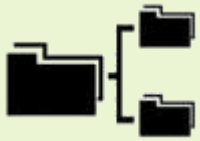


本节主题：

二叉树的基本运算及其实现

二叉树的基本运算

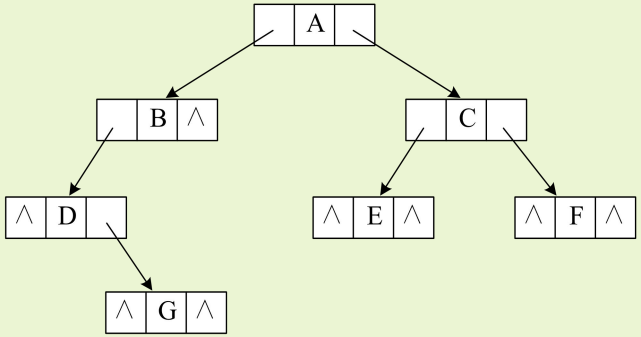
（1）创建二叉树**CreateBTNode(*b,*str)**：根据二叉树括号表示法的字符串*str生成对应的链式存储结构。

（2）查找节点**FindNode(*b,x)**：在二叉树b中寻找data域值为x的节点，并返回指向该节点的指针。

（3）找孩子节点**LchildNode(p)**和**Rchild-Node(p)**：分别求二叉树中节点*p的左孩子节点和右孩子节点。

（4）求高度**BTNodeDepth(*b)**：求二叉树b的高度。若二叉树为空，则其高度为0；否则，其高度等于左子树与右子树中的最大高度加1。

（5）输出二叉树**DispBTNode(*b)**：以括号表示法输出一棵二叉树。




```
typedef struct node
{
    ElemType data;
    struct node *lchild,*rchild;
} BTNode;
```

创建二叉树CreateBTNode(*b,*str)

- 例：str —— A (B (D (, G)) , C (E , F))
- 扫描采用括号表示法表示二叉树的字符串，读到的符号为ch
- 使用一个栈St保存双亲节点，k指定其后处理的节点是双亲节点（保存在栈中）的左孩子节点（k=1）还是右孩子节点（k=2）。
- 分以下几种情况：

