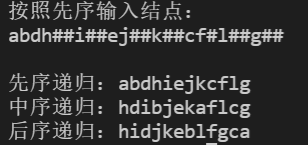
**测试用例**：abdh##i##ej##k##cf#l##g##

**运行结果**



**算法和数据结构设计**

1. 利用递归创建二叉树，按照先序输入结点，建立一个链表，用b来指向当前结点，每个结点由当前结点携带信息data，指向左子树的指针lson，右子树的指针rson，三者组成。如果当前输入为#，b设为空指针；若非#，剩下结点交付给b的左子树，直到某结点的左右子树都是空，即树的叶子，再返回到根结点，进入右子树，直到树的叶子。返回树的根结点
2. 用递归的先序遍历。输出根结点，再遍历左子树和右子树。中序，遍历左子树后，输出根节点，再遍历右子树。后序，遍历完左右子树后再输出根节点。
3. 非递归的遍历。先序，利用一个栈，先压入根节点，当栈不为空，输出栈顶元素，弹出栈顶，如果搜素指针当前指向的不为空，若左子树不为空，压栈，右子树不为空，压栈。如果搜索指针为空，弹出栈顶，交付给b。重复以上过程，直到栈为空。
4. 非递归的中序遍历。当搜索指针不为空，一直向其左子树方向前进。当其为空，弹出栈顶，输出，并赋值给b。当b的右子树不为空，弹出栈顶，并赋值给b。重复，直到b的左右子树都为空，并且栈为空。
5. 非递归的后序遍历。设立flag，表明是第一次或第二次访问该结点。压入根节点，当栈不为空，判断，若搜索指针b不为空，且不等于栈顶（即不是访问过左子树后回到该结点），该结点flag 赋值为0（第一次访问），压栈，b指向b的左子树；若不满足上述条件，且栈顶结点是第一次访问，将栈顶结点设为第二次访问，b指向栈顶的右子树。若依然不满足上述条件（即左右子树都访问过了）弹出栈顶，输出，b指向此时的栈顶结点。重复，直到栈为空。

#include <stdio.h>

#include <stdlib.h>

#define LEN sizeof(struct node)

#define NodeNum 50   //结点最大数量

typedef struct node {

    char data;

    struct node \*lson, \*rson;

}BTnode;

BTnode \*CreateBT();

void PreOrder1(BTnode \*b, char pre\_1[]); //非递归，先序

void InOrder1(BTnode \*b, char in\_1[]); //非递归，中序

void PostOrder1(BTnode \*b, char post\_1[]); //非递归，后序

void PreOrderRecursion(BTnode \*b); //递归，先序

void InOrderRecursion(BTnode \*b); //递归，中序

void PostOrderRecursion(BTnode \*b); //递归，后序

int Push(BTnode \*sta[], BTnode \*b, int top);

BTnode \*Pop(BTnode \*sta[], int top);

int main() {

    BTnode \*root = NULL; //root 是根节点

    int i = 0;

    char pre\_1[NodeNum], in\_1[NodeNum], post\_1[NodeNum];  //先序、中序、后序非递归遍历结果

    printf("按照先序输入结点：\n"); //结点总数不超过 NodeNum，#表明该结点为空

    root = CreateBT();

    //非递归

    PreOrder1(root, pre\_1);

    InOrder1(root, in\_1);

    PostOrder1(root, post\_1);

    //递归

    printf("\n先序递归：");

    PreOrderRecursion(root);

    printf("\n中序递归：");

    InOrderRecursion(root);

    printf("\n后序递归：");

    PostOrderRecursion(root);

    return 0;

}

void PostOrderRecursion(BTnode \*b) { //递归，后序

    if(b != NULL) {

        PostOrderRecursion(b->lson);

        PostOrderRecursion(b->rson);

        printf("%c", b->data);

    }

}

void InOrderRecursion(BTnode \*b) { //递归，中序

    if(b != NULL) {

        InOrderRecursion(b->lson);

        printf("%c", b->data);

        InOrderRecursion(b->rson);

    }

}

void PreOrderRecursion(BTnode \*b) { //递归，先序

    if (b != NULL) {

        printf("%c", b->data);

