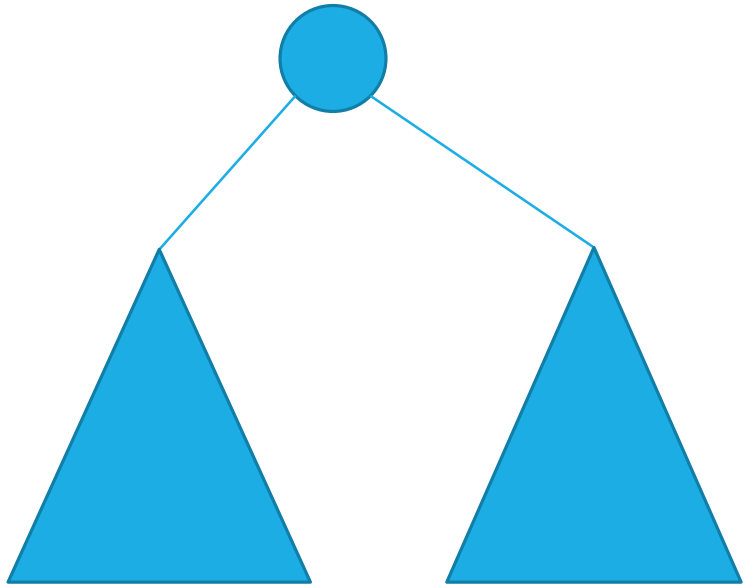
# 二叉树的遍历

- 沿某条搜索路径访问二叉树，对二叉树中的每个结点 **访问一次且仅访问一次**

遍历一棵非空二叉树
- 访问根结点 D
- 遍历左子树 L
- 遍历右子树 R

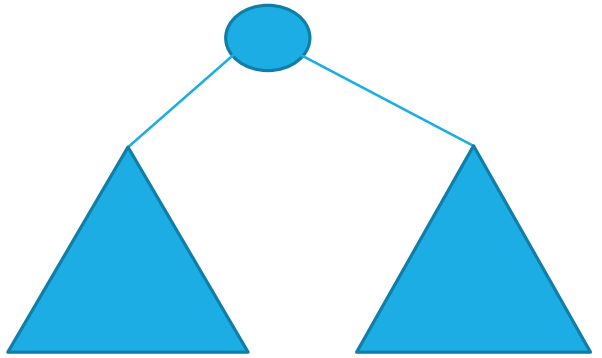


**先根次序遍历二叉树DLR**: 若二叉树非空，则：

① 访问根结点

② 先根次序遍历左子树

③ 先根次序遍历右子树

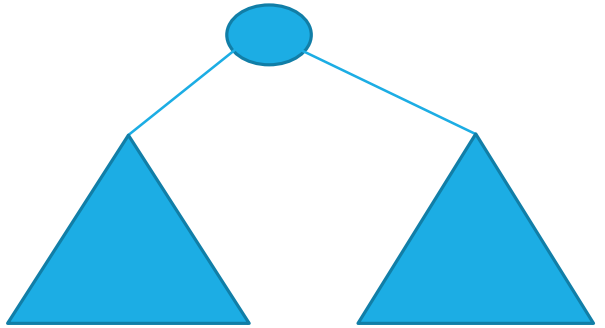


