Roll No.COA244

Assignment 9

INPUT :

#include<iostream> using namespace std;

class node

{

public:

string key; string meaning; node \*left; node \*right;

};

class AVL

{

node \*root; public:

AVL()

{

root=NULL;

}

void create();

node\* insert(node \*cur,node \*temp); node\* balance(node \*temp);

int dif(node \*temp);

int height(node \*temp); int maximum(int a,int b);

node\* LL(node \*par); node\* RR(node \*par); node\* LR(node \*par); node\* RL(node \*par);

void ascending(node \*temp);

node\* delete\_n(node \*root,string key1); void deleten();

node\* extractmin(node \*t); void descending(node \*temp); void display();

bool search(node \*cur,string key1); void search\_value();

};

void AVL::create()

{

char answer; node \*temp; do

{

temp=new node(); cout<<endl<<"Enter keyword:\t"; cin>>temp->key;

cout<<"Enter meaning:\t"; cin>>temp->meaning;

temp->left=temp->right=NULL; root=insert(root,temp);

cout<<endl<<"Add another word? (y/n):\t"; cin>>answer;

}

while(answer=='y'||answer=='Y');

}

node\* AVL::insert(node \*cur,node \*temp)

{

if(cur==NULL)

{

return temp;

}

if(temp->key<cur->key)

{

cur->left=insert(cur->left,temp); cur=balance(cur);

}

else if(temp->key>cur->key)

{

cur->right=insert(cur->right,temp); cur=balance(cur);

}

return cur;

}

node\* AVL::balance(node \*temp)

{

int bal; bal=dif(temp);

if(bal>=2)

{

if(dif(temp->left)<0) temp=LR(temp);

else

temp=LL(temp);

}

else if(bal<=-2)

{

if(dif(temp->right)<0) temp=RR(temp);

else

temp=RL(temp);

}

return temp;

}

int AVL::dif(node \*temp)

{

int l,r;

l=height(temp->left); r=height(temp->right); return(l-r);

}

int AVL::height(node \*temp)

{

if(temp==NULL) return(-1);

else

return(max(height(temp->left),height(temp->right))+1);

}

int AVL::maximum(int a,int b)

{

if(a>b)

return a; else

return b;

}

node\* AVL::LL(node \*par)

{

node \*temp,\*temp1; temp=par->left; temp1=temp->right; temp->right=par; par->left=temp1; return temp;

}

node\* AVL::RR(node \*par)

{

node \*temp,\*temp1; temp=par->right; temp1=temp->left; temp->left=par;

par->right=temp1; return temp;

}

node\* AVL::LR(node \*par)

{

par->left=RR(par->left); return(LL(par));

}

node\* AVL::RL(node \*par)

{

par->right=LL(par->right); return(RR(par));

}

void AVL::ascending(node \*temp)

{

if(temp!=NULL)

{

ascending(temp->left);

cout<<"\n\t"<<temp->key<<" : "<<temp->meaning; ascending(temp->right);

}

}

void AVL::descending(node \*temp)

{

if(temp!=NULL)

{

descending(temp->right);

cout<<"\n\t"<<temp->key<<" : "<<temp->meaning; descending(temp->left);

}

}

void AVL::display()

{

cout<<endl<<"Keywords in ascending order:\t"; ascending(root);

cout<<endl<<"Keywords in descending order:\t"; descending(root);

}

bool AVL::search(node \*cur,string key1)

{

if(cur)

{

if(cur->key==key1) return true;

if(cur->key>key1)

return search(cur->left,key1); else

return search(cur->right,key1);

}

return false;

}

void AVL::search\_value()

{

string key2;

cout<<endl<<"Keyword to search:\t"; cin>>key2;

if(search(root,key2))

cout<<endl<<"Keyword exists in AVL tree.";

else

cout<<endl<<"Keyword does not exist in AVL tree.";

}

node\* AVL::delete\_n(node\* cur,string key1)

{

if ( !cur) return cur;

if ( key1 < cur->key )

cur->left = delete\_n(cur->left, key1);

else if( key1 > cur->key )

cur->right = delete\_n(cur->right, key1);

else

{

node \*l = cur->left; node \*r = cur->right; delete cur;

if ( !r )

return l; node \*m=r;

while(m->left) m=m->left;

m->right = extractmin(r); m->left = l;

return balance(m);

}

return balance(cur);

}

node\* AVL::extractmin(node \*t)

{

if ( !t->left ) return t->right;

t->left = extractmin(t->left); return balance(t);

}

void AVL::deleten()

{

string key;

cout<<endl<<"Keyword to delete:\t"; cin>>key;

root=delete\_n(root,key);

}

int main()

{

char c; int ch; AVL a;

do

{

cout<<endl<<"--- MAIN MENU ---";

cout<<endl<<"1 -> Insert keyword"; cout<<endl<<"2 -> Display AVL tree"; cout<<endl<<"3 -> Search a keyword"; cout<<endl<<"4 -> Delete a keyword"; cout<<endl<<"Choose an option (1-4):\t"; cin>>ch;

switch(ch)

{

case 1 : a.create(); break;

case 2 : a.display(); break;

case 3 : a.search\_value(); break;

case 4 : a.deleten(); break;

default : cout<<endl<<"Please choose a valid option (1-4).";

}

cout<<endl<<"Would you like to continue? (y/n):\t"; cin>>c;

}

while(c=='y'||c=='Y'); cout<<"\n\n// END OF CODE\n\n";

return 0;

}

OUTPUT :