  ① 若ch='('：则将前面刚创建的节点作为双亲节点进栈，并置k=1，表示其后创建的节点将作为这个节点的左孩子节点；

  ② 若ch=‘,’：表示其后创建的节点为右孩子节点，置k=2；

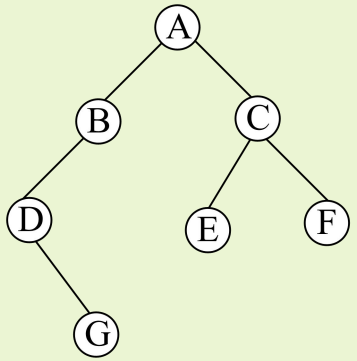
  ③ 若ch=')'：表示栈中节点的左右孩子节点处理完毕，退栈；

  ④ 其他情况：

  当k=1时，表示这个节点作为栈中节点的左孩子节点；

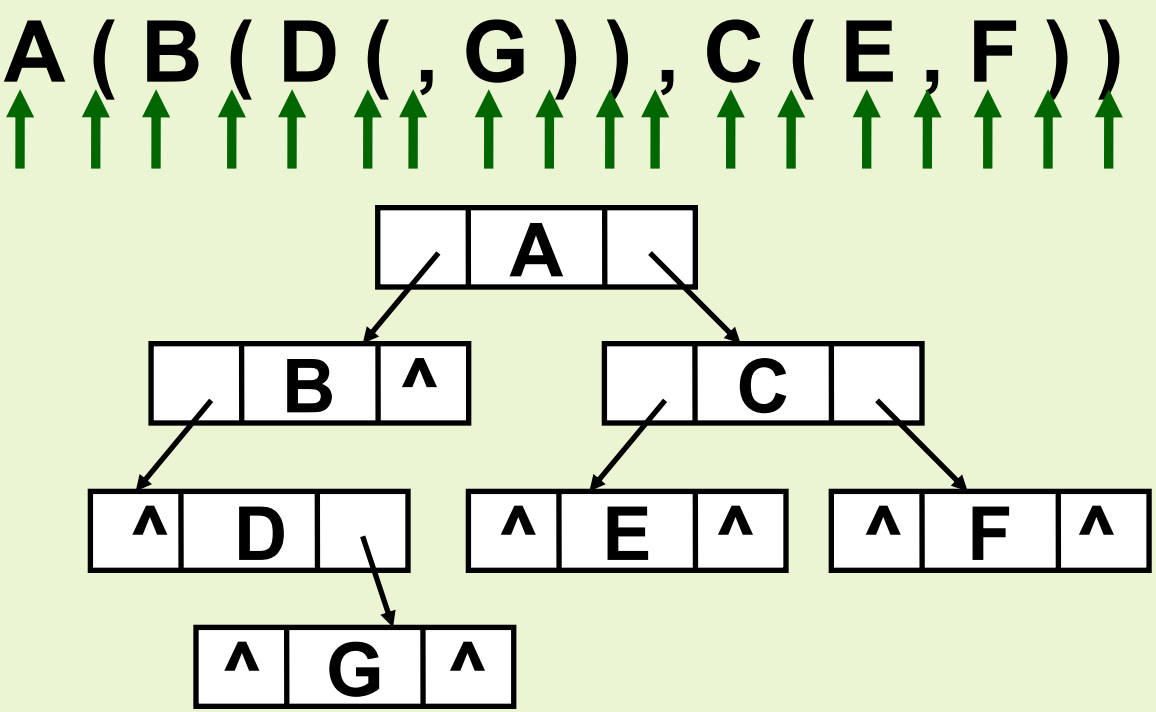
  当k=2时，表示这个节点作为栈中节点的右孩子节点。

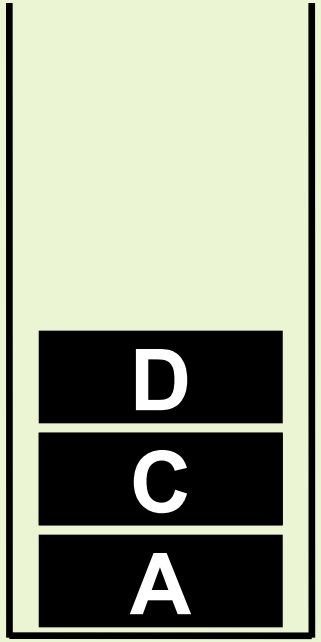
- 如此循环直到str处理完毕。



创建实例

A (B (D (, G)) , C (E , F))





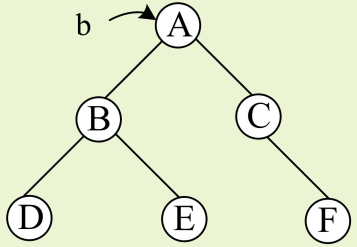
k= 2

算法描述

```
void CreateBTNode(BTNode * &b,char *str)
{
   BTNode *St[MaxSize],*p=NULL;
   int top=-1,k,j=0;
   char ch;
   b=NULL;
   ch=str[j];
   while (ch!='\0')
   {
      switch(ch)
      {
      case '(':
         top++;
         St[top]=p;
         k=1;
         break;
      case ')':
         top--;
         break;
      case ',':
         k=2;
         break;
      default:
         p=(BTNode *)malloc(sizeof(BTNode));
         p->data=ch;
         p->lchild=p->rchild=NULL;
         if (b==NULL)     //p为二叉树的根
            b=p;
         else    //p要作为已建立的二叉树的子树
         {
            switch(k)
            {
            case 1:
               St[top]->lchild=p;
               break;
            case 2:
               St[top]->rchild=p;
               break;
            }
         }
      }
      j++;
      ch=str[j];
   }
}
```

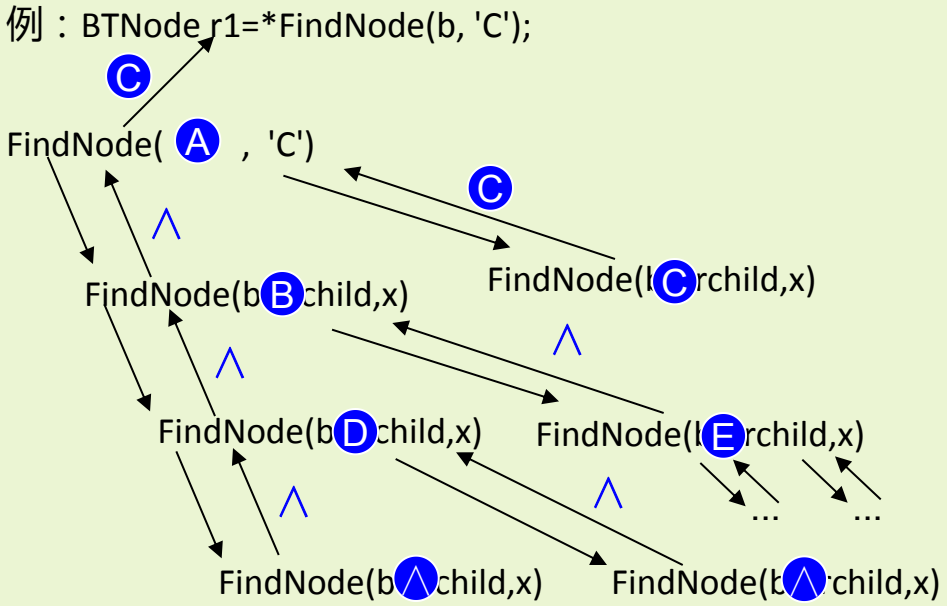
str: A (B (D (, G)) , C (E , F))

