///5.3.2节 算法FIO

```
//返回以结点t为根结点的中序线索二叉树的中根序列的第一个结点
template<class T>
ThreadNode<T> * ThreadInTree<T> :: FIO(ThreadNode<T> *t)
{
    if(t==NULL) return NULL;
    ThreadNode<T> *q = t;
    while ((q -> GetLThread()) == 0)        // 寻找树中最左下方的结点
    q = q ->GetLeft() ;
    return q ;                              // 返回树中最左下方的结点
};
```

5.3节 中序线索二叉树类ThreadInTree的定义

```
/* 中序线索二叉树结点类声明*/
template<class T>
class ThreadNode
{
private :
    int LThread,RThread ;       // 线索域，用来标记是否有左右子结点
    ThreadNode<T> *left ;       // 若有左子结点，则Left指向左子结点；否则指向其中根前驱结点
    ThreadNode<T> *right ;          // 若有右子结点，则Right指向右子结点；否则指向其中根后继结点
    T data ;
public :
    ThreadNode ( const T item=0 ): data (item) , left ( NULL) , right ( NULL), LThread (0) , RThread (0) { }
    ThreadNode<T> * GetLeft(void)const { return left ; }
    void SetLeft( ThreadNode<T> *t){ left = t ; }
    ThreadNode<T> * GetRight(void)const { return right ; }
    void SetRight( ThreadNode<T> *t){ right = t ;}
    void SetData(const T & item){ data = item ; }
    T& GetData() {    return    data;   }
    void SetLThread(const int d){ LThread=d ; }
    int   GetLThread() {    return   LThread;   }
    void SetRThread(const int d){ RThread=d ; }
    int   GetRThread() {    return   RThread;   }
} ;
/* 中序线索二叉树类声明*/
template<class T>
class ThreadInTree
{
private :
    ThreadNode<T> *head;            // 中序线索二叉树的表头结点
    T stop;
    ThreadNode<T> *pre;             //用于中序二叉树的线索化
public :
    ThreadInTree( ){ head = new ThreadNode<T>;
                     head->SetLeft(head),head->SetLThread(1);
                     head->SetRight(head),head->SetRThread(0); }        // 构造函数
    ThreadNode<T> *FIO( ThreadNode<T> *t ); // 返回中序线索二叉树t的中根序列的第一个结点
    ThreadNode<T> *LIO( ThreadNode<T> *t ); // 返回中序线索二叉树t的中根序列的最后一个结点
    //在以t为根结点的中序线索二叉树中搜索结点p的中根前驱结点
    ThreadNode<T> *PIO(ThreadNode<T> *p );
    //在以t为根结点的中序线索二叉树中搜索结点p的中根后继结点
   ThreadNode<T> *NIO(ThreadNode<T> *p);
   void InOrderOf(ThreadNode<T> *t );   //中根遍历以结点t为根结点的中序线索二叉树
    void InOrder( )  { InOrderOf(GetRoot()); };//中根遍历以root为根结点的中序线索二叉树
   void ReverseInOrderOf(ThreadNode<T> *t);//逆向中根遍历以结点t为根结点的中序线索二叉树
    void ReverseInOrder( ) {ReverseInOrderOf(GetRoot());}
```

```
    void ThreadingTree() ;                    //线索化以root为根结点的二叉树为中序线索二叉树
    void InThreading(ThreadNode<T> *t);

    void InsertRight(ThreadNode<T> *p, ThreadNode<T> *s); //插入一个结点p，作为结点s的右子结点
    void InsertLeft(ThreadNode<T> *p, ThreadNode<T> *s); //插入一个结点p，作为结点s的左子结点
    void DeleteRight(ThreadNode<T> *s); //删除结点s的右子结点p
    void DeleteLeft(ThreadNode<T> *s); //删除结点s的左子结点p

    //其他操作
    ThreadNode<T> * GetRoot(){return head->GetLThread()==0 ? head->GetLeft() : NULL;};
    void    SetRoot(ThreadNode<T> * t){ head->SetLeft(t), head->SetLThread( t == NULL ? 1 : 0 ); };
    T GetStop() { return stop ; }
    void SetStop(T tostop) {    stop=tostop ; }
    void CreateThreadingTree(T tostop) ;    //建立以root为根结点的中序线索二叉树
    ThreadNode<T> * Create() ;              //建立以root为根结点的尚未线索的二叉树
    //返回以root为根结点的树的中根序列的第一个结点
    ThreadNode<T> *First    ( )    {return FIO(GetRoot()); };
    //返回以root为根结点的树的中根序列的最后一个结点
    ThreadNode<T> *Last    ( )    {return LIO(GetRoot()); };
    ThreadNode<T> *    search(ThreadNode<T> *t,T item );    //在树t中搜索数据域为item的结点
} ;
```