

数据结构与算法课程设计 课程实验报告

学号：202200400053	姓名：王宇涵	班级：2202
上机学时：4	实验日期：2024-4-27	
课程设计题目： 目录树		
软件开发环境： Clion 2023.1.1		
报告内容： <p>1. 需求描述</p> <p>1.1 问题描述</p> <p>使用树结构实现一个简单文件目录系统的模拟程序。</p> <p>1.2 基本要求</p> <p>1. 设计并实现目录树 <code>CatalogTree</code> 的 ADT。</p> <p>2. 应用以上 <code>CatalogTree</code> 结构设计并实现一文件目录系统的模拟程序。</p> <p>3. 文件目录系统程序是一个不断等待用户输入命令的解释程序，根据用户输入的命令完成相关操作，直到退出（<code>quit</code>）。目录系统应支持如下基本操作：</p> <p>(1) <code>dir</code> 列出当前目录下的所有子目录与文件项。</p> <p>(2) <code>cd</code> 列出当前目录的绝对路径。</p> <p>(3) <code>cd</code> 当前目录变为当前目录的父目录。</p> <p>(4) <code>cd str</code> 当前目录变为 <code>str</code> 所表示路径的目录。</p> <p>(5) <code>mkdir str</code> 在（当前目录下）创建一个子目录（名为 <code>str</code>），若存在则不进行任何操作。</p> <p>(6) <code>mkfile str</code> 在（当前目录下）创建一个文件（名为 <code>str</code>），若存在则不进行任何操作。</p> <p>(7) <code>delete str</code> 删除（当前目录下）名为 <code>str</code> 的目录或文件，若不存在则不进行任何操作。</p> <p>(8) <code>save str</code>—— 将从根节点开始的目录树结构保存到文件（名为 <code>str</code>）中。</p> <p>(9) <code>load str</code> —— 从文件 <code>str</code> 中读取之前保存的目录树结构，并根据其重新建立当前目录树</p> <p>(10) <code>quit</code> —— 退出程序</p> <p>1.3 输入说明</p> <p>采用文件输入和文件输出，采用 <code>menu</code> 菜单展示功能</p> <p>输入界面设计</p>		

```
input.txt x main.cpp cc
1 cd
2 cd
3 dir
4 cd ..
5 cd
6 mkdir twoukhugngfgbl
7 cd ..
8 cd
9 dir
10 cd ..
11 cd /twoukhugngfgbl
12 cd ..
13 mkdir scnycqzgdbkagdu
14 mkdir ljjaek
15 mkfile roxnsocevdvonn
16 cd ljjaek
```

输入样例

输入样例过长，这里不进行展示，分为 10 个测试文件。

1.4 输出说明

输出界面设计

```
input.txt main.cpp console.h output.txt x
1 /
2 /
3 /
4 /
5 twoukhugngfgbl
6 /ljjaek
7 /ljjaek
8 cgewx*
9 /ljjaek
10 tobd*
11 /
12 roxnsocevdvonn*
13 ljjaek
14 scnycqzgdbkagdu
15 twoukhugngfgbl
16 jnvxbzulddmrupdewp*
17 nvayoueond*
```

输出样例

输出样例过长，这里不进行展示，分为 10 个测试文件。

2. 分析与设计

2.1 问题分析

本次实验设计并实现目录树 `CatalogTree` 的 ADT，并实现文件目录系统的模拟程序。我首先实现了目录树的 ADT，然后自己设计指令进行测试，根据不同的指令进行文件的创建，文件的删除，树结构打印等等，测试功能完善后，通过测试样例通过文件输入输出来检验正确性。成功完成基础实验后，我也自己设计了一些指令，如 `clear`, `display`, `size` 指令等等，深化了对于目录树的理解，完善了模拟系统的功能。

2.2 主程序设计

`CatalogTree.h`：目录树的类定义和函数实现

`Console.h`：菜单和输入输出

`Main.cpp`：主函数测试

`TreeNode.h`：树结点的类定义

2.3 设计思路

树结点是目录树的基本组成单位, 再通过孩子-兄弟-双亲法来实现树结构, 实现完 ADT 后再通过课程给定的样例进行测试功能的正确性.

