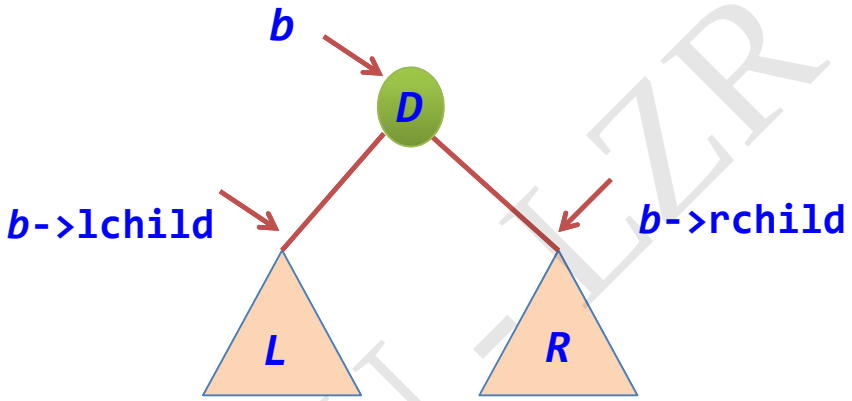
# 二叉树非递归算法设计

# 1．先序遍历非递归算法-1



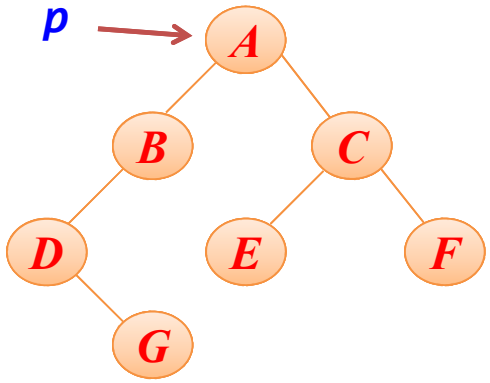

先序序列：*DLR*

- 用栈保存根结点（指针）
- 右孩子先进、左孩子后进栈，因为栈后进先出。

## 先序遍历非递归过程1如下：

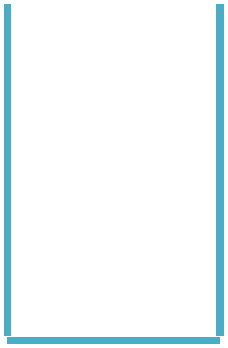
```
if (当前b树不空)
{
    根结点b进栈;
    while (栈不空)
    {
        Step1: 出栈结点p并访问之;
        Step2: 若p结点有右孩子，将其右孩子进栈;
        Step3: 若p结点有左孩子，将其左孩子进栈;
    }
}
```

