r) ? t2->high\_l + 1 : t2->high\_r + 1;

else

t1->high\_l = (t2->high\_l > t2->high\_r) ? t2->high\_l + 1 : t2->high\_r + 1;

if (t1->rev)

Update\_Upper\_Nodes(t1->rev, t1);

}

void Tree\_height\_update\_lr(tree\* t1, tree\* t2, tree\* t3, tree\* a, tree\* b, tree\* c)

{

if (a)

t2->high\_l = (a->high\_l >= a->high\_r) ? a->high\_l + 1 : a->high\_r + 1;

else

t2->high\_l = 0;

if (b)

t2->high\_r = (b->high\_l >= b->high\_r) ? b->high\_l + 1 : b->high\_r + 1;

else

t2->high\_r = 0;

t3->high\_l = (t2->high\_l >= t2->high\_r) ? t2->high\_l + 1 : t2->high\_r + 1;

if (c)

t3->high\_r = (c->high\_l >= c->high\_r) ? c->high\_l + 1 : c->high\_r + 1;

else

t3->high\_r = 0;

t1->high\_l = (t3->high\_l >= t3->high\_r) ? t3->high\_l + 1 : t3->high\_r + 1;

}

void Tree\_height\_update\_l(tree\* t1, tree\* t2, tree\* t3, tree\* d, tree\* c)

{

t2->high\_l = (t3->high\_l >= t3->high\_r) ? t3->high\_l + 1 : t3->high\_r + 1;

if (c)

t1->high\_l = (c->high\_l >= c->high\_r) ? c->high\_l + 1 : c->high\_r + 1;

else

t1->high\_l = 0;

if (d)

t1->high\_r = (d->high\_l >= d->high\_r) ? d->high\_l + 1 : d->high\_r + 1;

else

t1->high\_r = 0;

t2->high\_r = (t1->high\_l >= t1->high\_r) ? t1->high\_l + 1 : t1->high\_r + 1;

if (t2->rev)

{

Update\_Upper\_Nodes(t2->rev, t2);

}

}

void l\_l\_rull(tree\* t1, tree\* t2, tree\* t3)

{

tree\* temp2;

tree\* temp;

if (t2->right)

{

temp2 = t2->right;

temp2->rev = t1;

t1->left = temp2;

}

else

t1->left = NULL;

temp = t1->rev;

t2->right = t1;

t2->rev = temp;

if (temp && temp->num > t1->num)

temp->left = t2;

else if (temp)

temp->right = t2;

t1->rev = t2;

Tree\_height\_update\_l(t1, t2, t3, t1->right, t1->left);

}

void l\_r\_rull(tree\* t1, tree\* t2, tree\* t3)

{

tree\* temp=t3->left;

t1->left = t3;

t2->right = temp;

if (temp)

temp->rev = t2;

t3->left = t2;

t3->rev = t1;

t2->rev = t3;

Tree\_height\_update\_lr(t1, t2, t3, t2->left, t2->right, t3->right);

l\_l\_rull(t1, t3, t2);

}

void left\_tree\_rull(tree\* t1, tree\*\* Replacing\_head)

{

tree\* t2, \* t3;

t2 = t1->left;

if (t2->high\_r > t2->high\_l)

{

t3 = t2->right;

if (t1->rev == NULL)

\*Replacing\_head = t3;

l\_r\_rull(t1, t2, t3);

}

else

{

t3 = t2->left;

if (t1->rev == NULL)

\*Replacing\_head = t2;

l\_l\_rull(t1, t2, t3);

}

}

void Tree\_height\_update\_r(tree\* t1, tree\* t2, tree\* t3, tree\* a, tree\* b)

{

tree\* temp = t2->rev;

if (a)

t1->high\_l = (a->high\_l >= a->high\_r) ? a->high\_l + 1 : a->high\_r + 1;

else

t1->high\_l = 0;

if (b)

t1->high\_r = (b->high\_l >= b->high\_r) ? b->high\_l + 1 : b->high\_r + 1;

else

t1->high\_r = 0;

t2->high\_r = (t3->high\_l >= t3->high\_r) ? t3->high\_l + 1 : t3->high\_r + 1;

t2->high\_l = (t1->high\_l >= t1->high\_r) ? t1->high\_l + 1 : t1->high\_r + 1;

if (t2->rev)

