**CODE:-**

#include <stdio.h>

#include <stdlib.h>

struct node

{

struct node \*lchild;

int info;

struct node \*rchild;

int lthread;

int rthread;

};

struct node \*insert(struct node \*root, int ikey)

{

struct node \*tmp, \*par, \*ptr;

int found = 0;

ptr = root;

par = NULL;

while (ptr != NULL)

{

if (ikey == ptr->info)

{

found = 1;

break;

}

par = ptr;

if (ikey < ptr->info)

{

if (ptr->lthread == 0)

ptr = ptr->lchild;

else

break;

}

else

{

if (ptr->rthread == 0)

ptr = ptr->rchild;

else

break;

}

}

if (found)

printf("Duplicate key.\n");

else

{

tmp = (struct node \*)malloc(sizeof(struct node));

tmp->info = ikey;

tmp->lthread = 1;

tmp->rthread = 1;

if (par == NULL)

{

root = tmp;

tmp->lchild = NULL;

tmp->rchild = NULL;

}

else if (ikey < par->info)

{

tmp->lchild = par->lchild;

tmp->rchild = par;

par->lthread = 0;

par->lchild = tmp;

}

else if (ikey > par->info)

{

tmp->rchild = par->rchild;

tmp->lchild = par;

par->rthread = 0;

par->rchild = tmp;

}

}

return root;

}

struct node \*insucc(struct node \*root)

{

struct node \*ptr = root;

if (root->rthread == 1)

return root->rchild;

else

{

ptr = root->rchild;

while (ptr->lthread != 1)

ptr = ptr->lchild;

return ptr;

}

}

struct node \*inpred(struct node \*root)

{

struct node \*ptr = root;

if (root->lthread == 1)

return root->lchild;

else

{

ptr = root->lchild;

while (ptr->rthread != 1)

ptr = ptr->rchild;

return ptr;

}

}

struct node \*case\_a(struct node \*root, struct node \*par, struct node \*ptr)

{

if (ptr == NULL)

root = NULL;

else if (par->lchild == ptr)

{

par->lthread = 1;

par->lchild = ptr->lchild;

}

else

{

par->rthread = 1;

par->rchild = ptr->rchild;

}

free(ptr);

return root;

}

struct node \*case\_b(struct node \*root, struct node \*par, struct node \*ptr)

{

struct node \*child, \*p, \*s;

if (ptr->lthread == 0)

child = ptr->lchild;

else

child = ptr->rchild;

if (par == NULL)

root = child;

else if (par->lchild == ptr)

par->lchild = child;

else

par->rchild = child;

s = insucc(ptr);

p = inpred(ptr);

if (ptr->lthread == 0)

p->rchild = s;

else if (ptr->rthread == 0)

s->lchild = p;

free(ptr);

return root;

}

struct node \*case\_c(struct node \*root, struct node \*par, struct node \*ptr)

{

struct node \*succ, \*parsucc;

parsucc = ptr;

succ = ptr->rchild;

while (succ->lchild != NULL)

{

parsucc = succ;

succ = succ->lchild;

}

ptr->info = succ->info;

if (succ->lthread == 1 && succ->rthread == 1)

root = case\_a(root, parsucc, succ);

else

root = case\_b(root, parsucc, succ);

return root;

}

struct node \*del(struct node \*root, int dkey)

{

struct node \*par, \*ptr;

int found = 0;

ptr = root;

par = NULL;

while (ptr != NULL)

{

if (dkey == ptr->info)

{

found = 1;

break;

}

par = ptr;

if (dkey < ptr->info)

{

if (ptr->lthread == 0)

ptr = ptr->lchild;

else

break;

}

else

{

if (ptr->rthread == 0)

ptr = ptr->rchild;

else

break;

}

}

if (found == 0)

printf("%d not present in the tree.\n", dkey);

else if (ptr->lthread == 0 && ptr->rthread == 0)

root = case\_c(root, par, ptr);

else if (ptr->lthread == 0)

root = case\_b(root, par, ptr);

else if (ptr->rthread == 0)

root = case\_b(root, par, ptr);

else

root = case\_a(root, par, ptr);

return root;

}

void search(struct node \*ptr, int skey)

{

struct node \*p;

int found = 0;

p = ptr;

while (p != NULL)

{

if (p->info == skey)

{

found = 1;

break;

}

else if (skey < p->info)

{

if (p->lthread == 0)

p = p->lchild;

else

break;

}

else

{

if (p->rthread == 0)

p = p->rchild;

else

break;

}

}

if (found)

printf("Key found in tree.\n");

else

printf("Key not found in tree.\n");

}

void inorder(struct node \*root)

{

struct node \*ptr;

if (root == NULL)

{

printf("Tree does not exist.\n");

return;

}

ptr = root;

while (ptr->lthread == 0)

ptr = ptr->lchild;

while (ptr != NULL)

{

printf("%d ", ptr->info);

ptr = insucc(ptr);

}

}

int main()

{

int A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};

int ch, elem;

struct node \*root = NULL;

for (int i = 0; i < 15; i++)

root = insert(root, A[i]);

while (1)

{

printf("Enter 1 to insert element.\n");

printf("Enter 2 to delete element.\n");

printf("Enter 3 to search element.\n");

printf("Enter 4 to perform inorder traversal.\n");

printf("Enter 5 to exit.\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter the element to be inserted.\n");

scanf("%d", &elem);

root = insert(root, elem);

break;

case 2:

printf("Enter the element to be deleted.\n");

scanf("%d", &elem);

root = del(root, elem);

break;

case 3:

printf("Enter the element to be searched.\n");

scanf("%d", &elem);

search(root, elem);

break;

case 4:

printf("The inorder traversal of the tree is: ");

inorder(root);

printf("\n");

break;

case 5:

exit(1);

default:

printf("Erroneous input.\n");

}

}

return 0;

}

**OUTPUT:-**