        PreOrderRecursion(b->lson);

        PreOrderRecursion(b->rson);

    }

}

void PostOrder1(BTnode \*b, char post\_1[]) {    //非递归，后序

    if(b == NULL)

        exit(-1);

    BTnode \*sta[NodeNum];

    int flag[NodeNum] = {0}; //flag == 1 第二次访问

    int sta\_top = 0, i; //栈顶

    for (i = 0; i < NodeNum; i++) {

        sta[i] = NULL;

    }

    sta\_top = Push(sta, b, sta\_top); //压入根结点

    b = b->lson;

    i = 0;

    while(sta\_top > 0) {

        if(b != NULL && b != sta[sta\_top - 1]) {

            flag[sta\_top] = 0;

            sta\_top = Push(sta, b, sta\_top);

            b = b->lson;

        }

        else if(flag[sta\_top - 1] == 0) {

            flag[sta\_top - 1] = 1;

            b = sta[sta\_top - 1]->rson;

        }

        else{

            post\_1[i] = sta[sta\_top - 1]->data, i++;

            Pop(sta, sta\_top), sta\_top--;

            b = sta[sta\_top - 1];

        }

    }

}

void InOrder1(BTnode \*b, char in\_1[]) {   //一个栈 依次记录左子树，非递归

    if(b == NULL)

        exit(-1);

    BTnode \*sta[NodeNum];  //存左子树

    int sta\_top = 0, i; //栈顶

    for (i = 0; i < NodeNum; i++) {

        sta[i] = NULL;

    }

    i = 0;

    while(sta\_top > -1) {

        while(b != NULL) {

            sta\_top = Push(sta, b, sta\_top);

            b = b->lson;

        }

        in\_1[i] = sta[sta\_top - 1]->data, i++; //弹出左结点

        b = Pop(sta, sta\_top), sta\_top--;

        while(b->rson == NULL) {

            in\_1[i] = sta[sta\_top - 1]->data, i++;

            b = Pop(sta, sta\_top), sta\_top--;

        }

        b = b->rson;

        if(sta\_top == 0 && b->lson == NULL && b->rson == NULL) {   //遍历到最后一个结点

            in\_1[i] = b->data;

            break;

        }

    }

}

void PreOrder1(BTnode \*b, char pre\_1[]) { //一个堆栈，先存根结点最开始，再右节点，最后左节点，

    if(b == NULL)

        exit(-1);

    BTnode \*sta[NodeNum];  //存储右子树的栈

    int sta\_top = 0, i; //sta\_top 是栈顶

    for(i = 0; i < NodeNum; i++) {

        sta[i] = NULL;

    }

    sta\_top = Push(sta, b, sta\_top);  //压入根结点

    i = 0;

    while(sta\_top > 0) {

        pre\_1[i] = sta[sta\_top - 1]->data;

        i++;

        sta\_top--;

        if(b != NULL) {

            if(b->rson != NULL)

                sta\_top = Push(sta, b->rson, sta\_top);

            if(b->lson != NULL)

                sta\_top = Push(sta, b->lson, sta\_top);

            b = b->lson;

        }

        else {

            b = Pop(sta, sta\_top);

        }

    }

}

int Push(BTnode \*sta[], BTnode \*b, int top) {

    sta[top] = (BTnode \*)malloc(LEN);

    sta[top]->data = b->data;

    sta[top]->lson = b->lson;

    sta[top]->rson = b->rson;

    top++;

    return top;

}

BTnode \*Pop(BTnode \*sta[], int top) {

    BTnode \*n;

    n = (BTnode \*)malloc(LEN);

    top--;

    n->data = sta[top]->data;

    n->lson = sta[top]->lson;

    n->rson = sta[top]->rson;

    free(sta[top]);

    return n;

}

BTnode \*CreateBT() {

  char inf;

  BTnode \*b;

  scanf("%c", &inf);

  if (inf == '#') {

    b = NULL;

  } else {

    b = (BTnode \*)malloc(LEN);

    b->data = inf;

    b->lson = CreateBT();

    b->rson = CreateBT();

  }

  return b;

}