# 先序非递归算法1动画演示




先序序列：

*A B D G C E F*



指针栈

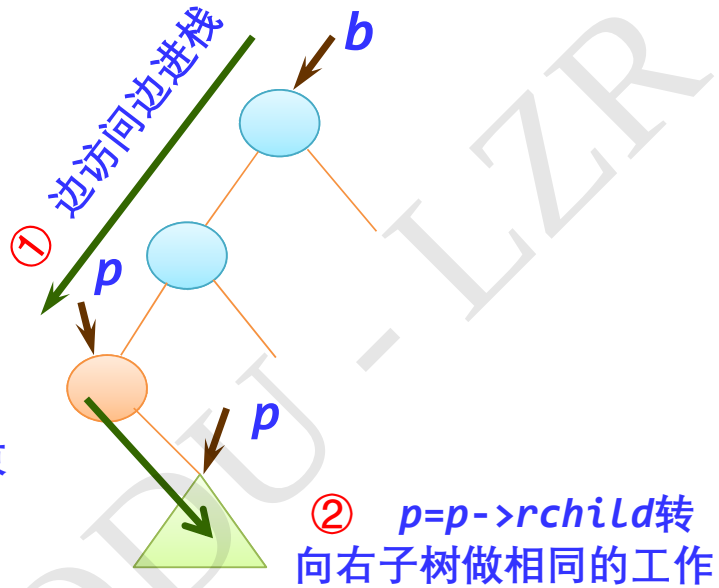
先序遍历完毕

## 先序遍历非递归算法1的实现：

```
void PreOrder1(BiTNode *b)
{
    BiTNode *p;
    SqStack *st;                        //定义栈指针st
    InitStack(st);                      //初始化栈st
    if(b != NULL) {
        Push(st, b);                            //根结点进栈
        while(!StackEmpty(st)) {                //栈不为空时循环
            Pop(st, p);                         //退栈结点p并访问它
            printf("%c ", p->data);
            if(p->rchild != NULL)               //有右孩子时将其进栈
                Push(st, p->rchild);
            if(p->lchild != NULL)               //有左孩子时将其进栈
                Push(st, p->lchild);
        }
    }
}
```

# 2．先序遍历非递归算法-2

*p*用于结点遍历，初始时*p*=*b*



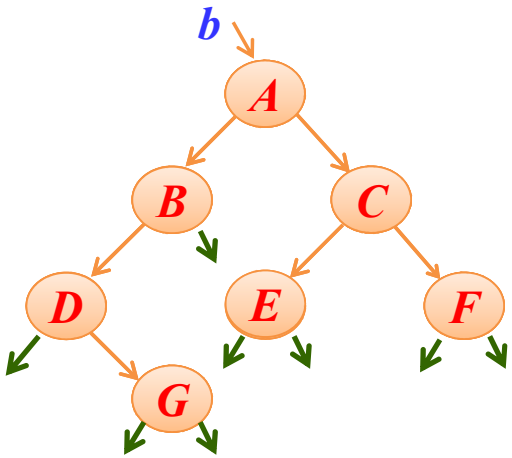

- 栈中结点均已经访问
- *p*指向刚刚出栈结点的右子树

→ 栈空且*p*=NULL结束

先序遍历非递归过程2如下：

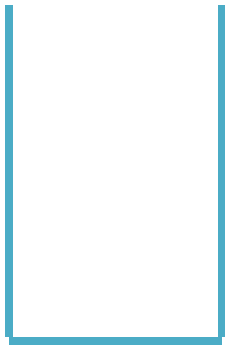
```
p＝b;
while (栈不空或者p!=NULL)
{
    while (p!=NULL)
    {   访问p所指结点;
        将p进栈;
        p=p->lchild;
    }
   //以下考虑栈顶结点
    if (栈不空)
    {   出栈p;
        p＝p->rchild;
    }
}
```

# 先序非递归算法2动画演示




先序序列：

*A B D G C E F*



指针栈

栈空　且 *p*=NULL

先序遍历完毕

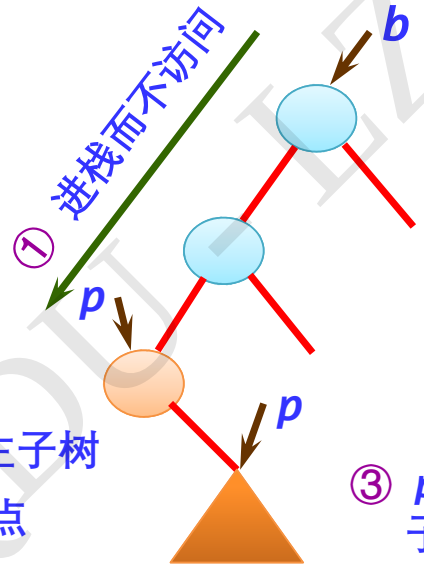
## 先序遍历非递归算法2如下：

```
void PreOrder2(BiTNode *b)
{
    BiTNode *p;
    SqStack *st;                    //定义一个顺序栈指针st
    InitStack(st);                  //初始化栈st
    p = b;
    while(!StackEmpty(st) || p != NULL) {
        while(p != NULL) {  //访问结点p及其所有左下结点并进栈
            printf("%c ", p->data);     //访问结点p
            Push(st, p);                //结点p进栈
            p = p->lchild;              //移动到左孩子
        }
        //以下考虑栈顶结点
        if(!StackEmpty(st)) {    //若栈不空
            Pop(st, p);          //出栈结点p
            p = p->rchild;       //转向处理其右子树
        }
    }
}
```

