**Exercise 8: Dictionary Application using AVL Tree**

**V1:**

**ADT.h:**

#include<stdio.h>

#include<stdlib.h>

struct AVL

{

int key;

struct AVL \*left;

struct AVL \*right;

int ht;

};

int max(int a, int b);

int height(struct AVL \*N);

struct AVL \*rightRotate(struct AVL \*y);

struct AVL \*leftRotate(struct AVL \*x);

int getBalance(struct AVL \*N);

struct AVL\* insert(struct AVL\*, int key);

**impl.h:**

#include "adt.h"

#include<malloc.h>

#define COUNT 10

int height(struct AVL\* N)

{

if (N == NULL)

return -1;

return N->ht;

}

int max(int a, int b)

{

return (a > b)? a : b;

}

/\*struct AVL\* newNode(int key)

{

struct AVL\* node = (struct Node\*)malloc(sizeof(struct Node));

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 0;

return(node);

}\*/

struct AVL \*rightRotate(struct AVL \*y)

{

struct AVL \*x = y->left;

struct AVL \*T2 = x->right;

x->right = y;

y->left = T2;

y->ht = max(height(y->left), height(y->right))+1;

x->ht = max(height(x->left), height(x->right))+1;

return x;

}

struct AVL \*leftRotate(struct AVL \*x)

{

struct AVL \*y = x->right;

struct AVL \*T2 = y->left;

y->left = x;

x->right = T2;

x->ht = max(height(x->left), height(x->right))+1;

y->ht = max(height(y->left), height(y->right))+1;

return y;

}

int getBalance(struct AVL \*N)

{

if (N == NULL)

return 0;

return height(N->left) - height(N->right);

}

struct AVL\* insert(struct AVL\* T, int key)

{

if (T== NULL)

{

T= (struct AVL\*)malloc(sizeof(struct AVL));

T->key = key;

T->left = NULL;

T->right = NULL;

T->ht = 0;

return(T);

}

if (key < T->key)

T->left = insert(T->left, key);

else if (key > T->key)

T->right = insert(T->right, key);

else

return T;

T->ht = 1+ max(height(T->left),height(T->right));

int bal = getBalance(T);

// Left Left Case

if (bal > 1 && key < T->left->key)

return rightRotate(T);

// Right Right Case

if (bal < -1 && key > T->right->key)

return leftRotate(T);

// Left Right Case

if (bal > 1 && key > T->left->key)

{

T->left = leftRotate(T->left);

return rightRotate(T);

}

// Right Left Case

if (bal < -1 && key < T->right->key)

{

T->right = rightRotate(T->right);

return leftRotate(T);

}

return T;

}

/\*void preOrder(struct AVL \*root)

{

if(root != NULL)

{

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

preOrder(root->left);

preOrder(root->right);

}

}\*/

void inorder(struct AVL \*T)

{

if(T)

{

inorder(T->left);

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

inorder(T->right);

}

}

void printCurLvl(struct AVL\* T, int lvl)

{

if(T==NULL)

return;

if(lvl==1)

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

else if(lvl>1) {

printCurLvl(T->left,lvl-1);

//printf("\n");

printCurLvl(T->right,lvl-1);

//printf("\n");

}

}

void printLvlOrd(struct AVL\* T)

{

int h=height(T)+1;

for(int i=1;i<=h;i++)

{

printCurLvl(T,i);

printf("\n");

}

printf("\n");

}

void print2DUtil(struct AVL \*root, int space)

{ if (root == NULL)

return;

space += COUNT;

print2DUtil(root->right, space);

printf("\n");

for (int i = COUNT; i < space; i++)

printf(" ");

printf("%d\n", root->key);

print2DUtil(root->left, space);

}

void print2D(struct AVL \*root)

{

print2DUtil(root, 0);

}

**appl.c:**

#include "impl.h"

int main()

{

struct AVL \*T = NULL;

printf("\nEnter the number of elements: ");

int n,ele;

scanf("%d",&n);

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

{

printf("Enter the element: ");

scanf("%d",&ele);

T=insert(T,ele);

}

/\*T = insert(T, 23);

T = insert(T, 12);

