1 附录 C 基于二叉链表存储结构实现二叉树的源程序

/* Binary Tree On Binary LinkList Structure */

main.cpp

```
#include "def.h"
#include "SingleTree.h"
#include "Manager.h"

int main() {
    int state, op, index, e;
    char filename[30];
    BiTree tmp = nullptr;
    TElemType value;
    Manager M;
    Menu();
    cout << "enter your choice:" << endl;
    cin >> op;
    while (op){
        switch (op) {
            case 1: // 添加成员
            state = M.AddMember();
            if (state == OK){
                cout << "add successfully. The preorder series is: ";
                traverse(M.member[M.length-1].T);
                cout << endl;
                cout << "the structure now is:" << endl;
                M.DispStructure();
                cout << '\n';
            }
            else{
                cout << "false order or duplicated key!" << endl;
            }
            break;
            case 2: // 删除成员
            state = M.DelMember();
            if (state == OK) {
                cout << "delete successfully, the structure now is:" <<
                endl;
```

```
            M.DispStructure();
        }
        else cout << "empty tree!" << endl;
        break;
        case 3: // 显示树形目录
        M.DispStructure();
        break;
        case 4: // 初始化
        index = M.GetCommand1();
        if (index != -1){
            if (IsEmpty(M.member[index].T) == false){
                cout << "existed tree!" << endl;
            }
            else{
                TElemType definition[100];
                GetData(definition);
                state = CreateBiTree(M.member[index].T, definition);
                if (state == OK) {
                    cout << "create successfully! Now the structure
                          is: " << endl;
                    M.DispStructure();
                }
                else cout << "error!" << endl;
            }
        }
        break;
        case 5: // 清空二叉树
        if ((index = M.GetCommand1()) != -1){
            if (IsEmpty(M.member[index].T) == true){
                cout << "empty tree!" << endl;
            }
            else{
                state = ClearBiTree(M.member[index].T);
                if (state == OK) {
                    cout << "clear successfully! Now the structure
                          is: " << endl;
                    M.DispStructure();
                }
                else cout << "error!" << endl;
            }
        }
```

```
            break;
            case 6: // 获取树深
            if ((index = M.GetCommand1()) != -1){
                if (IsEmpty(M.member[index].T) == true){
                    cout << "empty tree!" << endl;
                }
                else{
                    cout << "The depth is: " <<
                    GetDepth(M.member[index].T) <<endl;
                }
            }
            break;
            case 7: // 查找节点
            if ((index = M.GetCommand1()) != -1){
                if (IsEmpty(M.member[index].T) == true){
                    cout << "empty tree!" << endl;
                }
                else{
                    cout << "enter the key:" << endl;
                    cin >> e;
                    tmp = find(M.member[index].T, e);
                    if (tmp){
                        cout << "result: " << tmp->data.key << "," <<
                        tmp->data.others << endl;
                    }
                    else cout << "no results." << endl;
                }
            }
            break;
            case 8: // 节点赋值
            if ((index = M.GetCommand1()) != -1){
                if (IsEmpty(M.member[index].T) == true){
                    cout << "empty tree!" << endl;
                }
                else{
                    cout << "enter the key and new key-value pair:" <<
                    endl;
                    int key;