2.4 数据及数据类型定义

目录树类

```
class CatalogTree {
public:
    TreeNode * root;
    TreeNode * nowPtr;//树节点指针, 指向树中我们当前访问位置的树节点
    CatalogTree();//构造函数
    ~CatalogTree() {}//析构函数
        deleteChildren(root);//删除树的根节点及其 n 辈孩子节点
};
void deleteChildren(TreeNode* D);//删除树的根节点及其 n 辈孩子节点
void findPath(TreeNode* D, ofstream& outfile, TreeNode* origin);//找到路径
void mkdir(string name, TreeNode* t);//在对应 t 位置创建对应名字的目录
void mkfile(string name, TreeNode* t);//在对应 t 位置创建对应名字的文件
void ListDir(ofstream& outfile);//列出当前目录下的文件
void Delete(string str);//删除某文件或目录 (通过给出相对路径/绝对路径)
void cd(ofstream& outfile);//打印当前目录的绝对路径
void cdPath(string str);//跳转到指定路径
void cdFather();//跳转到父路径
void saveCatalog(string filename);//将目录结构保存至文件
void loadCatalog(string filename);//将目录结构从文件载入
void printTree2File(TreeNode* D, int Depth, ofstream &outfile);//向文件打印出目录结构
void printFile2Tree(ifstream &infile);//从文件读取目录结构
int size();//返回当前目录下的文件数目
void clear() ;//清空当前目录下的所有文件
void clearAll() ;//清空整个目录树
void displayTree(TreeNode* D, int Depth, ofstream& outfile) ;//显示当前目录的树形结构
};
```

树结点类

```
class TreeNode {
public:
    TreeNode *parent;//父指针
    TreeNode *FirstChild;//第一个儿子指针
    TreeNode *brother;//兄弟指针
    bool isFile;//true 表示文件, false 表示目录
    string fileName;//用字符串存储树节点对应的文件或目录的名字
    int depth;//当前节点的深度
    int childrenSize;//当前节点的子文件数目
    TreeNode() {
        parent = NULL;
        FirstChild = NULL;
        brother = NULL;
        isFile = false;
    }
};
```

```
        fileName.clear();
        depth = 0;
        childrenSize = 0;
    }
};
```

2.5. 算法设计及分析

这里我们分析目录树的主要 ADT 和时间复杂度

删除该结点及其所有子孙结点

void deleteChildren(TreeNode* D):

- 基本思想：递归删除节点及其所有子节点。
- 时间复杂度：取决于节点的类型和子节点数量。对于每个节点，如果节点为文件，删除其兄弟节点的时间复杂度为 $O(n)$ ，其中 n 是兄弟节点的数量；如果节点为目录，删除其孩子节点的时间复杂度为 $O(h)$ ，其中 h 是目录的高度。

建立新的目录

void mkdir(string name, TreeNode* t):

- 基本思想：在给定节点下创建一个新的子目录节点。
- 时间复杂度： $O(n)$ ，其中 n 是给定节点的子节点数量。

建立新的文件

void mkfile(string name, TreeNode* t):

- 基本思想：在给定节点下创建一个新的文件节点。
- 时间复杂度： $O(n)$ ，其中 n 是给定节点的子节点数量。

列出所有文件和子目录

void ListDir(ofstream& outfile):

- 基本思想：列出当前目录下的所有文件和子目录。
- 时间复杂度： $O(m\log m)$ ，其中 m 是当前目录下的子节点数量。这是因为在列出文件和子目录时，需要对它们进行排序（set 的插入操作的平均时间复杂度为 $O(\log m)$ ）。

列出目录的绝对路径

void cd(ofstream& outfile):

- 基本思想：输出当前目录的绝对路径。
- 时间复杂度： $O(h)$ ，其中 h 是当前目录的深度。

跳转到相应的目录。

void cdPath(string str):

- 基本思想：根据给定路径跳转到相应的目录。
- 时间复杂度： $O(h)$ ，其中 h 是目标目录的深度。

将目录结构保存到文件中。

void saveCatalog(string filename):