**对称次序遍历二叉树LDR**: 若二叉树非空，则：

① 对称次序遍历左子树

② 访问根结点

③ 对称次序遍历右子树

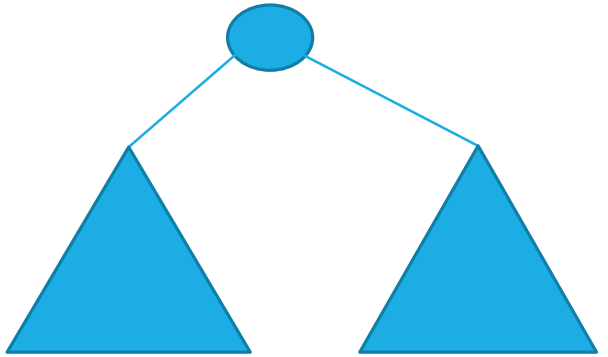


**后根次序遍历二叉树LRD**: 若二叉树非空，则：

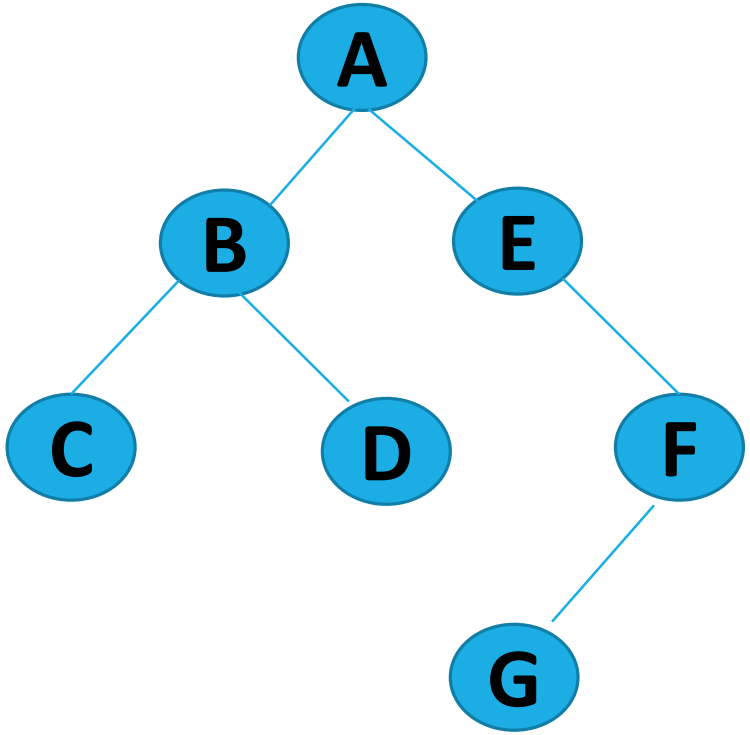
① 后根次序遍历左子树

② 后根次序遍历右子树

③ 访问根结点



# 遍历实例




用DLR遍历：

A B C D E F G

用LDR遍历：

C B D A E G F

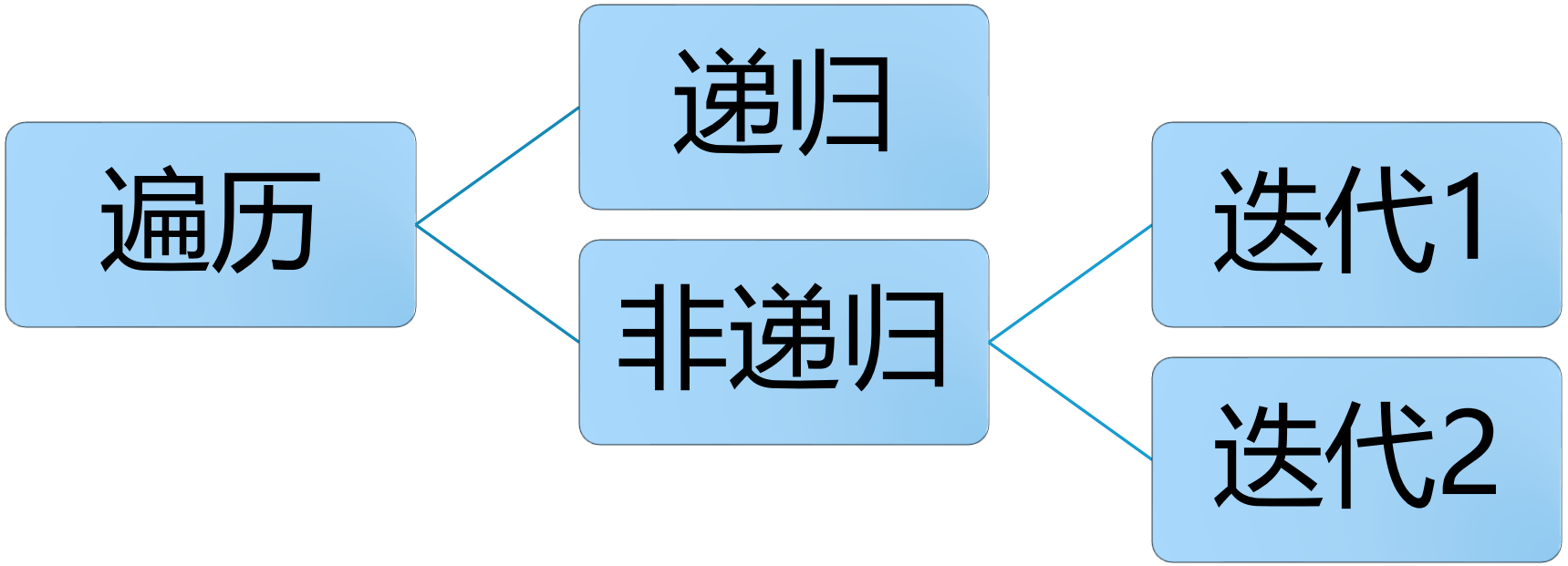
