## 1.树定义

1. typedef **char** ElemType;
2. //通过第一个孩子和下一个兄弟来确定整个树
3. typedef struct TNode
4. {
5. ElemType data;
6. TNode \*firstChild,\*nextSibling;
7. }TNode,\*Tree;

## 2.初始化树

1. //初始化一个节点
2. Tree initTree(ElemType e)
3. {
4. Tree pT;
5. pT = (TNode\*)malloc(sizeof(TNode));
6. pT->firstChild = NULL;
7. pT->nextSibling = NULL;
8. pT->data = e;
9. return pT;
10. }

## 3.删除树

1. //删除树
2. void deleteTree(Tree \*T)
3. {
4. if(NULL == \*T)
5. return ;
6. deleteTree(&(\*T)->nextSibling);
7. deleteTree(&(\*T)->firstChild);
8. free(\*T);
9. \*T = NULL;
10. }

## 4. 遍历树

1. //遍历树并打印
2. void printTree(Tree T)
3. {
4. if(NULL == T)
5. return;
6. else
7. {
8. printf("%c ",T->data);
9. /\*
10. //如果不习惯递归，可以这样写
11. Tree p = T->firstChild;
12. while(p != NULL)
13. {
14. printTree(p);
15. p = p->nextSibling;
16. }
17. \*/
18. //注释掉的部分等价于如下两行：
19. printTree(T->firstChild);
20. printTree(T->nextSibling);
21. }
22. }

## 5.定位

1. //全局变量记录找到的元素的地址
2. Tree result;
3. void locateElem(Tree T,ElemType e)
4. {
5. if(NULL == T)
6. return;
7. if(T->data == e)
8. result = T;
9. /\*
10. Tree p = T->firstChild;
11. while(p != NULL)
12. {
13. locateElem(p,e);
14. p = p->nextSibling;
15. }
16. \*/
17. locateElem(T->firstChild,e);
18. locateElem(T->nextSibling,e);
19. }

## 6.获取树的宽度和深度

1. //全局变量记录找到的元素的地址
2. int depth(TNode \* T)
3. {
4. TNode \*store[50];
5. memset(store,0,50);
6. int length=0;
7. int depth=0;
8. stack<TNode\*> st;
9. if(T)
10. {
11. st.push(T);
12. while(!st.empty())
13. {
14. depth++;
15. length=0;
16. while(!st.empty())
17. {
18. TNode \*p=st.top();
19. st.pop();
20. if(p->m\_left)
21. store[length++]=p->m\_left;
22. if(p->m\_right)
23. store[length++]=p->m\_right;
24. }
25. for(int i=0;i<length;i++)
26. st.push(store[length]);
27. }
28. return depth;
29. }
30. else return 0;
31. }