数 据 结 构 实 验 报 告 成绩_____

学号	1307402068	姓名	李世旺	授课教师	黄 欣
专业	信息与计算科学		实验报告递交日期		2015.05.19

实验题目

二叉树的存储、遍历、恢复和删除等。

一． 需求分析

1．程序实现的功能：二叉树的存储、遍历、恢复和删除等。

2. 编制函数：

1). 输入完全二叉树层次序列，建立二叉链表

2). 递归求高

3).非递归前序遍历二叉树，写入文件

4).递归中序遍历二叉树，写入文件

5).读文件，前序序列、中序序列存入数组

6).根据前序中序恢复二叉链表

7).再次遍历二叉链表

8).仿造非递归删除二叉链表。

3．数据输入的内容、输入形式与范围

以字符串形式输入完全二叉树层次序列，’@’补空，以’#’加回车符结束。

4．数据输出的内容与形式

显示前序序列、中序序列时，元素间有空格。

二. 主要算法的算法思想.

1．建立二叉链表的函数：

利用队列，保证先输入的结点孩子比后输入的结点孩子先进入队列。

2. 求树的高度函数：

a.任意一棵树的高度都是其子树高度较大者加一。

b.空树的高度为 0。

c.利用递归实现。

3. 非递归前序遍历函数：（栈实现）

a.访问节点，

b.右子树根结点进栈。

c.左子树根结点进栈。

4. 恢复二叉链表函数:

a.每次都可以获得左、右子树的前序和中序序列。

b.前序和中序序列为空的树一定是空树。

c.递归实现

5.仿造非递归前序遍历，删除树的函数：

删左子树，删右子树，释放根结点，递归实现。

三. 设计：

1．二叉树存储结构：二叉链表。

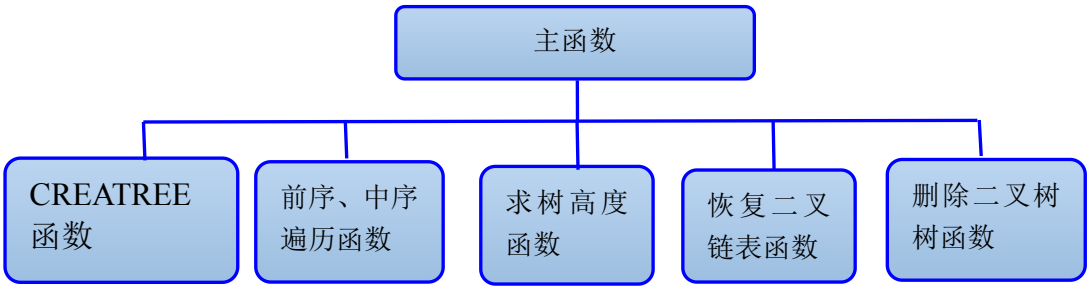
结点类型定义：

```
typedef struct node
{     datatype data;
      struct node *lchild,*rchild;
}bitree;/*二叉树结点类型*/
```

2．参数表（列出所有的符号常量与全局变量）

参数名	数据传递方式	数据内容	传递	所属函数
NULL	符号常量	空指针	0	所有函数
maxsize	符号常量	100	100	所有函数

3．函数间的调用关系图



4．列出每个函数的函数声明、函数作用、函数值、形参内容与形式、主要算法步骤等

1)．建立二叉链表函数

函数首部：bitree *CREATREE(void)

形参：无

函数作用：建立二叉链表

函数值：返回树的根指针。

局部变量：datatype ch;/*接收字符*/

int front,rear;/*队头队尾*/

bitree *root,*s;/*树根和新结点*/

bitree *Q[maxsize];/*队列*/

算法主要步骤：

(a)结束输入条件：　　ch=getchar();while(ch!='#'){...}

(b)入队和生成新结点　　if(ch!='@'){s=malloc(sizeof(bitree));
s->data=ch;s->lchild=NULL;s->rchild=NULL;}rear++;

(c)判断是左子还是右子　　if(rear%2==0)Q[front]->lchild=s;
else　Q[front]->rchild=s;

(d)出队条件：　　if(rear%2==1)front++;

2)．递归中序遍历函数

函数首部：void in_order(bitree *t,FILE *fp)

形参：t: 二叉树类型指针；fp:文件指针；

函数作用：中序遍历，写入文件。
函数值：无
局部变量　无
算法主要步骤：
(a)递归。if(t){
in_order(t->lchild,fp);
printf(" %c",t->data);fprintf(fp," %c",t->data);
in_order(t->rchild,fp);}

3). 非递归前序遍历函数 （栈实现）
函数首部：void pre_order(bitree *t,FILE *fp)
形参：t: 二叉树类型指针；fp:文件指针；
函数作用：前序遍历，写入文件。
函数值：无
局部变量：int top;/*栈顶*/
bitree *s;/*临时指针*/
bitree *Q[maxsize];/*栈空间*/
算法主要步骤：
(a) /*根节点进栈*/top=0;Q[top]=t;
(b)/*判空*/if(!t)return;
(c)/*循环步骤如下*/
do
{ fprintf(fp," %c",Q[top]->data); /*访问节点*/
printf(" %c",Q[top]->data);s=Q[top];top--; /*出栈*/
if(s->rchild){top++;Q[top]=s->rchild;} /*右子进栈*/
if(s->lchild){top++;Q[top]=s->lchild;} /*左子进栈*/
}while(top>=0);
4）. 比较大小函数：
函数首部：int big(int a,int b)
形参：a：一个数　b：另一个数
函数作用：选出大的数
函数值：较大的数
局部变量：无