T = insert(T, 13);

T = insert(T, 4);

T = insert(T, 45);

T = insert(T, 54);\*/

//printf("%d",height(T));

printf("\nInorder traversal of the constructed AVL"

" tree is \n");

//preOrder(T);

//printf("\n");

inorder(T);

printf("\n\nLevel Order Traversal:\n");

printLvlOrd(T);

printf("AVL TREE:\n");

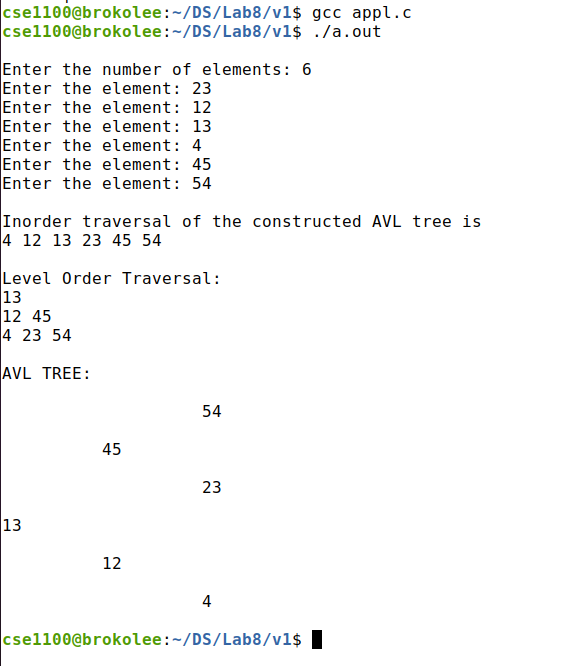
print2D(T);

printf("\n");

return 0;

}

**Sample I/O:**

****

**V2:**

**ADT.h:**

#include<stdio.h>

#include<stdlib.h>

struct wM

{

char word[30];

char mean[30];

};

struct AVL

{

struct wM dict;

struct AVL \*left;

struct AVL \*right;

int ht;

};

int max(int a, int b);

int height(struct AVL \*N);

struct AVL \*rightRotate(struct AVL \*y);

struct AVL \*leftRotate(struct AVL \*x);

int getBalance(struct AVL \*N);

struct AVL\* insert(struct AVL\*, struct wM);

**impl.h:**

#include "adt.h"

//#include<malloc.h>

#include<string.h>

#define COUNT 10

int height(struct AVL\* N)

{

if (N == NULL)

return -1;

return N->ht;

}

int max(int a, int b)

{

return (a > b)? a : b;

}

/\*struct AVL\* newNode(int key)

{

struct AVL\* node = (struct Node\*)malloc(sizeof(struct Node));

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 0;

return(node);

}\*/

struct AVL \*rightRotate(struct AVL \*y)

{

struct AVL \*x = y->left;

struct AVL \*T2 = x->right;

x->right = y;

y->left = T2;

y->ht = max(height(y->left), height(y->right))+1;

x->ht = max(height(x->left), height(x->right))+1;

return x;

}

struct AVL \*leftRotate(struct AVL \*x)

{

struct AVL \*y = x->right;

struct AVL \*T2 = y->left;

y->left = x;

x->right = T2;

x->ht = max(height(x->left), height(x->right))+1;

y->ht = max(height(y->left), height(y->right))+1;

return y;

}

int getBalance(struct AVL \*N)

{

if (N == NULL)

return 0;

return height(N->left) - height(N->right);

}

struct AVL\* insert(struct AVL\* T, struct wM W)

{

if (T== NULL)

{

T= (struct AVL\*)malloc(sizeof(struct AVL));

T->dict = W;

T->left = NULL;

T->right = NULL;

T->ht = 0;

return(T);

}

if ( strcmp(W.word,T->dict.word)<0)

T->left = insert(T->left, W);

else if (strcmp(W.word,T->dict.word)>0)

T->right = insert(T->right, W);

else

return T;

T->ht = 1+ max(height(T->left),height(T->right));

int bal = getBalance(T);

// Left Left Case

if (bal > 1 && strcmp(W.word,T->dict.word)<0)

return rightRotate(T);

