**Lab Program 4:**

**Insertion and Deletion operation- AVL Trees**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int key;

struct Node \*lft;

struct Node \*rgt;

int height;

};

int max(int a, int b);

int height(struct Node \*N)

{

if (N == NULL)

return 0;

return N->height;

}

int max(int a, int b)

{

return (a > b)? a : b;

}

struct Node\* newNode(int key)

{

struct Node\* node = (struct Node\*)

malloc(sizeof(struct Node));

node->key = key;

node->lft = NULL;

node->rgt = NULL;

node->height = 1;

return(node);

}

struct Node \*rgtRotate(struct Node \*y)

{

struct Node \*x = y->lft;

struct Node \*T2 = x->rgt;

x->rgt = y;

y->lft = T2;

y->height = max(height(y->lft), height(y->rgt))+1;

x->height = max(height(x->lft), height(x->rgt))+1;

return x;

}

struct Node \*lftRotate(struct Node \*x)

{

struct Node \*y = x->rgt;

struct Node \*T2 = y->lft;

y->lft = x;

x->rgt = T2;

x->height = max(height(x->lft), height(x->rgt))+1;

y->height = max(height(y->lft), height(y->rgt))+1;

return y;

}

intgetBalance(struct Node \*N)

{

if (N == NULL)

return 0;

return height(N->lft) - height(N->rgt);

}

struct Node\* insert(struct Node\* node, int key)

{

if (node == NULL)

return(newNode(key));

if (key < node->key)

node->lft = insert(node->lft, key);

else if (key > node->key)

node->rgt = insert(node->rgt, key);

else

return node;

node->height = 1 + max(height(node->lft),

height(node->rgt));

int balance = getBalance(node);

if (balance > 1 && key < node->lft->key)

return rgtRotate(node);

if (balance < -1 && key > node->rgt->key)

return lftRotate(node);

if (balance > 1 && key > node->lft->key)

{

node->lft = lftRotate(node->lft);

return rgtRotate(node);

}

if (balance < -1 && key < node->rgt->key)

{

node->rgt = rgtRotate(node->rgt);

return lftRotate(node);

}

return node;

}

struct Node \* minValueNode(struct Node\* node)

{

struct Node\* current = node;

/\* loop down to find the lftmost leaf \*/

while (current->lft != NULL)

current = current->lft;

return current;

}

struct Node\* deleteNode(struct Node\* base, int key)

{

if (base == NULL)

return base;

if ( key < base->key )

base->lft = deleteNode(base->lft, key);

else if( key > base->key )

base->rgt = deleteNode(base->rgt, key);

else

{

if( (base->lft == NULL) || (base->rgt == NULL) )

{

struct Node \*temp = base->lft ? base->lft :

base->rgt;

if (temp == NULL)

{

temp = base;

base = NULL;

}

else

\*base = \*temp;

free(temp);

}

else

{

struct Node\* temp = minValueNode(base->rgt);

base->key = temp->key;

base->rgt = deleteNode(base->rgt, temp->key);

}

}

if (base == NULL)

return base;

base->height = 1 + max(height(base->lft),

height(base->rgt));

int balance = getBalance(base);

if (balance > 1 &&getBalance(base->lft) >= 0)

return rgtRotate(base);

if (balance > 1 &&getBalance(base->lft) < 0)

{

base->lft = lftRotate(base->lft);

return rgtRotate(base);

}

if (balance < -1 &&getBalance(base->rgt) <= 0)

return lftRotate(base);

if (balance < -1 &&getBalance(base->rgt) > 0)

{

base->rgt = rgtRotate(base->rgt);

return lftRotate(base);

}

return base;

}

void preOrder(struct Node \*base)

{

if(base != NULL)

{

printf("%d ", base->key);

preOrder(base->lft);

preOrder(base->rgt);

}

}

int main()

{

struct Node \*base = NULL;

base = insert(base, 27);

base = insert(base, 9);

base = insert(base, 19);

base = insert(base, 91);

base = insert(base, 90);

base = insert(base, 72);

printf("The output of an AVL tree in preOrder form: \n");

preOrder(base);

printf("\n");

base = deleteNode(base, 9);

base = deleteNode(base, 72);

printf("The tree in preOrder traversal outputs after deletion: \n");

preOrder(base);

printf("\n");

base = insert(base, 1);

base = insert(base, 2709);

printf("The tree in preOrder traversal outputs 2 more inserts: \n");

preOrder(base);

printf("\n");

return 0;

}

OUTPUT