算法主要步骤：
{if(a>=b)return a;
return b;
}/*比较大小*/
5）. 求树的高度函数：
函数首部：int depth_of_tree(bitree *t)
形参：t：二叉树类型指针
函数作用：求树高
函数值：树高

局部变量：h：存放树高

算法主要步骤：
(a)判空　　int h;if(!t)h=0;
(b)递归 else {h=big(depth_of_tree(t->lchild),depth_of_tree(t->rchild))+1;}return h;
6）. 恢复二叉树的函数：
函数首部：bitree *restore(int start1,int start2,int len0,char *preod,char *inod)
形参：start1：前序序列起始下标,start2：中序序列起始下标,len0：当前序列长度
preod：放着前序序列的数组 inod：放着中序序列的数组
函数作用：恢复二叉树
函数值：树根指针
局部变量：

```
bitree *p;          /*存放新建结点*/
int loc,            /*目的节点下标*/
len1,len2,      /*左右分支元素个数*/
r1,r2,  /*左分支前序中序起始下标*/
s1,s2;  /*右分支前序中序起始下标*/
```

算法主要步骤：
(a)递归结束标志　if(len0<=0)return NULL;
(b)新建结点　p=malloc(sizeof(bitree));
　　p->data=preod[start1];
(c)寻找结点下标：loc=0;while((inod[loc]!=p->data))loc++;
(d)预置形参：r1=start1+1;r2=start2;len1=loc-start2;
　　s1=start1+len1+1;s2=loc+1;len2=len0-len1-1;
(e)递归　p->lchild=restore(r1,r2,len1,preod,inod);
　　p->rchild=restore(s1,s2,len2,preod,inod);
　　return p;
7）. 删除二叉链表函数：
　　函数首部：void delete_tree(bitree *t)
　　形参：t: 二叉树类型指针
　　函数作用：删除二叉链表
　　函数值：无
　　局部变量：无
　　算法主要步骤：
(a)判空　　if(!t)return;
(b)递归　　delete_tree(t->lchild);delete_tree(t->rchild);free(t);

四. 调试分析：

1．调试中出现的问题，解决的办法
　1）.‘->’漏成‘-’时，不会被明确指出。
　2）. 含有//的行出现错误：改为/**/格式
　3）. 出现大量错误：形如 typedef char datatype 加上分号
2．每个函数的时、空复杂性分析
　1）. 建立二叉链表函数 bitree *CREATREE(void)
　　T(n)=O(n)，　S(n)=O(n);

2）. 递归中序遍历函数 void in_order(bitree *t,FILE *fp)

T(n)=O(n)， S(n)=O(1);

3). 非递归前序遍历函数 （栈实现）void pre_order(bitree *t,FILE *fp)

T(n)=O(n)， S(n)=O(n);

4). 比较大小函数：int big(int a,int b)

T(n)=O(1)， S(n)=O(1);

5).求树的高度函数：int depth_of_tree(bitree *t)

T(n)=O(n)， S(n)=O(1);

6).恢复二叉树的函数：bitree *restore(int start1,int start2,int len0,char *preod,char *inod)

T(n)=O(n)， S(n)=O(n);

7).删除二叉链表函数：void delete_tree(bitree *t)

T(n)=O(n)， S(n)=O(n);

8). 主函数 main()

T(n)=O(n) ， S(n)=O(n).

3．改进设想，经验体会

有很多可以改进的地方，比如队列可以使用循环队列等

五. 使用说明：如何使用你编制的程序、操作步骤.

编译程序成功后，输入完全二叉树层次序列，’@’补空，以’#’加回车符结束。

六. 测试结果：

输入输出数据内容：

窗口显示如下：(下划线部分为输入部分，其余为输出部分)

测试数据一：

@ for NULL, # for end

ab@c@@@d#↙

pre_order: a b c d

in_order: d c b a

depth is 4

check preod[]: a b c d

check inod[]: d c b a

again check pre_order: a b c d

again check in_order: d c b a

测试数据二：

@ for NULL, # for end

#↙

empty tree!

pre_order:

in_order:

depth is 0

check preod[]:

check inod[]:

again check pre_order:

again check in_order:

