#include <stdlib.h>

#include    <stdio.h>

typedef struct node

{

    int data;

    struct node \*left;

    struct node \*right;

}Node;

*//先序遍历*

void preorder(Node\* *node*)

{

    if(node != NULL)

    {

        printf("%d\n",node->data);

        preorder(node->left);

        preorder(node->right);

    }

}

*//中序遍历*

void inorder(Node\* *node*)

{

    if(node!=NULL)

    {

        inorder(node->left);

        printf("%d\n",node->data);

        inorder(node->right);

    }

}

*//后续遍历*

void postorder(Node\* *node*)

{

    if(node!=NULL)

    {

        postorder(node->left);

        postorder(node->right);

        printf("%d\n",node->data);

    }

}

int main()

{

    Node n1;

    Node n2;

    Node n3;

    Node n4;

    n1.data=5;

    n2.data=6;

    n3.data=7;

    n4.data=8;

    n1.left=&n2;

    n1.right=&n3;

    n2.left=&n4;

    n2.right=NULL;

    n3.left=NULL;

    n3.right=NULL;

    n4.left=NULL;

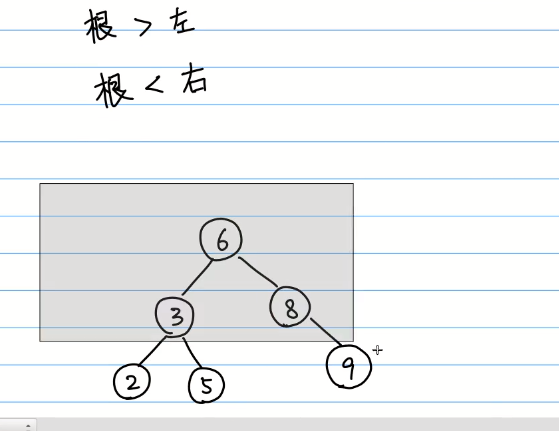
    n4.right=NULL;

   inorder(&n1);

}

中序遍历顺序排列!!!

BST



#include <stdlib.h>

#include    <stdio.h>

typedef struct node

{

    int data;

    struct node \*left;

    struct node \*right;

}Node;

typedef struct

{

    Node\* root;

}Tree;

*//先序遍历*

void preorder(Node\* *node*)

{

    if (node != NULL)

    {

        printf("%d\n", node->data);

        preorder(node->left);

        preorder(node->right);

    }

}

*//中序遍历*

void inorder(Node\* *node*)

{

    if (node != NULL)

    {

        inorder(node->left);

        printf("%d\n", node->data);

        inorder(node->right);

    }

}

*//后续遍历*

void postorder(Node\* *node*)

{

    if (node != NULL)

    {

        postorder(node->left);

        postorder(node->right);

        printf("%d\n", node->data);

    }

}

void insert(Tree\* *tree*,int *value*)

{

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

    node->data = value;

    node->left =NULL;

    node->right =NULL;

    if(tree->root==NULL)

    {

        tree->root =node;

    }

    else

    {

        Node\* temp=tree->root;

        while (temp !=NULL)

        {

            if(value<temp->data)

            {

                if(temp->left==NULL)

                {

                    temp->left=node;

                    return;

                }

                else

                {

                    temp=temp->left;

                }

            }

            else

            {

                if(temp->right==NULL)

                {

                    temp->right=node;

                    return ;

                }

                else

                {

                    temp=temp->right;

                }

            }

        }

    }

}

int get\_height(Node\* *node*)

{

    if(node==NULL)

    {

        return 0;

    }

    else

    {

        int left\_h=get\_height(node->left);

        int right\_h=get\_height(node->right);

        int max=left\_h;

        if(right\_h>max)

        {

            max=right\_h;

        }

        return max+1;

    }

}

int get\_maximum(Node\* *node*)

{

    if(node==NULL)

    {

        return -1;

    }

    else

    {

        int m1=get\_maximum(node->left);

        int m2=get\_maximum(node->right);

        int m3=node->data;

        int max=m1;

        if(m2>max)

        {

            max=m2;

        }

        if(m3>max)

        {

            max=m3;

        }

        return max;

    }

}

int main()

{

    int arr[7]={6,3,8,2,5,1,7};

    Tree tree;

    tree.root=NULL;

    int i;

    for(i=0;i<7;i++)

    {

        insert(&tree,arr[i]);

    }

    int h=get\_height(tree.root);

    printf("%d\n",h);

    int m=get\_maximum(tree.root);

    printf("max=%d\n",m);

}

