第 1 题代码参考如下：

```
#include <stdio.h>
//#include <malloc.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define OVERFLOW -2
typedef int Status;
typedef char ElemType;
typedef struct CSNode {
      ElemType data;
      struct CSNode *firstchild, *nextsibling;     //孩子兄弟指针
}CSNode, *CSTree;
int Nodenum=0;   //统计树的结点个数
int Count=0;      //统计树中叶子结点个数

Status CreateCSTree(CSTree &T) {   // 算法 6.4
  // 按先序次序输入树中结点的值（一个字符），#字符表示空树，
  // 构造二叉链表表示的树 T。
  ElemType ch;
  scanf("%c",&ch);
  if (ch=='#') T = NULL;
  else {
    if (!(T = (CSNode *)malloc(sizeof(CSNode)))) exit(OVERFLOW);
    T->data = ch;                          // 生成根结点
    CreateCSTree(T->firstchild);     // 构造左子树
    CreateCSTree(T->nextsibling);     // 构造右子树
  }
  return OK;
} // CreateCSTree

Status PreOrderTraverse(CSTree T) {
    // 算法 6.1
    // 采用二叉链表存储结构
    // 先根遍历树 T 的递归算法。
    if (T) {
          printf("%c", T->data);
          if (PreOrderTraverse(T->firstchild))
              if (PreOrderTraverse(T->nextsibling)) return OK;
          return ERROR;
    } else return OK;
} // PreOrderTraverse
```

```
Status PostOrderTraverse(CSTree T) {
    // 采用二叉链表存储结构
    // 后根遍历树 T 的递归算法。
     int m=0;
     if (T) {
          if (PostOrderTraverse(_____)) m=1;
          printf("%c", T->data);
          if (m)
               if (PostOrderTraverse(_____)) return OK;
          return ERROR;
    } else return OK;
} // InOrderTraverse

// 用递归方法求树的深度
int CSTreeDepth(CSTree T)
{
     int h1, h2;
     if (T == NULL) return 0;
     else {
          h1 = _____;
          h2 = _____;
          if (h1+1>=h2) return h1+1;
          else return h2;
     }
}

Status NodeCountCSTree(CSTree T) {
     //统计树的结点个数
     if (T) {
          Nodenum++;
          if (_____)
               if (_____) return OK;
          return ERROR;
     } else return OK;
} // NodeCountCSTree

Status LeafCountCSTree(CSTree T) {
     //输出树的叶子结点，并统计叶子结点个数
     if (T) {
          if (_____) {
               printf("%c", T->data); Count++;
          }
          if (_____)
               if (_____) return OK;
          return ERROR;
     } else return OK;
} // LeafCount
```

```
Status CopyCSTree(CSTree T, CSTree &B) {
    //复制树
    if (T==NULL)
        B = NULL;
    else {
        if (!(B = (CSNode *)malloc(sizeof(CSNode)))) exit(OVERFLOW);
        B->data = T->data;                          //复制一个根节点*B
        ____________;  //递归复制左子树
        ____________;  //递归复制右子树
    }
    return OK;
} // CopyCSTree

Status DestroyCSTree(CSTree &T) {
  //销毁树
  if (T) {
      DestroyCSTree(____________);
      DestroyCSTree(____________);
      free(T);
  }
  return OK;
} // DestroyCSTree

Status ClearCSTree(CSTree &T) {
  //置树为空树
  if (T) {
      T = NULL;
  }
  return OK;
} // ClearCSTree

void main()
{
    CSTree T, B;
    printf("创建树，按先序次序输入树中结点的值：\n");
    CreateCSTree(T);
    NodeCountCSTree(T);
    printf("树的结点个数为：%d\n", Nodenum);
    printf("树的深度为：%d\n", CSTreeDepth(T));
    printf("先根遍历树，结果是：\n");
    PreOrderTraverse(T);
    printf("\n");
    printf("后根遍历树，结果是：\n");
    PostOrderTraverse(T);
    printf("\n");
    printf("输出树的叶子结点：\n");
    LeafCountCSTree(T);
    printf("\n");
```

```
    printf("统计树的叶子结点个数：%d\n", Count);
    if (CopyCSTree(T, B)==OK) printf("成功复制树 T 到树 B！\n");
    if (DestroyCSTree(T)==OK) printf("成功销毁树 T！\n");
    if (ClearCSTree(T)==OK) printf("将树 T 置为空树！\n");
    printf("先根遍历树 B，结果是：\n");
    PreOrderTraverse(B);
    printf("\n");
    printf("后根遍历树 B，结果是：\n");
    PostOrderTraverse(B);
    printf("\n");
}
```