假设树以孩子兄弟链的方法存储，试分别设计实现它的前序、后序和层次遍历的算法。

【解】为了调试三个遍历函数，我们设计了一个最基本的基于孩子兄弟链的树类 tree。tree 类有五个最基本的函数：构造函数、创建一棵树 createTree、前序遍历 preOrder、后序遍历 postOrder 和层次遍历 levelOrder。事实上，还应该有一个析构函数，否则会有内存泄露。有兴趣的读者可以自己加上这个函数。与二叉树类中的前序遍历和后序遍历一样，它们也可以用递归的方法实现，因此有两个对应的私有成员函数。tree 类的定义如代码清单 5-18 所示。

代码清单 5-18 tree 类的定义

```
template <class T>
class tree {
    struct node { // 树中的结点类型
        T data;
        node *son, *brother;

        node(T d, node *s = NULL, node *b =NULL)
      {data = d; son = s; brother = b;}
        node() {}
    };

    node *root;   // 唯一的数据成员，指向根结点的指针

public:
    tree(node *t = NULL) {root = t;}
    void createTree(T flag);
    void preOrder()  { preOrder(root); }
    void postOrder() { postOrder(root); }
    void levelOrder();

private:
    void preOrder(node *t);
    void postOrder(node *t);
};
```

创建一棵树的过程与创建一棵二叉树基本类似。采用一个队列存放已添加到树上的结点，然后对每个结点输入它的所有儿子，再将儿子入队。重复这个过程，直到队列为空。该过程如代码清单 5-19 所示。

代码清单 5-19 createTree 函数的实现

```
template <class T>
void tree<T>::createTree(T flag)
{ linkQueue< node * > que;
  node *tmp;
  T x, son;
```

```
6.
7.     //创建树，输入 flag 表示空
8.     cout << "\n 输入根结点： ";
9.     cin >> x;
10.    root = new node(x);
11.    que.enQueue(root);
12.
13.    while (!que.isEmpty()) {                    // 处理每个结点的儿子
14.        tmp = que.deQueue();
15.        cout << "\n 输入" << tmp->data << "的所有儿子(以" << flag << "结束)： ";
16.        cin >> son ;                             // 输入第一个儿子
17.        if (son == flag) continue;                    // 该结点没有儿子
18.     tmp->son = new node(son);                  // 为第一个儿子申请空间
19.        tmp = tmp->son;
20.        que.enQueue(tmp);                             // 第一个儿子入队
21.     while (true) {                             // 处理后续的儿子
22.             cin >> son ;
23.             if (son == flag) break;
24.             tmp->brother = new node(son);
25.             tmp = tmp->brother;
26.             que.enQueue(tmp);
27.        }
28.    }
29.    cout << "create completed!\n";
30.  }
```

两个私有的前序和后序遍历函数的实现与二叉树类中的实现非常类似。前序遍历先访问根结点，然后递归调用前序遍历函数遍历它的每一棵子树。后序遍历先递归调用后序遍历函数遍历每一棵子树，然后访问根结点。这两个函数的实现见代码清单 5-20。

代码清单 5-20 私有的前序和后序遍历函数的实现

```
1.    template <class T>
2.    void tree<T>::preOrder(node *t)
3.    {     if (t == NULL) return;
4.          cout << t->data;              // 访问根结点
5.
6.          node *p = t->son;
7.          while (p != NULL) {           // 前序遍历每一棵子树
8.               preOrder(p);
9.               p = p->brother;
10.         }
11.   }
12.
13.   template <class T>
```

```
14. void tree<T>::postOrder(node *t)
15. {     if (t == NULL) return;
16.
17.       node *p = t->son;                  // 找到第一个儿子
18.       while (p != NULL) {                // 后序遍历每一棵子树
19.             postOrder(p);
20.             p = p->brother;
21.       }
22.
23.       cout << t->data;                   // 访问根结点
24. }
```

层次遍历采用一个队列作为工具。先把根结点入队，然后重复从队列中取出元素，访问该元素，把该元素的儿子入队，直到队列为空。它的实现如代码清单 5-21 所示

代码清单 5-21 层次遍历的实现

```
1.  template <class T>
2.  void tree<T>::levelOrder()
3.  {     linkQueue< node * > que;
4.     node *tmp;
5.
6.       if (root == NULL) return;
7.       que.enQueue(root);
8.
9.       while (!que.isEmpty()) {
10.            tmp = que.deQueue();
11.            cout << tmp->data;
12.            tmp = tmp->son;
13.            while (tmp != NULL) {
14.                  que.enQueue(tmp);
15.                  tmp = tmp->brother;
16.            }
17.      }
18. }
```