                    cin >> key >> value.key >> value.others;
                    state = Assign(M.member[index].T, key, value);
                    if (state == -1) cout << "duplicated key!" << endl;
```

```
            else if (state == ERROR) cout << "can't find the
                 corresponding node!" << endl;
            else {
                cout << "assign successfully! The preorder is:
                     " << endl;
                PreOrder(M.member[index].T);
            }
        }
    }
    break;
    case 9: // 获取兄弟
    if ((index = M.GetCommand1()) != -1){
        if (IsEmpty(M.member[index].T) == true){
            cout << "empty tree!" << endl;
        }
        else{
            cout << "enter the key:" << endl;
            cin >> e;
            tmp = GetSibling(M.member[index].T, e);
            if (!tmp) cout << "can't find sibling!" << endl;
            else{
                cout << "Sibling is " << tmp->data.key << ","
                << tmp->data.others << endl;
            }
        }
    }
    break;
    case 10: // 插入节点
    if ((index = M.GetCommand1()) != -1){
        if (IsEmpty(M.member[index].T) == true){
            cout << "empty tree!" << endl;
        }
        else{
            cout << "enter the key:" << endl;
            cin >> e;
            cout << "enter the new node info: " << endl;
            cin >> value.key >> value.others;
            cout << "how to insert(enter -1 or 0 or 1): " <<
            endl;
            int lr;
            cin >> lr;
```

```
            state = InsertNode(M.member[index].T, e, lr, value);
            if (state == -1) cout << "duplicated key!" << endl;
            else if (state == ERROR) cout << "can't find the
                 key!" << endl;
            else{
                cout << "insert over. The preorder traverse is:
                      " << endl;
                PreOrder(M.member[index].T);
            }
        }
    }
    break;
    case 11: // 删除节点
    if ((index = M.GetCommand1()) != -1){
        if (IsEmpty(M.member[index].T) == true){
            cout << "empty tree!" << endl;
        }
        else{
            cout << "enter the key:" << endl;
            cin >> e;
            state = DeleteNode(M.member[index].T, e);
            if (state == ERROR) cout << "can't find the node!"
            << endl;
            else{
                cout << "delete over. The preorder is: " <<
                endl;
                PreOrder(M.member[index].T);
            }
        }
    }
    break;
    case 12: // 前序遍历
    if ((index = M.GetCommand1()) != -1){
        if (IsEmpty(M.member[index].T) == true){
            cout << "empty tree!" << endl;
        }
        else{
            PreOrder(M.member[index].T);
            cout << endl;
        }
    }
```

```
            break;
            case 13: // 中序遍历
            if ((index = M.GetCommand1()) != -1){
                if (IsEmpty(M.member[index].T) == true){
                    cout << "empty tree!" << endl;
                }
                else{
                    InOrder(M.member[index].T);
                    cout << endl;
                }
            }
            break;
            case 14: // 后序遍历
            if ((index = M.GetCommand1()) != -1){
                if (IsEmpty(M.member[index].T) == true){
                    cout << "empty tree!" << endl;
                }
                else{
                    PostOrder(M.member[index].T);
                    cout << endl;
                }
            }
            break;
            case 15: // 层序遍历
            if ((index = M.GetCommand1()) != -1){
                if (IsEmpty(M.member[index].T) == true){
                    cout << "empty tree!" << endl;
                }
                else{
                    LevelOrder(M.member[index].T);
                    cout << endl;
