**数据结构第二次实验源代码（编译通过）**

1.BST

#include<iostream>

#include<string>

using namespace std;

class BinNodeptr{

public:

int it;

BinNodeptr\* Left;

BinNodeptr\* Right;

public:

BinNodeptr(){

Left=Right=NULL;

}

BinNodeptr(int value,BinNodeptr\* l=NULL,BinNodeptr\* r=NULL){

Left=l;

Right=r;

it=value;

}

~BinNodeptr(){}

int& val(){

return it;

}

void setVal(const int value){

it=value;

}

BinNodeptr\* left()const{

return Left;

}

BinNodeptr\* right()const{

return Right;

}

void setLeft(BinNodeptr\* node=NULL){

Left=node;

}

void setRight(BinNodeptr\* node=NULL){

Right=node;

}

bool isleaf(){

return (Left==NULL)&&(Right==NULL);

}

};

class BST:public BinNodeptr{

public:

BinNodeptr\* root;

int nodecount;

int height;

public:

BST(){

root=NULL;

nodecount=0;

height=0;

}

~BST(){}

BinNodeptr\* insert(BinNodeptr\*& root,BinNodeptr\* node){

if(root==NULL){

root=node;

return root;

}

else{

if(node->val()<root->val()&&root->left()==NULL){root->setLeft(node);return root;}

if(node->val()>=root->val()&&root->right()==NULL){root->setRight(node);return root;}

if(node->val()<root->val()&&root->left()!=NULL)insert(root->Left,node);

if(node->val()>=root->val()&&root->right()!=NULL)insert(root->Right,node);

}

}

int max(int m,int n){

if(m>=n)return m;

return n;

}

int bstheight(BinNodeptr\* root){

if(root==NULL)return 0;

return 1+max(bstheight(root->Left),bstheight(root->Right));

}

BinNodeptr\* getRoot(){

return root;

}

};

int main(){

int n;

cin>>n;

int value;

char divide;

BST\* mybst=new BST();

while(n--){

mybst->root=NULL;

//mybst->setRoot();

divide='\0';

while(divide!='\n'){

cin>>value;

cin.get(divide);

mybst->insert(mybst->root,new BinNodeptr(value));

}

cout<<mybst->bstheight(mybst->root)<<endl;

}

}

2.编译未通过

#include "stdio.h"

#include "malloc.h"

#include "stack.h"

struct node

{

int data;

node \*right;

node \*left;

};

node \*root;

void insert(node \*&root,int data)

{

if(root==NULL)

{

printf("%d\n",data);

root=(node\*)malloc(sizeof(node));

root->data=data;

root->right=NULL;

root->left=NULL;

}

else

{

if(root->data<data)//要插入的数据data大于节点则插入右边

insert(root->right,data);

else

insert(root->left,data);

}

}

node \*creatTree(int a[],int n)

{

int i;

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

insert(root,a[i]);

return root;

}

void leveltree(node \*root)//层序遍历

{

if(!root)

return;

deque<node\*> dequelist; //双端都可以插入的队列

dequelist.push\_back(root); //插入一个元素到队列的尾部

while(dequelist.size())

{

node\* pnode=dequelist.front();//获取队列的头

dequelist.pop\_front();//弹出队列的头

printf("data=%d\n",pnode->data);

if(pnode->left)

dequelist.push\_back(pnode->left); //若有左子树，把它的左子树压入队列中

if(pnode->right)

dequelist.push\_back(pnode->right); //若有右子树，把它的右子树压入队列中

}

}

int main()

{

int a[]={8,6,5,7,10,9,11};

root=creatTree(a,7);

leveltree(root);//采用循环的方式

return 0;

}