七．源代码清单

```
#include"stdio.h"
#include"conio.h"
#include"stdlib.h"
#define maxsize 100
typedef char datatype;
typedef struct node
{    datatype data;
     struct node *lchild,*rchild;
}bitree;
bitree *CREATREE(void)
{
     datatype ch;            /*接收字符*/
     int front,rear;         /*队头队尾*/
     bitree *root,*s; /*树根和新结点*/
     bitree *Q[maxsize];       /*队列*/
     root=NULL;
     front=1;rear=0;
     ch=getchar();
     while(ch!='#')
     {
          s=NULL;
          if(ch!='@')
          {
               s=malloc(sizeof(bitree));
               s->data=ch;s->lchild=NULL;s->rchild=NULL;
          }
          rear++;Q[rear]=s;
          if(rear==1)root=s;
          else
          {
               if(s&&Q[front])
                    {     if(rear%2==0)Q[front]->lchild=s;
                          else      Q[front]->rchild=s;
                      }
               if(rear%2==1)front++;
          }
          ch=getchar();
     }return root;
}/*CREATREE*/
void in_order(bitree *t,FILE *fp)
{
     if(t)
     {
```

```
            in_order(t->lchild,fp);
            printf(" %c",t->data);fprintf(fp," %c",t->data);
            in_order(t->rchild,fp);
      }
}/*in_order*/

void pre_order(bitree *t,FILE *fp)
{     int top;                          /*栈顶*/
       bitree *s;                    /*临时指针*/
      bitree *Q[maxsize];          /*栈空间*/
      top=0;Q[top]=t;        /*根节点进栈*/
      if(!t)return;

do
{     fprintf(fp," %c",Q[top]->data);                  /*访问节点*/
      printf(" %c",Q[top]->data);s=Q[top];top--; /*出栈*/
       if(s->rchild){top++;Q[top]=s->rchild;}      /*右子进栈*/
       if(s->lchild){top++;Q[top]=s->lchild;}      /*左子进栈*/
}while(top>=0);
}
int big(int a,int b)
{
       if(a>=b)return a;
       return b;
}/*比较大小*/

int depth_of_tree(bitree *t)
{
      int h;
      if(!t)h=0;
      else {h=big(depth_of_tree(t->lchild),depth_of_tree(t->rchild))+1;}
      return h;
}/*递归求高*/

bitree *restore(int start1,int start2,int len0,char *preod,char *inod)
{     bitree *p;                    /*存放新建结点*/
      int loc,                       /*目的节点下标*/
      len1,len2,              /*左右分支元素个数*/
      r1,r2,          /*左分支前序中序起始下标*/
      s1,s2;          /*右分支前序中序起始下标*/
      if(len0<=0)return NULL;
else{      p=malloc(sizeof(bitree));
           p->data=preod[start1];/*printf("\nlist:");*/
           loc=0;
```

```
            while((inod[loc]!=p->data))
             {loc++;/*printf(" %c",inod[loc]);*/}
            r1=start1+1;r2=start2;len1=loc-start2;
            s1=start1+len1+1;s2=loc+1;len2=len0-len1-1;
            p->lchild=restore(r1,r2,len1,preod,inod);
            p->rchild=restore(s1,s2,len2,preod,inod);
              return p;
}
}
void delete_tree(bitree *t)
{    if(!t)return;
      delete_tree(t->lchild);
     delete_tree(t->rchild);
     free(t);
}

main()
{
     FILE *fp;         /*文件指针（mydata.txt）*/
     char ch,                              /*单个字符*/
     preod[20],                              /*前序*/
     inod[20];                             /*中序*/
     bitree *t,*t_restore; /*生成树，恢复树*/
     int i=0;                              /*计数*/

     if((fp=fopen("mydata.txt","wt+"))==NULL)
     {printf("failed to open file !");getch();exit(1);}

     printf("@ for NULL, # for end\n");
     t=CREATREE();
     if(!t)printf("empty tree!\n");                    /*判空*/
     printf("pre_order:");
     pre_order(t,fp);fprintf(fp,"\n");              /*前序*/
      printf("\n in_order:");                        /*中序*/
     in_order(t,fp);fprintf(fp,"\n");

      printf("\n depth is %d\n",depth_of_tree(t));/*深度*/

     printf("\ncheck preod[]:");           /*恢复和检查数组*/
      rewind(fp);                              /*文件开始位置*/
      while((ch=fgetc(fp))!='\n')          /*恢复和检查数组*/
      {
            preod[i]=fgetc(fp);printf(" %c",preod[i]);i++;
      }
```

```
    i=0;printf("\ncheck inod[]:");
    while((ch=fgetc(fp))!='\n')          /*恢复和检查数组*/
    {
        inod[i]=fgetc(fp);printf(" %c",inod[i]);i++;
    }fclose(fp);

    t_restore=restore(0,0,i,preod,inod);              /*恢复*/

                              /*检查时追加遍历序列到文件*/
    if((fp=fopen("mydata.txt","at+"))==NULL)
    {printf("failed to open file !");getch();exit(1);}
    printf("\nagain check pre_order:");          /*检查*/
    pre_order(t_restore,fp);fprintf(fp,"\n");
    printf("\nagain check in_order:");           /*检查*/
    in_order(t_restore,fp);fprintf(fp,"\n");
    fclose(fp);                                 /*检查完毕*/
    delete_tree(t);                             /*删除树*/
    delete_tree(t_restore);                     /*删除树*/
    getch();                                    /*暂停*/
}
```