                }
            }
            break;
            case 16: // 保存文件
            if ((index = M.GetCommand1()) != -1){
                if (IsEmpty(M.member[index].T) == true){
                    cout << "empty tree!" << endl;
                }
                else{
                    cout << "enter the filename:" << endl;
```

```
            cin >> filename;
            state = SaveBiTree(M.member[index].T, filename);
            if (state == ERROR) cout << "IO Error!" << endl;
            else cout << "save successfully!" << endl;
        }
    }
    break;
    case 17: // 载入文件
    if ((index = M.GetCommand1()) != -1){
        if (IsEmpty(M.member[index].T) == false){
            cout << "tree existed!" << endl;
        }
        else{
            cout << "enter the filename:" << endl;
            cin >> filename;
            state = LoadBiTree(M.member[index].T, filename);
            if (state == ERROR) cout << "IO Error!" << endl;
            else cout << "load successfully!" << endl;
        }
    }
    break;
    case 18: // 最大路径和
    if ((index = M.GetCommand1()) != -1){
        cout << "max sum is: " << MaxPathSum(M.member[index].T)
        << endl;
    }
    break;
    case 19: // lca问题
    if ((index = M.GetCommand1()) != -1){
        cout << "enter 2 child" << endl;
        int e1, e2;
        cin >> e1 >> e2;
        tmp = LCA(M.member[index].T, e1, e2);
        cout << "the ancestor is: " << tmp->data.key << "," <<
        tmp->data.others << endl;
    }
    break;
    case 20: // 翻转二叉树
    if ((index = M.GetCommand1()) != -1){
        InvertTree(M.member[index].T);
        cout << "Invert over, the preorder is:" << endl;
```

```
                PreOrder(M.member[index].T);
            }
            break;
            default:
            cout << "wrong command!" << endl;
            break;
            case 21:
            if ((index = M.GetCommand1()) != -1){
                auto judge = IsEmpty(M.member[index].T);
                if (judge == true) cout << "empty!" << endl;
                else cout << "not empty!" << endl;
            }
            break;
        }
        Menu();
        cout << "enter your command:" << endl;
        cin >> op;
    }
    cout << "bye!" << endl;
    return 0;
}
```

def.h

```
#pragma once
#include "cstdio"
#include "cstdlib"
#include <string>
#include <iostream>
#include <queue>
#include <map>
using namespace std;

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define MaxSize 50

typedef int status;
```

```
typedef int KeyType;
typedef struct TreeElem{
    KeyType key;
    char others[30];
} TElemType; //二叉树结点类型定义


typedef struct BiTNode{ //二叉链表结点的定义
    TElemType data;
    struct BiTNode *lchild,*rchild;
    BiTNode() : lchild(nullptr), rchild(nullptr){}
} BiTNode, *BiTree;

typedef struct TreeUnit{
    string name;
    BiTree T;
    TreeUnit() : name("unnamed"), T(nullptr){}
} TU;
```

SingleTree.h

```
#pragma once
#include "def.h"

//展示菜单
void Menu();

//判断是否是空树
status IsEmpty(BiTree T);

//获取先序初始化数组
void GetData(TElemType definition[]);

// 由带空节点的前序创建二叉树
status CreateBiTree(BiTree &T,TElemType definition[]);
BiTree build(BiTree &cur, TElemType definition[], int &cnt); // 辅助函数
void traverse(BiTree T); // 先序遍历判断创建是否成功

//清空二叉树
status ClearBiTree(BiTree &T);
void clear(BiTree T); // 辅助函数
```

```
//求二叉树深度
int GetDepth(BiTree T);

//查找给定关键词的节点
BiTree find(BiTree cur, KeyType e);

//节点赋值，要求保证关键词唯一性
status Assign(BiTree &T,KeyType e,TElemType value);
int traverse(BiTree T, KeyType e, TElemType value);

//获取兄弟节点
BiTNode* GetSibling(BiTree T,KeyType e);

//插入节点
status InsertNode(BiTree &T,KeyType e,int LR,TElemType c);
int traverse1(BiTree T, KeyType e, TElemType value); // 辅助判断键唯一性

//删除节点
status DeleteNode(BiTree &T,KeyType e);
BiTree delete_(BiTree cur, KeyType e); // 删除辅助函数
