二叉树遍历题目描述：

根据给定的二叉树结构描述字符串，输出该二叉树按照中序遍历结果字符串。中序遍历顺序为：左子树，根结点，右子树。

输入描述：

由大小写字母、左右大括号、逗号组成的字符串：

1、字母代表一个节点值，左右括号内包含该节点的子节点。

2、左右子节点使用逗号分隔，逗号前为空则表示左子节点为空，没有逗号则表示右子节点为空。

3、二叉树节点数最大不超过 100。

注：输入字符串格式是正确的，无需考虑格式错误的情况。

输出描述：

输出一个字符串，为二叉树中序遍历各节点值的拼接结果。

补充说明：

中序遍历是二叉树遍历的一种，遍历方式是首先遍历左子树，然后访问根结点，最后遍历右子树。

示例 1

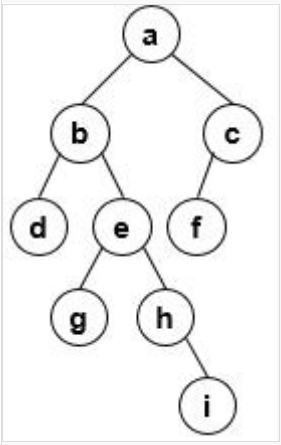
输入：

```
a{b{d,e{g,h{,i}}},c{f}}
```

输出：

```
dbgehiafc
```

说明：



中序遍历，首先遍历左子树，再访问根节点，最后遍历右子树，比如：

a 有左子树，访问其左子树

b 有左子树，访问其左子树

d 没有左子树，读取值"d"

b 的左子树已经访问，读取值‘‘b’’，再访问其右子树

e 有左子树，访问其左子树

g 没有左子树，读取其值‘‘g’’

e 的左子树已经访问，读取值‘‘e’’，再访问其右子树

依次类推……

```
#include <cstddef>
#include <cstdio>
#include <iostream>
#include <type_traits>
#include <stack>
using namespace std;
struct Node
{
	char val;
	Node * left;
	Node * right;
	Node():
	val(0),
	left(nullptr),
	right(nullptr)
	{

	}
};
using Tree =Node *;
void PrintTree(Tree tree)
{
	if(!tree) return;
	if(tree->left)
	{
		PrintTree(tree->left);
	}
	cout << tree->val;
```

```
    if(tree->right)
    {
        PrintTree(tree->right);
    }
}
int main() {
    char ch;
    Tree chTree = new Node;
    Node * AddNode = nullptr , * parent = nullptr;
    stack<Node * > parents;
    parents.push(chTree);
    bool bIsLeft = true;
    //构建树
    while (cin >> ch) { //  注意  while  处理多个  case
        if(ch == '}')
        {
            //AddNode = new Node;
            //chTree = parents.top(); //退回
            parents.pop();
        }
        else if(ch == '{')
        {
            bIsLeft = true;
            parents.push(AddNode);
        }
        else if(ch == ',')
        {
            bIsLeft = false;
        }
        else { //新节点
            AddNode = new Node;
            AddNode->val = ch;
            if(bIsLeft)
            {
                parents.top()->left = AddNode;
            }
            else
            {
                parents.top()->right = AddNode;
            }

        }
    }
    PrintTree(chTree->left);
```

```
}
// 64  位输出请用  printf("%lld")
```