ARRAY OPERATIONS-INSERTION

#include <stdio.h>

#include <stdlib.h>

#define FALSE 0

#define TRUE 1

struct node

{

struct node \*lchild;

int info;

struct node \*rchild;

int balance;

};

struct node \*RotateLeft(struct node \*pptr);

struct node \*RotateRight(struct node \*pptr);

struct node \*insert(struct node \*pptr, int ikey);

struct node \*insert\_left\_check(struct node \*pptr, int \*ptaller);

struct node \*insert\_right\_check(struct node \*pptr, int \*ptaller);

struct node \*insert\_LeftBalance(struct node \*pptr);

struct node \*insert\_RightBalance(struct node \*pptr);

struct node \*del(struct node \*pptr, int dkey);

struct node \*del\_left\_check(struct node \*pptr, int \*pshorter);

struct node \*del\_right\_check(struct node \*pptr, int \*pshorter);

struct node \*del\_LeftBalance(struct node \*pptr,int \*pshorter);

struct node \*del\_RightBalance(struct node \*pptr,int \*pshorter);

void display(struct node \*ptr,int level);

int main()

{

int choice,key;

struct node \*root = NULL;

while(1)

{

printf("\n");

printf("1.Insert\n");

printf("2.Display\n");

printf("3.Delete\n");

printf("4.Quit\n");

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("\nEnter the key to be inserted : ");

scanf("%d",&key);

root = insert(root,key);

break;

case 2:

printf("\n");

display(root,0);

printf("\n");

break;

case 3:

printf("\nEnter the key to be deleted : ");

scanf("%d",&key);

root = del(root,key);

break;

case 4:

exit(1);

default:

printf("\nWrong choice\n");

}

}

return 0;

}

void display(struct node \*ptr,int level)

{

int i;

if(ptr == NULL )

return;

else

{

display(ptr->rchild, level+1);

printf("\n");

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

printf(" ");

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

display(ptr->lchild, level+1);

}

}

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

{

static int taller;

if(pptr==NULL)

{

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

pptr->info = ikey;

pptr->lchild = NULL;

pptr->rchild = NULL;

pptr->balance = 0;

taller = TRUE;

}

else if(ikey < pptr->info)

{

pptr->lchild = insert(pptr->lchild, ikey);

if(taller==TRUE)

pptr = insert\_left\_check( pptr, &taller );

}

else if(ikey > pptr->info)

{

pptr->rchild = insert(pptr->rchild, ikey);

if(taller==TRUE)

pptr = insert\_right\_check(pptr, &taller);

}

else

{

printf("Duplicate key\n");

taller = FALSE;

}

return pptr;

}

struct node \*insert\_left\_check(struct node \*pptr, int \*ptaller )

{

switch(pptr->balance)

{

case 0:

pptr->balance = 1;

break;

case -1:

pptr->balance = 0;

\*ptaller = FALSE;

break;

case 1:

pptr = insert\_LeftBalance(pptr);

\*ptaller = FALSE;

}

return pptr;

}

struct node \*insert\_right\_check(struct node \*pptr, int \*ptaller )

{

switch(pptr->balance)

{

case 0:

pptr->balance = -1;

break;

case 1:

pptr->balance = 0;

\*ptaller = FALSE;

break;

case -1:

pptr = insert\_RightBalance(pptr);

\*ptaller = FALSE;

}

return pptr;

}

struct node \*insert\_LeftBalance(struct node \*pptr)

{

struct node \*aptr, \*bptr;

aptr = pptr->lchild;

if(aptr->balance == 1)

{

pptr->balance = 0;

aptr->balance = 0;

pptr = RotateRight(pptr);

}

else

{

bptr = aptr->rchild;

switch(bptr->balance)

{

case -1:

pptr->balance = 0;

aptr->balance = 1;

break;

case 1:

pptr->balance = -1;

aptr->balance = 0;

break;

case 0:

pptr->balance = 0;

aptr->balance = 0;

}

bptr->balance = 0;

pptr->lchild = RotateLeft(aptr);

pptr = RotateRight(pptr);

}

return pptr;

}

struct node \*insert\_RightBalance(struct node \*pptr)

{

struct node \*aptr, \*bptr;

aptr = pptr->rchild;

if(aptr->balance == -1)

{

pptr->balance = 0;

aptr->balance = 0;

pptr = RotateLeft(pptr);

}

else

{

bptr = aptr->lchild;

switch(bptr->balance)

