实验三 二叉树遍历算法

一、 实验目的

1. 进一步理解掌握二叉树二叉链表存储结构。
2. 掌握二叉树遍历的递归与非递归算法。

二、 实验要求

1. 认真阅读和掌握（先序、中序、后序和层次）遍历的递归与非递归算法。
2. 上机调试（先序、中序、后序和层次）遍历的递归与非递归算法。
3. 保存和打印出程序的运行结果，并结合程序进行分析。
4. 上机后，认真整理源程序及其注释，完成实验报告（包括源程序、实验结果、算法分析、实验总结等）。

三、 实验内容

先序、中序、后序、遍历的递归与非递归算法和层次遍历的算法实现

四、 源程序：

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define maxsize 100
typedef char datatype;
int count=0;//全局变量
typedef struct tnode{//二叉树定义，递归遍历
    datatype data;
    struct tnode *lchild;
    struct tnode *rchild;
}tnode,*bintree;
typedef struct snode{//顺序栈定义,非递归遍历
    bintree data[maxsize];
    int top;
}seqstack,*pseqstack;
typedef struct qnode{//顺序队列定义，层次遍历
    bintree data[maxsize];
    int front;
    int rear;
}seqqueue,*pseqqueue;
//*******二叉树部分********//
bintree create(void)
{//构建二叉树
    bintree t;
    datatype ch;
    ch=getchar();
    if(ch=='#')
        t=NULL;
    else{
        t=(bintree)malloc(sizeof(tnode));
        t->data=ch;
        t->lchild=create();
        t->rchild=create();
    }
    return t;
}
//*******顺序栈部分********//
pseqstack init(void)
{//创建顺序栈
    pseqstack s;
    s=(pseqstack)malloc(sizeof(seqstack));
    if(s)
```

```
        s->top=-1;
    return s;
}
bool empty(pseqstack s)
{//判断栈是否为空
    if(s->top==-1)
        return 1;
    else
        return 0;
}
bool push(pseqstack s,bintree x)
{//栈顶插入新元素x
    if(s->top==maxsize-1)
        return 0;//栈满无法入栈
    else{
        s->top++;
        s->data[s->top]=x;
        return 1;
    }
}
bool pop(pseqstack s,bintree *x)
{//删除栈顶元素，并保存在*x
    if(empty(s))
        return 0;//栈空不能出栈
    else{
        *x=s->data[s->top];
        s->top--;
        return 1;
    }
}
//*******队列部分********//
pseqqueue initqueue(void)
{//初始化队列
    pseqqueue q;
    if (!(q=(pseqqueue)malloc(sizeof(seqqueue)))){
        printf("内存分配失败！");
        exit(-1);
    }
    q->front=q->rear=-1;
    return q;
}
bool emptyqueue(pseqqueue q)
{//队列判空
    if(q->front==q->rear)
        return 1;//队空
    return 0;
}
bool inqueue(pseqqueue q,bintree t)
{//入队
    if(q->rear==maxsize-1)
        return 0;//队满
    q->rear++;
    q->data[q->rear]=t;
    return 1;
}
bool outqueue(pseqqueue q,bintree *t)
{//出队
    if(q->front==q->rear)
        return 0;
    q->front++;
    *t=q->data[q->front];
    return 1;
}
//*******遍历部分********//
void perorder1(bintree t)
{//先序遍历的递归算法
    if(t){
        printf("%c ",t->data);
        perorder1(t->lchild);
        perorder1(t->rchild);
    }
}
void inorder1(bintree t)
{//中序遍历的递归算法
    if(t){
        inorder1(t->lchild);
        printf("%c ",t->data);
        inorder1(t->rchild);
    }
}
void postorder1(bintree t)
{//后序遍历的递归算法
    if(t){
        postorder1(t->lchild);
        postorder1(t->rchild);
        printf("%c ",t->data);
    }
}
```