{

Update\_Upper\_Nodes(t2->rev, t2);

}

}

void Tree\_height\_update\_rl(tree\* t1, tree\* t2, tree\* t3, tree\* c, tree\* d, tree\* b)

{

if (c)

t2->high\_l = (c->high\_l >= c->high\_r) ? c->high\_l + 1 : c->high\_r + 1;

else

t2->high\_l = 0;

if (d)

t2->high\_r = (d->high\_l >= d->high\_r) ? d->high\_l + 1 : d->high\_r + 1;

else

t2->high\_r = 0;

t3->high\_r = (t2->high\_l >= t2->high\_r) ? t2->high\_l + 1 : t2->high\_r + 1;

if (b)

t3->high\_l = (b->high\_l >= b->high\_r) ? b->high\_l + 1 : b->high\_r + 1;

else

t3->high\_l = 0;

t1->high\_r = (t3->high\_l >= t3->high\_r) ? t3->high\_l + 1 : t3->high\_r + 1;

}

void r\_r\_rull(tree\* t1, tree\* t2, tree\* t3)

{

tree\* temp2;

tree\* temp;

if (t2->left)

{

temp2 = t2->left;

temp2->rev = t1;

t1->right = temp2;

}

else

t1->right = NULL;

temp = t1->rev;

t2->left = t1;

t2->rev = temp;

if (temp && temp->num > t1->num)

temp->left = t2;

else if (temp)

temp->right = t2;

t1->rev = t2;

Tree\_height\_update\_r(t1, t2, t3, t1->left, t1->right);

}

void r\_l\_rull(tree\* t1, tree\* t2, tree\* t3)

{

tree\* temp=t3->right;

t1->right = t3;

t2->left = temp;

if (temp)

temp->rev = t2;

t3->right = t2;

t3->rev = t1;

t2->rev = t3;

Tree\_height\_update\_rl(t1, t2, t3, t2->left, t2->right, t3->left);

r\_r\_rull(t1, t3, t2);

}

void right\_tree\_rull(tree\* t1, tree\*\* Replacing\_head)

{

tree\* t2, \* t3;

t2 = t1->right;

if (t2->high\_r < t2->high\_l)

{

t3 = t2->left;

if (t1->rev == NULL)

\*Replacing\_head = t3;

r\_l\_rull(t1, t2, t3);

}

else

{

t3 = t2->right;

if (t1->rev == NULL)

\*Replacing\_head = t2;

r\_r\_rull(t1, t2, t3);

}

}

void check\_tree(tree\* t, tree\*\* Replacing\_head)

{

if ((t) == NULL)

return;

else if ((t)->high\_l - (t)->high\_r > 1)

left\_tree\_rull(t, Replacing\_head);

else if ((t)->high\_l - (t)->high\_r < -1)

right\_tree\_rull(t, Replacing\_head);

else

check\_tree((t)->rev, Replacing\_head);

}

void new\_member(tree\*\* head, tree\*\* temp, int num)

{

\*head = (tree\*)malloc(sizeof(tree));

(\*head)->num = num;

(\*head)->left = NULL;

(\*head)->right = NULL;

(\*head)->rev = (\*temp);

(\*head)->high\_l = 0;

(\*head)->high\_r = 0;

}

void build\_tree(tree\*\* head, tree\*\* temp, int num)

{

int b, c;

static int a;

b = 0;

a = 0;

c = 0;

if (\*head == NULL)

new\_member(head, temp, num);

else

{

\*temp = \*head;

if ((\*head)->num > num)

{

b = 1;

build\_tree((&(\*head)->left), temp, num);

}

else if ((\*head)->num < num)

{

c = 1;

build\_tree((&(\*head)->right), temp, num);

}

else if (((\*head)->num == num))

return;

a++;

if (b)

(\*head)->high\_l = ((\*head)->high\_l > a) ? (\*head)->high\_l : a;

if (c)