{

case -1:

pptr->balance = 1;

aptr->balance = 0;

break;

case 1:

pptr->balance = 0;

aptr->balance = -1;

break;

case 0:

pptr->balance = 0;

aptr->balance = 0;

}

bptr->balance = 0;

pptr->rchild = RotateRight(aptr);

pptr = RotateLeft(pptr);

}

return pptr;

}

struct node \*RotateLeft(struct node \*pptr)

{

struct node \*aptr;

aptr = pptr->rchild;

pptr->rchild = aptr->lchild;

aptr->lchild = pptr;

return aptr;

}

struct node \*RotateRight(struct node \*pptr)

{

struct node \*aptr;

aptr = pptr->lchild;

pptr->lchild = aptr->rchild;

aptr->rchild = pptr;

return aptr;

}

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

{

struct node \*tmp, \*succ;

static int shorter;

if( pptr == NULL)

{

printf("Key not present \n");

shorter = FALSE;

return(pptr);

}

if( dkey < pptr->info )

{

pptr->lchild = del(pptr->lchild, dkey);

if(shorter == TRUE)

pptr = del\_left\_check(pptr, &shorter);

}

else if( dkey > pptr->info )

{

pptr->rchild = del(pptr->rchild, dkey);

if(shorter==TRUE)

pptr = del\_right\_check(pptr, &shorter);

}

else

{

if( pptr->lchild!=NULL && pptr->rchild!=NULL )

{

succ = pptr->rchild;

while(succ->lchild)

succ = succ->lchild;

pptr->info = succ->info;

pptr->rchild = del(pptr->rchild, succ->info);

if( shorter == TRUE )

pptr = del\_right\_check(pptr, &shorter);

}

else

{

tmp = pptr;

if( pptr->lchild != NULL )

pptr = pptr->lchild;

else if( pptr->rchild != NULL)

pptr = pptr->rchild;

else

pptr = NULL;

free(tmp);

shorter = TRUE;

}

}

return pptr;

}

struct node \*del\_left\_check(struct node \*pptr, int \*pshorter)

{

switch(pptr->balance)

{

case 0:

pptr->balance = -1;

\*pshorter = FALSE;

break;

case 1:

pptr->balance = 0;

break;

case -1:

pptr = del\_RightBalance(pptr, pshorter);

}

return pptr;

}

struct node \*del\_right\_check(struct node \*pptr, int \*pshorter)

{

switch(pptr->balance)

{

case 0:

pptr->balance = 1;

\*pshorter = FALSE;

break;

case -1:

pptr->balance = 0;

break;

case 1:

pptr = del\_LeftBalance(pptr, pshorter );

}

return pptr;

}

struct node \*del\_LeftBalance(struct node \*pptr,int \*pshorter)

{

struct node \*aptr, \*bptr;

aptr = pptr->lchild;

if( aptr->balance == 0)

{

pptr->balance = 1;

aptr->balance = -1;

\*pshorter = FALSE;

pptr = RotateRight(pptr);

}

else if(aptr->balance == 1 )

{

pptr->balance = 0;

aptr->balance = 0;

pptr = RotateRight(pptr);

}

else

{

bptr = aptr->rchild;

switch(bptr->balance)

{

case 0:

pptr->balance = 0;

aptr->balance = 0;

break;

case 1:

pptr->balance = -1;

aptr->balance = 0;

break;

case -1:

pptr->balance = 0;

aptr->balance = 1;

}

bptr->balance = 0;

pptr->lchild = RotateLeft(aptr);

pptr = RotateRight(pptr);

}

return pptr;

}

struct node \*del\_RightBalance(struct node \*pptr,int \*pshorter)

{

struct node \*aptr, \*bptr;

aptr = pptr->rchild;

if (aptr->balance == 0)

{

pptr->balance = -1;

aptr->balance = 1;

\*pshorter = FALSE;

pptr = RotateLeft(pptr);

}

else if(aptr->balance == -1 )

{

pptr->balance = 0;

aptr->balance = 0;

pptr = RotateLeft(pptr);

}

else

{

bptr = aptr->lchild;

switch(bptr->balance)

{

case 0:

pptr->balance = 0;

aptr->balance = 0;

break;

case 1:

pptr->balance = 0;

aptr->balance = -1;

break;

case -1:

pptr->balance = 1;

aptr->balance = 0;

}

bptr->balance = 0;

pptr->rchild = RotateRight(aptr);

pptr = RotateLeft(pptr);

}

return pptr;

}

Output:

