```c
//2020240567 大数据201 苏恒

#include<stdio.h>
#include<malloc.h>
#include<time.h>

typedef char DataType;
typedef struct Node {
   DataType data;
   struct Node *lchild, *rchild;
} BTNode;
typedef BTNode* pBiTree;

void InitBiTree( pBiTree *pRoot );
void VisitNode( pBiTree root );
void PreOrder( pBiTree root );
void InOrder( pBiTree root );
void PostOrder( pBiTree root );
void CreateBiTree_semi( pBiTree *pRoot );
void DestroyBiTree( pBiTree root );
pBiTree CreateBiTree_full( int n, int h );
int GetDepth(pBiTree root );
pBiTree InsertLeftNode( pBiTree pCurNode, DataType x );
pBiTree InsertRightNode( pBiTree pCurNode, DataType x );

pBiTree root2;

int main(){
   pBiTree root1;
   printf("输入待生成二叉树的前序串: ");
   CreateBiTree_semi( &root1 );
   printf("已生成该二叉树, 前序: ");
   PreOrder( root1 );
   printf(", 中序: ");
   InOrder( root1 );
   printf("。\n");
   printf("\n输入待生成二叉树的结点n和深度h(|_logn_|+1 <= h <= n):");
   int n, h;
   scanf("%d %d", &n, &h);
   while(GetDepth( root2 ) != h ) {
      CreateBiTree_full( n, h );
   }
   printf("\n已生成满足要求的二叉树, 中序: ");
   InOrder( root2 );
   printf(", 后序: ");
```

```
    PostOrder( root2 );
    printf("。");
    DestroyBiTree( root1 );
    DestroyBiTree( root2 );
    return 0;
}
void InitBiTree( pBiTree *pRoot ) {
    *pRoot = NULL;
    *pRoot = ( pBiTree )malloc( sizeof( BTNode ) );
}
void CreateBiTree_semi( pBiTree *pRoot ) {
    InitBiTree( pRoot );
    char ch;
    scanf("%c", &ch);
    if( ch == '^')
        ( *pRoot ) = NULL;
    else{
        ( *pRoot ) = ( pBiTree )malloc( sizeof( BTNode ) );
        ( *pRoot )->data = ch;
        CreateBiTree_semi( &( *pRoot )->lchild );
        CreateBiTree_semi( &( *pRoot )->rchild );
    }
}
pBiTree CreateBiTree_full( int n, int h ) {
    InitBiTree( &root2 );
    int t;
    char ch = 65;
    root2->data = ch;
    root2->lchild = root2->rchild = NULL;
    srand( time( NULL ) );
    for(int i = 0; i <= n - 2 ; i++) {
        t = rand() % 2;
        if( t == 0 ) {
            root2->lchild = InsertLeftNode( root2->lchild, ++ch );
        }
        else {
            root2->rchild = InsertRightNode( root2->rchild, ++ch );
        }
    }
}
pBiTree InsertLeftNode( pBiTree pCurNode, DataType x) {
    if( pCurNode == NULL ) {
        pCurNode = ( pBiTree )malloc( sizeof( BTNode ) );
        pCurNode->data = x;
        pCurNode->lchild = pCurNode->rchild = NULL;
```

```
    }
    else {
        pCurNode->lchild = InsertLeftNode( pCurNode->lchild, x );
    }
    return pCurNode;
    if( !( InsertLeftNode( pCurNode, x)))
        return NULL;
}
pBiTree InsertRightNode( pBiTree pCurNode, DataType x) {
    if( pCurNode == NULL ) {
        pCurNode = ( pBiTree )malloc( sizeof( BTNode ) );
        pCurNode->data = x;
        pCurNode->lchild = pCurNode->rchild = NULL;
    }
    else {
        pCurNode->rchild = InsertRightNode( pCurNode->rchild, x );
    }
    return pCurNode;
    if( !( InsertRightNode( pCurNode, x)))
        return NULL;
}
int GetDepth(pBiTree root ) {
    int ld,rd;
    if( root == NULL )
        return 0;
    else {
        ld = GetDepth( root->lchild );
        rd = GetDepth( root->rchild );
    }
    return ld > rd ? ld + 1 : rd + 1;
}
void PreOrder( pBiTree root) {
    if( root != NULL ) {
        VisitNode( root );
        PreOrder( root->lchild );
        PreOrder( root->rchild );
    }
}
void InOrder( pBiTree root ) {
    if( root != NULL ) {
        InOrder( root->lchild );
        VisitNode( root );
        InOrder( root->rchild );
    }
}
```

```c
void PostOrder( pBiTree root ) {
   if( root != NULL ) {
      PostOrder( root->rchild );
      PostOrder( root->lchild );
      VisitNode( root );
   }
}
void VisitNode( pBiTree root ) {
   printf( "%c", root->data );
}
void DestroyBiTree( pBiTree root ) {
   if( root == NULL )
      return;
   DestroyBiTree( root->lchild );
   DestroyBiTree( root->rchild );
   free( root );
}
```