用LRD遍历：

C D B G F E A

半线性结构

非线性结构结点线性化

# 二叉树的遍历算法



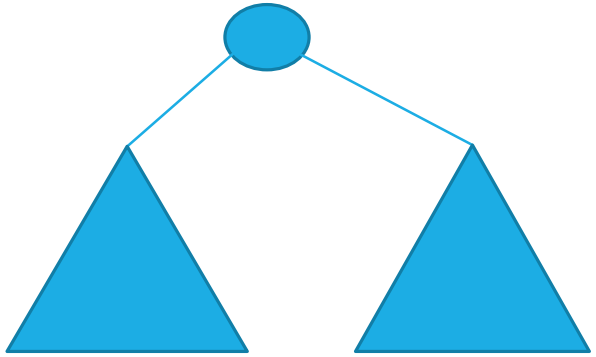
# 递归遍历算法—先根次序

**先根次序遍历二叉树DLR: 若二叉树非空，则：**

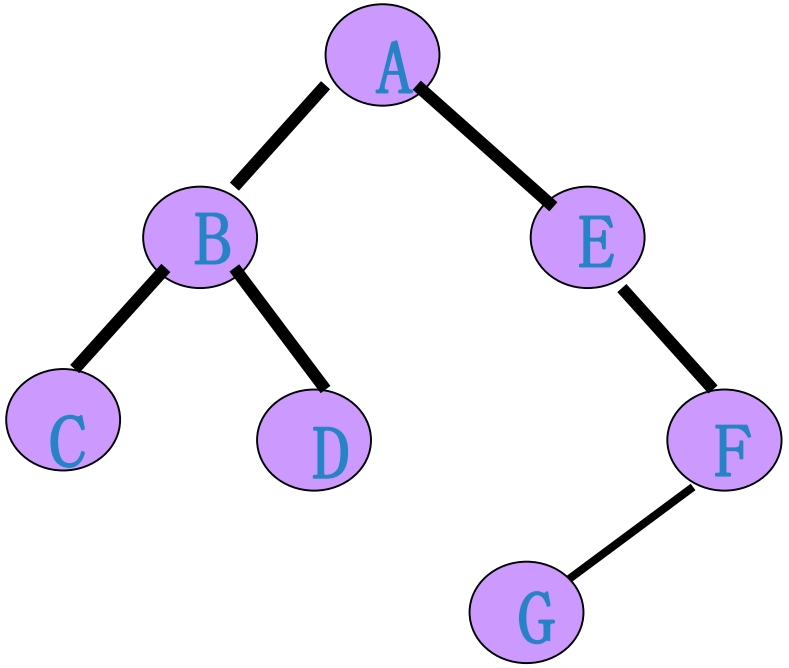
① 访问根结点

② 先根次序遍历左子树

③ 先根次序遍历右子树



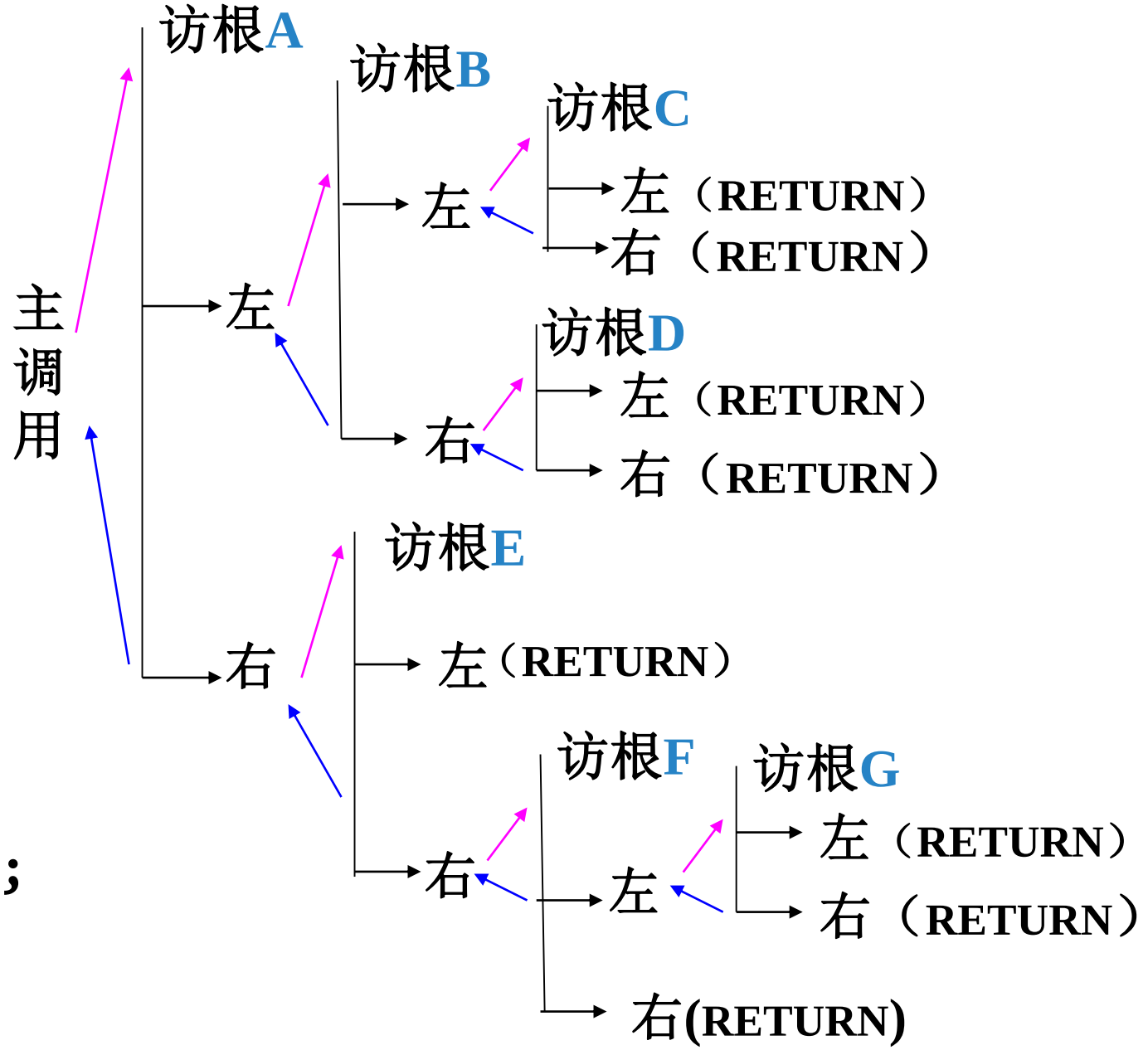
```
void PreOrder（BinTree t）{
if (   t==NULL    ) return;     /*递归调用的结束条件*/
Visit（root(t)）                /*访问结点的数据域*/
 PreOrder （ leftChild(t) ）    /*先根递归遍历t的左子树*/
 PreOrder （ Rightchild(t) ）   /*先根递归遍历t的右子树*/
}
```




```
printf(root(t));

PreOrder(leftchild(t))

PreOrder(rightchild(t));
```



