**Title :** A dictionary stores keywords and its meanings .Provide facility for adding

new keywords, deleting keywords ,updating values of entry Provide facility to

display whole data sorted in ascending or descending order .Also find how

many maximum cmparisons may require for finding any keyword .Use height

balanced tree and find complexiety

**Name : -** Mehatab Sanadi

**Roll No. : -** CO2058

**Class : -** SE Computer

**Subject : -** Data Structure and Algorithms

**Date : -**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*

**Program**

#include<iostream>

#include<cstring>

#include<cstdlib>

#define MAX 50

#define SIZE 20

using namespace std;

struct AVLnode

{

public:

char cWord[SIZE],cMeaning[MAX];

AVLnode \*left,\*right;

int iB\_fac,iHt;

};

class AVLtree

{

public:

AVLnode \*root;

AVLtree()

{

root=NULL;

}

int height(AVLnode\*);

int bf(AVLnode\*);

AVLnode\* insert(AVLnode\*,char[SIZE],char[MAX]);

AVLnode\* rotate\_left(AVLnode\*);

AVLnode\* rotate\_right(AVLnode\*);

AVLnode\* LL(AVLnode\*);

AVLnode\* RR(AVLnode\*);

AVLnode\* LR(AVLnode\*);

AVLnode\* RL(AVLnode\*);

AVLnode\* delet(AVLnode\*,char x[SIZE]);

void inorder(AVLnode\*);

};

AVLnode \*AVLtree::delet(AVLnode \*curr,char x[SIZE])

{

AVLnode \*temp;

if(curr==NULL)

return(0);

else

if(strcmp(x,curr->cWord)>0)

{

curr->right=delet(curr->right,x);

if(bf(curr)==2)

if(bf(curr->left)>=0)

curr=LL(curr);

else

curr=LR(curr);

}

else

if(strcmp(x,curr->cWord)<0)

{

curr->left=delet(curr->left,x);

if(bf(curr)==-2)

if(bf(curr->right)<=0)

curr=RR(curr);

else

curr=RL(curr);

}

else

{

if(curr->right!=NULL)

{

temp=curr->right;

while(temp->left!=NULL)

temp=temp->left;

strcpy(curr->cWord,temp->cWord);

curr->right=delet(curr->right,temp->cWord);

if(bf(curr)==2)

if(bf(curr->left)>=0)

curr=LL(curr);

else

curr=LR(curr);

}

else

return(curr->left);

}

curr->iHt=height(curr);

return(curr);

}

AVLnode\* AVLtree :: insert(AVLnode\*root,char newword[SIZE],char newmeaning[MAX])

{

if(root==NULL)

{

root=new AVLnode;

root->left=root->right=NULL;

strcpy(root->cWord,newword);

strcpy(root->cMeaning,newmeaning);

}

else if(strcmp(root->cWord,newword)!=0)

{

if(strcmp(root->cWord,newword)>0)

{

root->left=insert(root->left,newword,newmeaning);

if(bf(root)==2)

{

if (strcmp(root->left->cWord,newword)>0)

root=LL(root);

else

root=LR(root);

}

}

else if(strcmp(root->cWord,newword)<0)

{

root->right=insert(root->right,newword,newmeaning);

if(bf(root)==-2)

{

if(strcmp(root->right->cWord,newword)>0)

root=RR(root);

else

root=RL(root);

}

}

}

else

cout<<"\nRedundant AVLnode";

root->iHt=height(root);

return root;

}

int AVLtree :: height(AVLnode\* curr)

{

int lh,rh;

if(curr==NULL)

return 0;

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

return 0;

else

{

lh=lh+height(curr->left);

rh=rh+height(curr->right);

if(lh>rh)

return lh+1;

return rh+1;

}

}

int AVLtree :: bf(AVLnode\* curr)

{

int lh,rh;

if(curr==NULL)

return 0;

else

{

if(curr->left==NULL)

lh=0;

else

lh=1+curr->left->iHt;

if(curr->right==NULL)

rh=0;

else

rh=1+curr->right->iHt;

return(lh-rh);

}

}

AVLnode\* AVLtree :: rotate\_right(AVLnode\* curr)