## 2．中序遍历非递归算法

在先序遍历非递归算法2的基础上改进而来的

*p*用于结点遍历，初始时*p*=*b*，当*p*=NULL并且栈为空结束




这个最左下结点，没有左子树

② 访问栈顶结点

③ *p*=*p*->*rchild*转向右子树做相同的工作

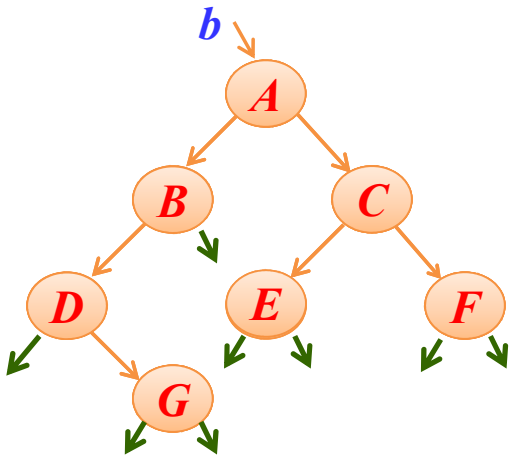
- 栈中结点均没有访问
- *p*指向刚刚出栈结点的右子树

→ 栈空且*p*=NULL结束

中序遍历非递归过程如下：

```
p＝b;
while (栈不空或者p!=NULL)
{
      while (p!=NULL)
      {    将p进栈；
           p=p->lchild;
       }
       //以下考虑栈顶结点
      if (栈不空)
      {     出栈p并访问之；
           p＝p->rchild;
       }
}
```

# 中序非递归算法动画演示

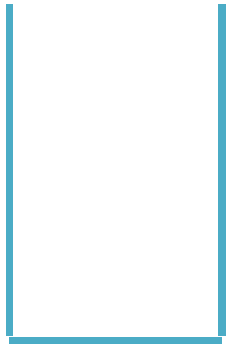



中序序列：

*D G B A E C F*

栈空 且 *p*=NULL

**中序遍历完毕**



指针栈

## 中序遍历非递归算法如下：

```
void InOrder(BiTNode *b)
{
    BiTNode *p;
    SqStack *st;                    //定义一个顺序栈指针st
    InitStack(st);                  //初始化栈st
    p = b;
    while(!StackEmpty(st) || p != NULL) {
        while(p != NULL) {    //扫描结点p的所有左下结点并进栈
            Push(st, p);          //结点p进栈
            p = p->lchild;        //移动到左孩子
        }
        //以下考虑栈顶结点
        if(!StackEmpty(st)) {             //若栈不空
            Pop(st, p);                   //出栈结点p，访问结点p
            printf("%c ", p->data);
            p = p->rchild;                //转向处理其右子树
        }
    }
}
```

```
void InOrderTraverse2(BiTree T, void(*Visit)(TElemType))
{
  // 采用二叉链表存储结构，Visit是对数据元素操作的应用函数。算法6.2，
  // 中序遍历二叉树T的非递归算法(利用栈)，对每个数据元素调用函数Visit
  SqStack S;
  BiTree p;
  InitStack(S);
  Push(S, T);                          // 根指针进栈
  while(!StackEmpty(S)) {
    while(GetTop(S, p) && p)
      Push(S, p->lchild);              // 向左走到尽头
    Pop(S, p);                         // 空指针退栈
    if(!StackEmpty(S)) {               // 访问结点，向右一步
      Pop(S, p);
      Visit(p->data);
      Push(S, p->rchild);
    }
  }
}
```

END