```
void preorder2(bintree t)
{//栈的先序遍历的非递归算法
    pseqstack s;
    bintree p=t;
    s=init();
    while (p||!empty(s))
    {
        if(p){
            printf("%c ",p->data);
            push(s,p);
            p=p->lchild;
        }
        else{
            pop(s,&p);
            p=p->rchild;
        }
    }
}
void inorder2(bintree t)
{//栈的中序遍历的非递归算法
    pseqstack s;
    bintree p=t;
    s=init();
    while (p||!empty(s)){
        if(p){
            push(s,p);
            p=p->lchild;
        }
        else{
            pop(s,&p);
            printf("%c ",p->data);
            p=p->rchild;
        }
    }
}
void postorder2(bintree t)
{//栈的后序遍历的非递归算法
    pseqstack s1;//最终结果栈
    pseqstack s2;//辅助栈
    bintree p=t;
    s1=init();
    s2=init();
    while (p||!empty(s2)){
        if(p){
            push(s1,p);
            push(s2,p);
            p=p->rchild;
        }
        else{
            pop(s2,&p);
            p=p->lchild;
        }
    }
    while (!empty(s1))
    {
        pop(s1,&p);
        printf("%c ",p->data);
    }
}
void preorder3(bintree t)
{//先序非递归-“栈”
    bintree p=t;
    bintree s[maxsize];
    int top=-1;
    do{
        while(p){
            printf("%c ",p->data);
            top++;
            s[top]=p;//保留当前 p 结点在栈 s 中
            p=p->lchild;//转到当前 p 结点的左孩子
        }
        p=s[top];//取 p 的上一个结点
        top--;//“头指针”减 1
        p=p->rchild;//转到当前 p 结点的右孩子
    }while(top!=-1||p);
}
void inorder3(bintree t)
{//中序非递归-“栈”
    bintree p=t;
    bintree s[maxsize];
    int top=-1;
    do{
        while (p){
            top++;
            s[top]=p;//保留当前 p 结点在栈 s 中
            p=p->lchild;//转到当前 p 结点的左孩子
        }
        p=s[top];//取 p 的上一个结点
```

```
        top--;//“头指针”减 1
        printf("%c ",p->data);
        p=p->rchild;
    }while(top!=-1||p);
}
void postorder3(bintree t)
{//后序非递归-“栈”
    bintree p=t;
    bintree s1[maxsize];//最终结果栈
    bintree s2[maxsize];//辅助栈
    int top1=-1;
    int top2=-1;
    do{
        while (p){
            top1++;
            top2++;
            s1[top1]=p;//保留当前 p 结点在栈 s1 中
            s2[top2]=p;//保留当前 p 结点在栈 s2 中
            p=p->rchild;//转到当前 p 结点的左孩子
        }
        p=s2[top2];//取 p 的上一个结点
        top2--;//“头指针”减 1
        p=p->lchild;//转到当前 p 结点的左孩子
    }while(top2!=-1||p);
    while (top1!=-1){
        p=s1[top1];//s1 从后往前输出 p 结点
        printf("%c ",p->data);
        top1--;
    }
}
void levelorder(bintree t)
{//层次遍历
    pseqqueue q;
    q=initqueue();//创建队列
    if(t!= NULL)
        inqueue(q,t);//将结点 t 存入队列中
    while (!emptyqueue(q)){
        outqueue(q,&t);// 出队时的节点
        printf("%c ",t->data);// 输出节点存储的值
        if(t->lchild!= NULL)//有左孩子时将该节点进队
            inqueue(q,t->lchild);
        if(t->rchild != NULL)//有右孩子时将该节点进队
            inqueue(q,t->rchild);
    }
}
int main()
{
    //ABDH###E##CF##G##
    //1248###5##36##7##
    //1248##9##50###36##7##
    bintree t;
    printf("请输入字符串:");
    t=create();
    printf("\n先序递归: ");
    perorder1(t);
    printf("\n中序递归: ");
    inorder1(t);
    printf("\n后序递归: ");
    postorder1(t);
    printf("\n先序非递归-栈: ");
    preorder2(t);
    printf("\n中序非递归-栈: ");
    inorder2(t);
    printf("\n后序非递归-栈: ");
    postorder2(t);
    printf("\n先序非递归-“栈”: ");
    preorder3(t);
    printf("\n中序非递归-“栈”: ");
    inorder3(t);
    printf("\n后序非递归-“栈”: ");
    postorder3(t);
    printf("\n层次遍历-队列: ");
    levelorder(t);
    printf("\n");
}
```

五、运行结果

```
请输入字符串:1248##9##50###36##7##

先序递归: 1 2 4 8 9 5 0 3 6 7
中序递归: 8 4 9 2 0 5 1 6 3 7
后序递归: 8 9 4 0 5 2 6 7 3 1
先序非递归-栈: 1 2 4 8 9 5 0 3 6 7
中序非递归-栈: 8 4 9 2 0 5 1 6 3 7
后序非递归-栈: 8 9 4 0 5 2 6 7 3 1
先序非递归-"栈": 1 2 4 8 9 5 0 3 6 7
中序非递归-"栈": 8 4 9 2 0 5 1 6 3 7
后序非递归-"栈": 8 9 4 0 5 2 6 7 3 1
层次遍历-队列: 1 2 3 4 5 6 7 8 9 0
Program ended with exit code: 0
```

```
请输入字符串:ABDH###E##CF##G##

先序递归: A B D H E C F G
中序递归: H D B E A F C G
后序递归: H D E B F G C A
先序非递归-栈: A B D H E C F G
中序非递归-栈: H D B E A F C G
后序非递归-栈: H D E B F G C A
先序非递归-"栈": A B D H E C F G
中序非递归-"栈": H D B E A F C G
后序非递归-"栈": H D E B F G C A
层次遍历-队列: A B C D E F G H
Program ended with exit code: 0
```

六、实验总结

递归算法:

1. 先（根）序遍历的递归算法定义:
若二叉树非空，则依次执行如下操作:
(1) 访问根结点
(2) 遍历左子树
(3) 遍历右子树