- 基本思想：将目录结构保存到文件中。
- 时间复杂度： $O(n)$ ，其中 n 是目录树的节点数量。

从文件中加载目录结构。

`void loadCatalog(string filename):`

- 基本思想：从文件中加载目录结构。
- 时间复杂度： $O(n)$ ，其中 n 是文件中记录的节点数量。

递归查找节点的绝对路径。

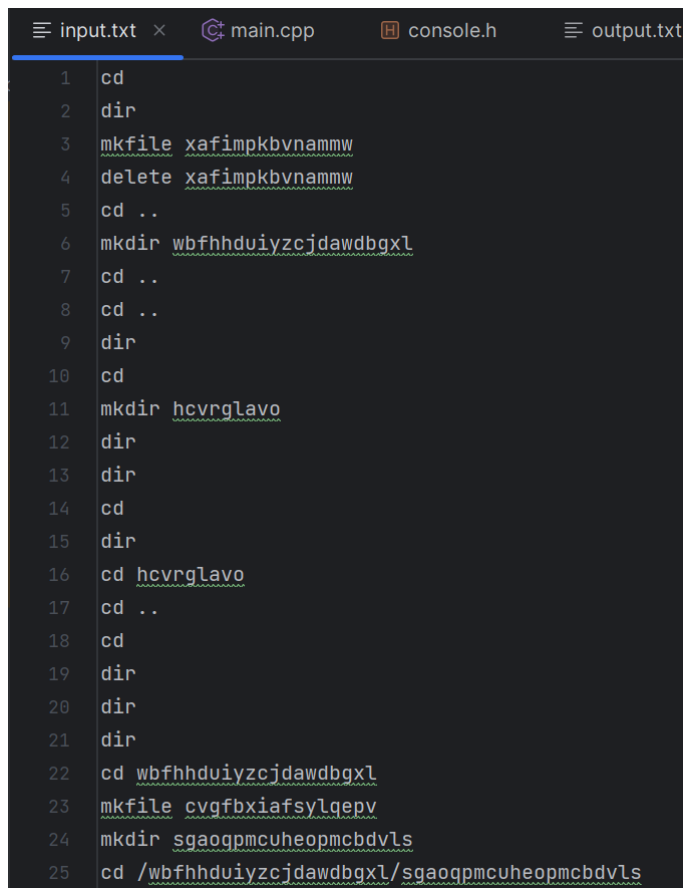
`void findPath(TreeNode* D, ofstream& outfile, TreeNode* origin):`

- 基本思想：递归查找节点的绝对路径。
- 时间复杂度： $O(h)$ ，其中 h 是目标节点到根节点的距离。

3. 测试

我们先进行基础功能的测试，通过测试样例进行检测，全部通过，以例 5 为例

将例 5 数据输入到 `input.txt` 文件中

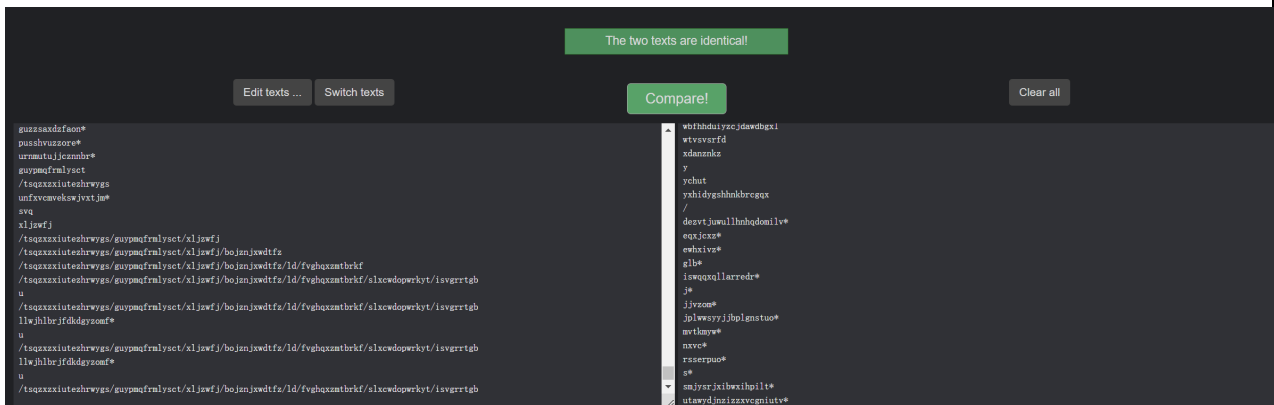


```
1 cd
2 dir
3 mkfile xafimpkbvnammw
4 delete xafimpkbvnammw
5 cd ..
6 mkdir wbfhhduiyzcjdawdbgxl
7 cd ..
8 cd ..
9 dir
10 cd
11 mkdir hcvrglavo
12 dir
13 dir
14 cd
15 dir
16 cd hcvrglavo
17 cd ..
18 cd
19 dir
20 dir
21 dir
22 cd wbfhhduiyzcjdawdbgxl
23 mkfile cvgfbxiafsylqepv
24 mkdir sgaoqpmcuheopmcbdvls
25 cd /wbfhhduiyzcjdawdbgxl/sgaoqpmcuheopmcbdvls
```