{

AVLnode\* temp;

temp=curr->left;

curr->left=temp->right;

temp->left=curr;

curr->iHt=height(curr);

temp->iHt=height(temp);

return temp;

}

AVLnode\* AVLtree :: rotate\_left(AVLnode\* curr)

{

AVLnode\* temp;

temp=curr->right;

curr->right=temp->left;

temp->left=curr;

curr->iHt=height(curr);

temp->iHt=height(temp);

return temp;

}

AVLnode\* AVLtree :: RR(AVLnode\* curr)

{

curr=rotate\_left(curr);

return curr;

}

AVLnode\* AVLtree :: LL(AVLnode\* curr)

{

curr=rotate\_right(curr);

return curr;

}

AVLnode\* AVLtree :: RL(AVLnode\* curr)

{

curr->right=rotate\_right(curr->right);

curr=rotate\_left(curr);

return curr;

}

AVLnode\* AVLtree::LR(AVLnode\* curr)

{

curr->left=rotate\_left(curr->left);

curr=rotate\_right(curr);

return curr;

}

void AVLtree :: inorder(AVLnode\* curr)

{

if(curr!=NULL)

{

inorder(curr->left);

cout<<"\n\t"<<curr->cWord<<"\t"<<curr->cMeaning;

inorder(curr->right);

}

}

int main()

{

int iCh;

AVLtree a;

AVLnode \*curr=NULL;

char cWd[SIZE],cMean[MAX];

cout<<"\n--------------------------------------";

cout<<"\n\tAVL TREE IMPLEMENTATION";

cout<<"\n--------------------------------------";

do

{

cout<<"\n--------------------------------";

cout<<"\n\t\tMENU";

cout<<"\n--------------------------------";

cout<<"\n1.Insert\n2.Inorder\n3.Delete\n4.Exit";

cout<<"\n--------------------------------";

cout<<"\nEnter your choice :";

cin>>iCh;

switch(iCh)

{

case 1:

cout<<"\nEnter Word : ";

cin>>cWd;

/\*for(int i;cMean[i]!='\0';i++)

{

if(cMean[i]>='A'&& cMean[i]<='Z')

{

cMean[i]=cMean[i]+32;

}

}

cout<<cMean;\*/

cout<<"\nEnter Meaning : ";

cin.ignore();

cin.getline(cMean,MAX);

a.root=a.insert(a.root,cWd,cMean);

break;

case 2:

cout<<"\n\tWORD\tMEANING";

a.inorder(a.root);

break;

case 3:

cout<<"\nEnter the word to be deleted : ";

cin>>cWd;

curr=a.delet(a.root,cWd);

if(curr==NULL)

cout<<"\nWord not present!";

else

cout<<"\nWord deleted Successfully!";

curr=NULL;

break;

case 4:

exit(0);

}

}while(iCh!=4);

return 0;

}

**Output : -**

--------------------------------------

AVL TREE IMPLEMENTATION

--------------------------------------

--------------------------------

MENU

--------------------------------

1.Insert

2.Inorder

3.Delete

4.Exit

--------------------------------

Enter your choice :1

Enter Word : A

Enter Meaning : 11

--------------------------------

MENU

--------------------------------

1.Insert

2.Inorder

3.Delete

4.Exit

--------------------------------

Enter your choice :1

Enter Word : B

Enter Meaning : 12

--------------------------------

MENU

--------------------------------

1.Insert

2.Inorder

3.Delete

4.Exit

--------------------------------

Enter your choice :1

Enter Word : C

Enter Meaning : 13

--------------------------------

MENU

--------------------------------

1.Insert

2.Inorder

3.Delete

4.Exit

--------------------------------

Enter your choice :1

Enter Word : D

Enter Meaning : 14

--------------------------------

MENU

--------------------------------

1.Insert

2.Inorder

3.Delete

4.Exit

--------------------------------

Enter your choice :1

Enter Word : E

Enter Meaning : 15

--------------------------------

MENU

--------------------------------

1.Insert

2.Inorder

3.Delete

4.Exit

--------------------------------

Enter your choice :3

Enter the word to be deleted : C

Word deleted Successfully!

--------------------------------

MENU

--------------------------------

1.Insert

2.Inorder

3.Delete

4.Exit

--------------------------------

Enter your choice :4

--------------------------------