```
121 void perorder1(bintree t)
122 {//先序遍历的递归算法
123     if(t)
124     {
125         printf("%c ",t->data);
126         perorder1(t->lchild);
127         perorder1(t->rchild);
128     }
129 }
```

2. 中（根）序遍历的递归算法定义:
若二叉树非空，则依次执行如下操作:
(1)遍历左子树;
(2)访问根结点;
(3)遍历右子树。

```
130 void inorder1(bintree t)
131 {//中序遍历的递归算法
132     if(t)
133     {
134         inorder1(t->lchild);
135         printf("%c ",t->data);
136         inorder1(t->rchild);
137     }
138 }
```

3. 后（根）序遍历得递归算法定义:
若二叉树非空，则依次执行如下操作:
(1)遍历左子树;
(2)遍历右子树;
(3)访问根结点。

```
139 void postorder1(bintree t)
140 {//后序遍历的递归算法
141     if(t)
142     {
143         postorder1(t->lchild);
144         postorder1(t->rchild);
145         printf("%c ",t->data);
146     }
147 }
```

非递归算法:

1. 先序:

1. 首先用 rear 标记树的根(root)，当 rear 非空的时候；
2. 就直接打印根，并且将 rear(也就是 root)入栈；
3. 接着遍历根的左子树，一直遍历到最左边；
4. 当循环出来后，最左边的节点就已经入栈了；
5. 然后将栈此时非空(这就是最外面循环的一个条件)，就弹出栈顶元素，并且用 rear 标记弹出元素的右节点；
6. 所以如果右节点非空的时候还要将右边节点打印并且入栈，所以最外面还要增加一个条件，让右节点也被打印，就是 rear 不为空；

2. 中序：

跟前序一样的原理，只是是先找到最左边的节点，打印了左边节点，才是打印根，然后右边节点；

3. 后序：

1. 后序遍历同样是通过 rear 找到将节点一个一个入栈，一直到最后面一个左节点，然后出栈，将左边节点打印；
2. 然后要判断是否存在右边节点，如果右节点为空，直接将根打印；
3. 但是会存在错误，就是会，因此需要将打印的节点用 front 标记下来，以防后面，出现后面反复打印一个节点的情况
4. 如果你不标记已经打印过的节点，当已经打印过去’H’这个节点，但是当你本来需要打印‘E’，然后发现它的右边节点还是非空，又会将’H’入栈，因此需要标记，进入 while 语句打印根节点。

```
void preorder2(bintree t)
{//栈的先序遍历的非递归算法
    pseqstack s;
    bintree p=t;
    s=init();
    while (p||!empty(s))
    {
        if(p)
        {
            printf("%c ",p->data);
            push(s,p);
            p=p->lchild;
        }
        else
        {
            pop(s,&p);
            p=p->rchild;
        }
    }
}
void inorder2(bintree t)
{//栈的中序遍历的非递归算法
    pseqstack s;
    bintree p=t;
    s=init();
    while (p||!empty(s))
    {
        if(p)
        {
            push(s,p);
            p=p->lchild;
        }
        else
        {
            pop(s,&p);
            printf("%c ",p->data);
            p=p->rchild;
        }
    }
}
void postorder2(bintree t)
{//栈的后序遍历的非递归算法
    pseqstack s1;//最终结果栈
    pseqstack s2;//辅助栈
    bintree p=t;
    s1=init();
    s2=init();
    while (p||!empty(s2))
    {
        if(p)
        {
            push(s1,p);
            push(s2,p);
            p=p->rchild;
        }
        else
        {
            pop(s2,&p);
            p=p->lchild;
        }
    }
    while (!empty(s1))
    {
        pop(s1,&p);
        printf("%c ",p->data);
    }
}
```

层次遍历：

按照二叉树中的层次从左到右依次遍历每层中的结点。具体的实现思路是：通过使用队列的数据结构，从树的根结点开始，依次将其左孩子和右孩子入队。而后每次队列中一个结点出队，都将其左孩子和右孩子入队，直到树中所有结点都出队，出队结点的先后顺序就是层次遍历的最终结果。

```
279 void levelorder(bintree t)
280 {//层次遍历
281     pseqqueue q;
282     q=initqueue();//创建队列
283     if(t!= NULL)
284         inqueue(q,t);//将结点 t 存入队列中
285     while (!emptyqueue(q))
286     {
287         outqueue(q,&t);// 出队时的节点
288         printf("%c ",t->data);// 输出节点存储的值
289         if(t->lchild!= NULL)//有左孩子时将该节点进队列
290             inqueue(q,t->lchild);
291         if(t->rchild != NULL)//有右孩子时将该节点进队列
292             inqueue(q,t->rchild);
293     }
294 }
```