// Right Right Case

if (bal < -1 && strcmp(W.word,T->dict.word)>0)

return leftRotate(T);

// Left Right Case

if (bal > 1 && strcmp(W.word,T->dict.word)>0)

{

T->left = leftRotate(T->left);

return rightRotate(T);

}

// Right Left Case

if (bal < -1 && strcmp(W.word,T->dict.word)<0)

{

T->right = rightRotate(T->right);

return leftRotate(T);

}

return T;

}

struct AVL\* find(struct AVL \*T,char s[])

{

if(!T||strcmp(T->dict.word,s)==0)

return T;

else if(strcmp(T->dict.word,s)> 0)

return find(T->left,s);

return find(T->right,s);

}

void preOrder(struct AVL \*root)

{

if(root != NULL)

{

printf("%s: %s\n",root->dict.word,root->dict.mean);

preOrder(root->left);

preOrder(root->right);

}

}

void inorder(struct AVL \*T)

{

if(T)

{

inorder(T->left);

printf("%s: %s\n",T->dict.word,T->dict.mean);

inorder(T->right);

}

}

void printCurLvl(struct AVL\* T, int lvl)

{

if(T==NULL)

return;

if(lvl==1)

printf("%s: %s\n",T->dict.word,T->dict.mean);

else if(lvl>1) {

printCurLvl(T->left,lvl-1);

//printf("\n");

printCurLvl(T->right,lvl-1);

//printf("\n");

}

}

void printLvlOrd(struct AVL\* T)

{

int h=height(T)+1;

for(int i=1;i<=h;i++)

{

printCurLvl(T,i);

printf("\n");

}

printf("\n");

}

void print2DUtil(struct AVL \*root, int space)

{

if (root == NULL)

return;

space += COUNT;

print2DUtil(root->right, space);

printf("\n");

for (int i = COUNT; i < space; i++)

printf(" ");

printf("%s: %s\n",root->dict.word,root->dict.mean);

print2DUtil(root->left, space);

}

void print2D(struct AVL \*root)

{

print2DUtil(root, 0);

}

**appl.c:**

#include "impl.h"

#include<ctype.h>

int main()

{

struct AVL \*T = NULL;

struct wM dict[10];

printf("\nEnter the number of word-meaning pairs: ");

int n;

char s1[20],s2[20];

scanf("%d",&n);

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

{

printf("Enter the word: ");

scanf("%s",s1);

printf("Enter the meaning: ");

scanf(" %[^\n]%\*c", s2);

strcpy(dict[i].word,s1);

strcpy(dict[i].mean,s2);

T=insert(T,dict[i]);

}

/\*strcpy(dict[0].word,"bisk");

strcpy(dict[0].mean,"soup");

strcpy(dict[1].word,"cite");

strcpy(dict[1].mean,"refer");

strcpy(dict[2].word,"boom");

strcpy(dict[2].mean,"sound");

strcpy(dict[3].word,"able");

strcpy(dict[3].mean,"opportunity");

strcpy(dict[4].word,"aged");

strcpy(dict[4].mean,"old");

strcpy(dict[5].word,"crew");

strcpy(dict[5].mean,"group of poeple");\*/

/\*T = insert(T, 23);

T = insert(T, 12);

T = insert(T, 13);

T = insert(T, 4);

T = insert(T, 45);

T = insert(T, 54);\*/

/\*for(int i=0;i<n;i++)

{

T=insert(T,dict[i]);

}\*/

//printf("%d",height(T));

printf("\nInorder traversal of the constructed AVL"

" tree is \n");

//preOrder(T);

//printf("\n");

inorder(T);

/\*printf("\n\nLevel Order Traversal:\n");

printLvlOrd(T);

printf("AVL TREE:\n");

print2D(T);

printf("\n");\*/

char word[30];

printf("\nEnter word to search for meaning: ");

scanf("%s",word);

struct AVL \*temp=find(T,word);

if(temp==NULL)

printf("\n%s - Word is not in our dict!\n",word);

else

printf("%s: %s\n",temp->dict.word,temp->dict.mean);

return 0;

}

**Sample I/O:**