(\*head)->high\_r = ((\*head)->high\_r > a) ? (\*head)->high\_r : a;

}

}

void remove\_leaf(tree\* t1, tree\* t2, tree\*\* Replacing\_head, int a)

{

if (t1 == NULL)

{

\*Replacing\_head = NULL;

free(t2);

}

else

{

int temp = t2->num;

if (t1->num > t2->num||a==1)

{

t1->high\_l -= 1;

t1->left = NULL;

}

else

{

t1->high\_r -= 1;

t1->right = NULL;

}

if (t1->rev)

Update\_Upper\_Nodes(t1->rev, t1);

free(t2);

check\_tree(t1, Replacing\_head);

}

}

void remove\_node\_one\_son(tree\* t1, tree\* t2, tree\*\* Replacing\_head)

{

int a;

if (t1->num > t2->num)

a = 1;

else

a = 2;

t1->num = t2->num;

remove\_leaf(t1, t2, Replacing\_head,a);

}

void remove\_full\_node(tree\* t1,tree\* t2, tree\*\* Replacing\_head, int a)

{

if (t2->right)

remove\_full\_node(t1,t2->right, Replacing\_head,0);

else

{

t1->num = t2->num;

if (t2->left)

remove\_node\_one\_son(t2, t2->left, Replacing\_head);

else

remove\_leaf(t2->rev, t2, Replacing\_head,a);

}

}

void remove\_member(tree\* t, tree\*\* Replacing\_head)

{

if (t->left == NULL && t->right == NULL)

remove\_leaf(t->rev, t, Replacing\_head,0);

else if (t->left == NULL && (t->right))

remove\_node\_one\_son(t, t->right, Replacing\_head);

else if (t->left && (t->right == NULL))

remove\_node\_one\_son(t, t->left, Replacing\_head);

else

remove\_full\_node(t, t->left, Replacing\_head,1);

}

void finding\_member(tree\* head, tree\*\* Replacing\_head, int num, int a)

{

if (head==NULL)

printf("The number is not in the list");

else

{

if ((head)->num < num)

finding\_member(head->right, Replacing\_head, num, a);

else if ((head)->num > num)

finding\_member(head->left, Replacing\_head, num, a);

else if (a == 1 )

remove\_member(head, Replacing\_head);

else if (a == 0)

check\_tree(head, Replacing\_head);

else

printf("The number is in the list");

}

}

void display\_tree(tree\* head)

{

if (head)

{

display\_tree(head->left);

printf("/n%p %p %p %p %d %d %d\n", head, head->left, head->right, head->rev, head->num, head->high\_l, head->high\_r);

display\_tree(head->right);

}

else return;

}

char choise()

{

char c;

printf("\n\n\r(a) - add to list\n\r");

printf("(r) - remove from list\n\r");

printf("(f) - Finding a number\n\r");

printf("(d) - display tha list\n\r");

printf("(e) - exit\n\r");

scanf(" %c", &c);

return c;

}

void del\_t( tree\*\* t)

{

if (\*t != NULL)

{

del\_t(&(\*t)->left);

del\_t(&(\*t)->right);

printf("%p\t", \*t);

free(\*t);

}

else

return;

}

void main()

{

char c;

int i, number, b;

tree\* head = NULL;

tree\* temp = NULL;

tree\* Replacing\_head = NULL;

printf("choose char to: \n\r");

c = choise();

while (c != 'e')

{

if (c == 'a')

{

printf("enter a number");

scanf("%d", &number);

if (head == NULL)

temp = NULL;

build\_tree(&head, &temp, number);

finding\_member(head, &Replacing\_head, number, 0);

if (Replacing\_head)

head = Replacing\_head;

}

else if (c == 'r')

{

Replacing\_head=head;

printf("enter a number");

scanf("%d", &number);

finding\_member(head, &Replacing\_head, number, 1);

head = Replacing\_head;

}

else if (c == 'f')

{

printf("enter a number");

scanf("%d", &number);

finding\_member(head, &Replacing\_head, number, 2);

}

else if (c == 'd')

display\_tree(head);

else printf("Error, insert an appropriate char");

c = choise();

}

del\_t(&head);

}