int traverse2(BiTree T, KeyType e); // 判断是否有目标

//前序遍历
void PreOrder(BiTree T);

//中序遍历
void InOrder(BiTree T);

//后序遍历
void PostOrder(BiTree T);

//层序遍历
void LevelOrder(BiTree T);

//保存到文件
status SaveBiTree(BiTree T, char FileName[]);
void save_traverse(BiTree T, FILE* fp); // 保存文件辅助函数

//从文件读取
status LoadBiTree(BiTree &T, char FileName[]);
BiTree read(BiTree T, FILE* fp); //文件读取辅助函数
```

```
//最大路径和
int MaxPathSum(BiTree T);

//最近公共祖先
BiTree LCA(BiTree T, KeyType e1, KeyType e2);
int FindChild(BiTree T, KeyType e);

//翻转二叉树
void InvertTree(BiTree &T);
```

SingleTree.cpp

```
#include "SingleTree.h"

void Menu(){
   cout << "                  Menu" << endl;
   cout << "-------------------------------------------" << endl;
   cout << "    1. NewTree                2. DelTree" << endl;
   cout << "    3. DispStructure          " << endl;
   cout << endl;
   cout << "    4. CreateBiTree           5. ClearBiTree" << endl;
   cout << "    6. GetDepth               7. FindNode" << endl;
   cout << "    8. TreeNodeAssign         9. GetSibling" << endl;
   cout << "    10. InsertNode            11. DeleteNode" << endl;
   cout << "    12. PreOrder              13. InOrder" << endl;
   cout << "    14. PostOrder             15. LevelOrder" << endl;
   cout << "    16. Save                  17. Load" << endl;
   cout << "    18. MaxPathSum            19. LCA" << endl;
   cout << "    20. InvertTree            21. IsEmpty" << endl;
   cout << "    0. quit" << endl;
   cout << "-------------------------------------------" << endl;
}


status IsEmpty(BiTree T){
   if (!T) return true;
   else return false;
}


void GetData(TElemType definition[]){
```

```
    cout << "enter the key-value pair(preorder) end with key \'-1\':"
    << endl;
    int key, i = 0;
    string value;
    cin >> key;
    while (key != -1){
        definition[i].key = key;
        scanf("%s", definition[i].others);
        i++;
        cin >> key;
    }
    definition[i].key = -1;
    scanf("%s", definition[i].others);
    cout << "read successfully!" << endl;
}


status CreateBiTree(BiTree &T,TElemType definition[])
/*根据带空枝的二叉树先根遍历序列definition构造一棵二叉树，将根节点指针赋值给T
并返回OK，
如果有相同的关键字，返回ERROR.*/
{
    int hash[200] = {0};
    int i = 0;
    while (definition[i].key != -1){
        if (hash[definition[i].key] == 1 && definition[i].key != 0){
            return ERROR;
        }
        hash[definition[i].key] = 1;
        i++;
    }
    int cnt = 0;
    T = build(T, definition, cnt);
    return OK;
}

BiTree build(BiTree &cur, TElemType definition[], int& cnt){
    if (definition[cnt].key == 0){
        cnt++;
        return nullptr;
    }
```

```
    cur = (BiTree)malloc(sizeof(struct BiTNode));
    cur->data = definition[cnt++];
    cur->lchild = build(cur->lchild, definition, cnt);
    cur->rchild = build(cur->rchild, definition, cnt);
    return cur;
}

void traverse(BiTree T){
    if (T == nullptr) return;
    printf("%d,%s    ", T->data.key, T->data.others);
    traverse(T->lchild);
    traverse(T->rchild);
}

status ClearBiTree(BiTree &T)
//将二叉树设置成空，并删除所有结点，释放结点空间
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if (T == nullptr) return ERROR;
    clear(T);
    T = nullptr;
    return OK;

    /********** End **********/
}

void clear(BiTree T){
    if (T == nullptr) return;
    clear(T->lchild);
    clear(T->rchild);
    free(T);
}


int GetDepth(BiTree T){
    if (T == nullptr) return 0;
    int l = GetDepth(T->lchild) + 1;
    int r = GetDepth(T->rchild) + 1;
    if (l >= r) return l;
    else return r;
```

```
}


BiTree find(BiTree cur, KeyType e){
    if (cur == nullptr) return nullptr;
    if (cur->data.key == e) return cur;
    BiTree ansl = find(cur->lchild, e);
    BiTree ansr = find(cur->rchild, e);
    if (ansl) return ansl;
    if (ansr) return ansr;
    return nullptr;
}