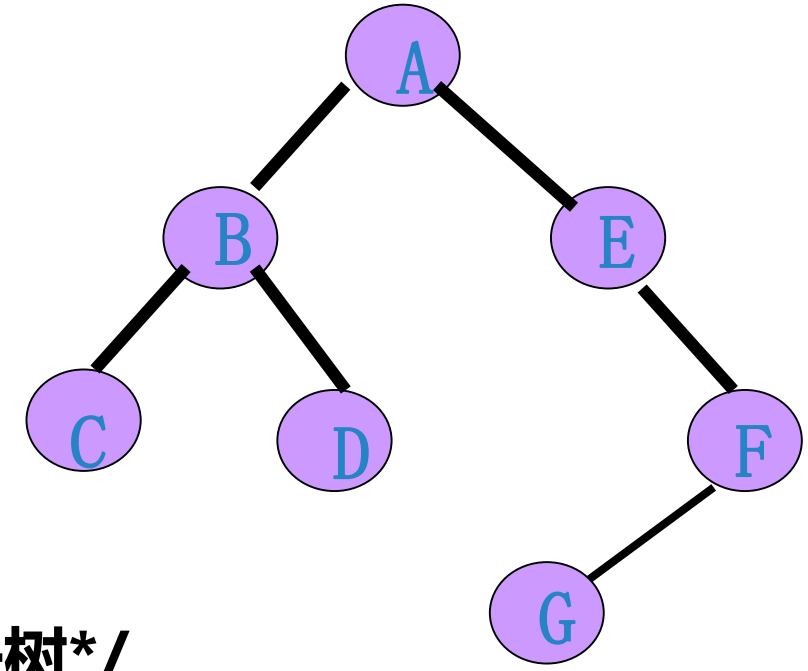
# 递归遍历算法—对称次序

**对称次序遍历二叉树LDR: 若二叉树非空，则：**

① 对称次序遍历左子树

② 访问根结点

③ 对称次序遍历右子树




```
void InOrder（BinTree t)
{/*对称遍历二叉树bt*/
  if (t==NULL) return;     /*递归调用的结束条件*/
  InOrder（leftchild(t)）;   /*对称递归遍历t的左子树*/
  Visit（root(t)）;       /*访问结点的数据域*/
  InOrder（rightchild(t)）;   /*对称递归遍历t的右子树*/
}
```

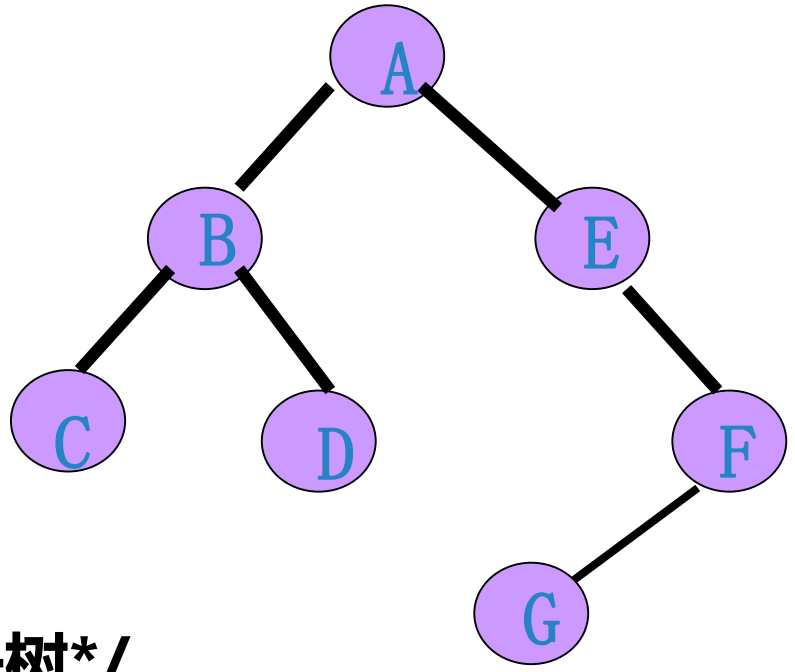
# 递归遍历算法—后根次序

**后根次序遍历二叉树LRD: 若二叉树非空，则：**

① 后根次序遍历左子树

② 后根次序遍历右子树

③ 访问根结点




```
void PostOrder（BinTree t）{
if (bt==NULL) return;          /*递归调用的结束条件*/
  PostOrder（leftchild(t)）；  /*后根递归遍历t的左子树*/
  PostOrder（rightchild(t)）； /*后根递归遍历t的右子树*/
  Visit（root(t)）；           /*访问结点的数据域*/
}
```