查找节点FindNode(*b,x)



- 找到值为x的结点后返回节点指针，否则返回NULL。
- 用递归算法，采用"根-左子树-右子树"的顺序，查找值为x的节点。

```
BTNode *FindNode(BTNode *b,ElemType x)
{
   BTNode *p;
   if (b==NULL)
      return NULL;
   else if (b->data==x)
      return b;
   else
   {
      p=FindNode(b->lchild,x);
      if (p!=NULL)
         return p;
      else
         return FindNode(b->rchild,x);
   }
}
```



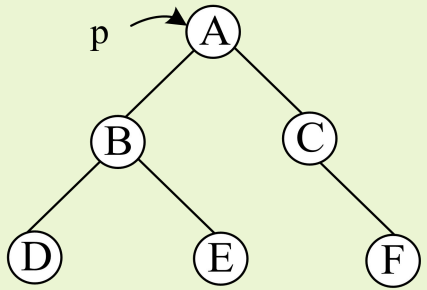
再例：BTNode r2=*FindNode(b, 'K');

找孩子节点LchildNode(p)和RchildNode(p)

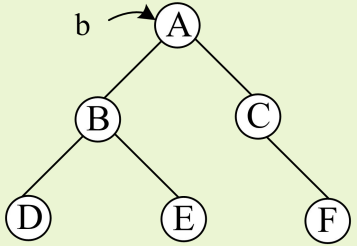
- 直接返回*p节点的左孩子节点或右孩子节点的指针。

```
BTNode *LchildNode(BTNode *p)
{
    return p->lchild;
}
```

```
BTNode *RchildNode(BTNode *p)
{
    return p->rchild;
}
```



求高度BTNodeDepth(*b)



- 二叉树的高度的递归模型f()

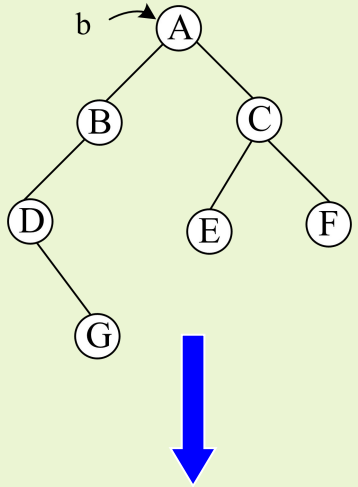
$$f(b)=\begin{cases}0 & ,b=NULL\\ MAX\{f(b->lchild),f(b->rchild)\}+1 & 其他情况\end{cases}$$

```
int BTNodeDepth(BTNode *b)
{
    int lchilddep,rchilddep;
    if (b==NULL)
        return(0); //空树的高度为0
    else
    {
        lchilddep=BTNodeDepth(b->lchild); //求左子树的高度
        rchilddep=BTNodeDepth(b->rchild); //求右子树的高度
        return (lchilddep>rchilddep)? (lchilddep+1):(rchilddep+1));
    }
}
```

输出二叉树DispBTNode(*b)

- 用括弧表示法输出二叉树。
- 对于非空二叉树b
  - 先输出其元素值
  - 当存在左孩子或右孩子节点时
    - 输出一个“(”符号
    - 递归处理左子树
    - 输出一个“,”符号
    - 递归处理右子树
    - 最后输出一个“)”符号

```
void DispBTNode(BTNode *b)
{
   if (b!=NULL)
   {
      printf("%c",b->data);
      if (b->lchild!=NULL || b->rchild!=NULL)
      {
         printf("(");
         DispBTNode(b->lchild);
         if (b->rchild!=NULL)
            printf(",");
         DispBTNode(b->rchild);
         printf(")");
      }
   }
}
```



A (B (D (, G)) , C (E , F))