status Assign(BiTree &T,KeyType e,TElemType value){
    if (traverse(T, e, value) == 0) return -1; // -1表示键重复
    BiTree ans = find(T, e);
    if (!ans) return ERROR;
    else{
        ans->data = value;
        return OK;
    }
}

int traverse(BiTree T, KeyType e, TElemType value){
    if (T == nullptr) return 1;
    if (T->data.key != e && T->data.key == value.key) return 0;
    else return traverse(T->lchild, e, value) && traverse(T->rchild, e,
    value);
}


BiTNode* GetSibling(BiTree T,KeyType e){
    if (T == nullptr) return nullptr;
    if (!(T->lchild && T->rchild)) return nullptr;
    if (T->lchild->data.key == e) return T->rchild;
    if (T->rchild->data.key == e) return T->lchild;
    BiTree l = GetSibling(T->lchild, e);
    if (l) return l;
    BiTree r = GetSibling(T->rchild, e);
    if (r) return r;
```

```
    return nullptr;
}


status InsertNode(BiTree &T,KeyType e,int LR,TElemType c){
    if (traverse1(T, e, c) == 0) return -1;
    BiTree ans = find(T, e);
    if (!ans) return ERROR;
    BiTree new_node = (BiTree)malloc(sizeof(struct BiTNode));
    if (LR == 0){
        new_node->rchild = ans->lchild;
        new_node->lchild = nullptr;
        new_node->data = c;
        ans->lchild = new_node;
    }
    else if (LR == -1){
        new_node->rchild = T;
        new_node->lchild = nullptr;
        new_node->data = c;
        T = new_node;
    }
    else{
        new_node->rchild = ans->rchild;
        new_node->lchild = nullptr;
        new_node->data = c;
        ans->rchild = new_node;
    }
    return OK;
}

int traverse1(BiTree T, KeyType e, TElemType value){
    if (T == nullptr) return 1;
    if (T->data.key == value.key) return 0;
    else return traverse(T->lchild, e, value) && traverse(T->rchild, e,
    value);
}


status DeleteNode(BiTree &T,KeyType e){
    if (!traverse2(T, e)) return ERROR;
    T = delete_(T, e);
```

```
    return OK;
}

BiTree delete_(BiTree cur, KeyType e){
    if (!cur) return nullptr;
    if (cur->data.key != e){
        cur->lchild = delete_(cur->lchild, e);
        cur->rchild = delete_(cur->rchild, e);
    }
    else{
        if (!cur->lchild && !cur->rchild){
            free(cur);
            return nullptr;
        }
        else if (!cur->lchild){
            BiTree p = cur->rchild;
            free(cur);
            return p;
        }
        else if (!cur->rchild){
            BiTree p = cur->rchild;
            free(cur);
            return p;
        }
        else{
            BiTree p = cur->lchild;
            BiTree p0 = p;
            while (p0->rchild){
                p0 = p0->rchild;
            }
            p0->rchild = cur->rchild;
            free(cur);
            return p;
        }
    }
    return cur;
}

int traverse2(BiTree T, KeyType e){
    if (T == nullptr) return 0;
    if (T->data.key == e) return 1;
```

```
    return (traverse2(T->lchild, e) || traverse2(T->rchild, e));
}


void PreOrder(BiTree T){
    if (T == nullptr) return;
    cout << T->data.key << "," << T->data.others << " " << endl;
    PreOrder(T->lchild);
    PreOrder(T->rchild);
}


void InOrder(BiTree T){
    if (T == nullptr) return;
    InOrder(T->lchild);
    cout << T->data.key << "," << T->data.others << " " << endl;
    InOrder(T->rchild);
}


void PostOrder(BiTree T){
    if (T == nullptr) return;
    PostOrder(T->lchild);
    PostOrder(T->rchild);
    cout << T->data.key << "," << T->data.others << " " << endl;
}


void LevelOrder(BiTree T){
    queue<BiTree> q;
    BiTree tmp;
    q.push(T);
    while (!q.empty()){
        tmp = q.front();
        q.pop();
        cout << tmp->data.key << "," << tmp->data.others << " " << endl;
        if (tmp->lchild) q.push(tmp->lchild);
        if (tmp->rchild) q.push(tmp->rchild);
    }
}
```

```
status SaveBiTree(BiTree T, char FileName[])
{
    FILE* fp = fopen(FileName, "w");
    if (fp){
        save_traverse(T, fp);
        fclose(fp);
        return OK;
    }
    fclose(fp);
    return ERROR;
}

void save_traverse(BiTree T, FILE* fp){
    if (!T) {
        fprintf(fp, "%d\n", 0);
        return;
    }
    fprintf(fp, "%d %s\n", T->data.key, T->data.others);
    save_traverse(T->lchild, fp);
    save_traverse(T->rchild, fp);
}


status LoadBiTree(BiTree &T, char FileName[])
{
    FILE* fp = fopen(FileName, "r");
    if (fp){
        T = read(T, fp);
        fclose(fp);
        return OK;
    }
    fclose(fp);
    return ERROR;
}

BiTree read(BiTree T, FILE* fp){
    int key;
    fscanf(fp, "%d", &key);
    if (key == 0){
        return nullptr;
```

```
    }
    else{
        T = (BiTree)malloc(sizeof(struct BiTNode));
        T->data.key = key;
        fscanf(fp, "%s", T->data.others);
        T->lchild = read(T->lchild, fp);
        T->rchild = read(T->rchild, fp);
        return T;
    }
}


int MaxPathSum(BiTree T){
    if (T == nullptr) return 0;
    int l = MaxPathSum(T->lchild);
    int r = MaxPathSum(T->rchild);
    return T->data.key + max(l, r);
}


BiTree LCA(BiTree T, KeyType e1, KeyType e2){
    int l1 = FindChild(T->lchild, e1);
    int l2 = FindChild(T->rchild, e1);
    int r1 = FindChild(T->lchild, e2);
    int r2 = FindChild(T->rchild, e2);
    if (l1 && r2 || l2 && r1) return T;
    if (T->data.key == e1 || T->data.key == e2) return T;
    if (l1 && r1) return LCA(T->lchild, e1, e2);
    else return LCA(T->rchild, e1, e2);
}

int FindChild(BiTree T, KeyType e){
    if (T == nullptr) return 0;
    if (T->data.key == e) return 1;
    return FindChild(T->lchild, e) || FindChild(T->rchild, e);
}


void InvertTree(BiTree &T){
    if (T == nullptr) return;
    BiTree tmp = T->lchild;
```

```
    T->lchild = T->rchild;
    T->rchild = tmp;
    InvertTree(T->lchild);
    InvertTree(T->rchild);
}
```

Manager.h

```
#include "def.h"
#include "SingleTree.h"

class Manager {
    private:

    public:
    TU member[MaxSize];
    int length;
    int size;
    map<string, int> name_index;

    Manager();

    // 获取树名，返回索引
    int GetCommand1();

    // 添加一棵树
    status AddMember();

    // 删除一棵树
    status DelMember();

    // 显示当前结构
    void DispStructure();

};
```

Manager.cpp

```
#include "Manager.h"

Manager::Manager() : length(0), size(MaxSize){

}
```

```
int Manager::GetCommand1(){
    string name;
    cout << "enter the name of the tree:" << endl;
    cin >> name;
    auto it = this->name_index.find(name);
    if (it == this->name_index.end()){
        cout << "can't find the tree!" << endl;
        return -1;
    }
    return it->second;
}

status Manager::AddMember() {
    if (this->length == size){
        return OVERFLOW;
    }
    string name;
    cout << "enter a name for this tree:" << endl;
    cin >> name;
    this->name_index.insert(pair<string, int>(name, this->length));
    this->member[this->length].name = name;
    TElemType definition[100];
    GetData(definition);

    int state = CreateBiTree(this->member[this->length].T, definition);
    if (state == ERROR) return ERROR;
    else{
        this->length++;
        return OK;
    }
}

status Manager::DelMember() {
    int index;
    index = this->GetCommand1();
    if (index != -1){
        int state = ClearBiTree(this->member[index].T);
        if (state == OK) {
            return OK;
        }
```

```
            else return ERROR;
        }
        return ERROR;
    }

    void Manager::DispStructure() {
        cout << "---------------------" << endl;
        cout << "|-Manager" << endl;
        for (int i = 0; i < this->length; i++){
            if (this->member[i].T == nullptr) cout << "    |-null" << endl;
            else cout << "    |-" << this->member[i].name << endl;
        }
        cout << "---------------------" << endl;
    }
```