运行后，生成 `output.txt` 文件

```
input.txt  main.cpp  console.h  output.txt  test.txt  Catalog
1  /
2  wbfhhduiyzcjdawdbgxL
3  /
4  hcvrglavo
5  wbfhhduiyzcjdawdbgxL
6  hcvrglavo
7  wbfhhduiyzcjdawdbgxL
8  /
9  hcvrglavo
10 wbfhhduiyzcjdawdbgxL
11 /
12 hcvrglavo
13 wbfhhduiyzcjdawdbgxL
14 hcvrglavo
15 wbfhhduiyzcjdawdbgxL
16 hcvrglavo
17 wbfhhduiyzcjdawdbgxL
18 /wbfhhduiyzcjdawdbgxL/sgaoqpmcuheopmcbdvls
19 /wbfhhduiyzcjdawdbgxL/sgaoqpmcuheopmcbdvls/cymtx
20 /wbfhhduiyzcjdawdbgxL/sgaoqpmcuheopmcbdvls/cymtx/ozkr/lzbamlliinzcooiplo
21 /wbfhhduiyzcjdawdbgxL/sgaoqpmcuheopmcbdvls/cymtx/ozkr
```

与给定输出样例进行对比



则验证了正确性.

接下来我们测试 load , save, clear, size 等指令

Test.txt (使用\t 的个数来代表树的深度,)

```
root.d
  hi.d
    dd.d
      ww.f
      we.f
    2e.f
  wwe.d
  xfcwf.f
```

输入

```
load test.txt
cd hi/dd
size
clear
size
save text.txt
```

输出

Text.txt 文件

```
root.d
  hi.d
    dd.d
    2e.f
    wwe.d
    xfcwf.f
```

OutPut.txt 文件

```
2
d|
```

则成功验证了指令!

4. 分析与探讨

本次实验我通过学习目录树的数据结构, 了解它的思想, 并实现了基础功能, 通过这个有趣的数据结构成功完成了文件目录系统的模拟, 通过了测试样例.

此外, 我也成功设计了一些新的指令, 扩展了目录树, 深化了对它的理解.

当然, 过程不是一帆风顺的, 我也遇到了一些问题.

1. 如何实现树结构的打印?

答: 经过思考, 我是用\t 的个数来标志树的深度, 如 root 结点无\t, 它的孩子有 1 个\t... 并通过递归来实现打印.

2. 通过哪种方式来实现树结构?

答: 经过思考, 选用了最为全面的孩子-兄弟-双亲的树结点来实现树结构, 具有便于指令找到父结点, 找到兄弟结点等等, 非常实用.

5. 附录: 实现源代码

[CatalogTree.h](#)

```
#ifndef CatalogTree_CatalogTree_H
#define CatalogTree_CatalogTree_H
#include "bits/stdc++.h"
#include "TreeNode.h"
using namespace std;

class CatalogTree {
public:
    TreeNode * root;
    TreeNode * nowPtr;//树节点指针, 指向树中我们当前访问位置的树节点
```

```

CatalogTree();//构造函数
~CatalogTree() { //析构函数
    deleteChildren(root);//删除树的根节点及其 n 辈孩子节点
};
void deleteChildren(TreeNode* D);//删除树的根节点及其 n 辈孩子节点
void findPath(TreeNode* D, ofstream& outfile, TreeNode* origin);//找到路径
void mkdir(string name, TreeNode* t);//在对应 t 位置创建对应名字的目录
void mkfile(string name, TreeNode* t);//在对应 t 位置创建对应名字的文件
void ListDir(ofstream& outfile);//列出当前目录下的文件
void Delete(string str);//删除某文件或目录（通过给出相对路径/绝对路径）
void cd(ofstream& outfile);//打印当前目录的绝对路径
void cdPath(string str);//跳转到指定路径
void cdFather();//跳转到父路径
void saveCatalog(string filename);//将目录结构保存至文件
void loadCatalog(string filename);//将目录结构从文件载入
void printTree2File(TreeNode* D, int Depth, ofstream &outfile);//向文件打印出目录结构
void printFile2Tree(ifstream &infile);//从文件读取目录结构
int size();//返回当前目录下的文件数目
void clear() ;//清空当前目录下的所有文件
void clearAll() ;//清空整个目录树
void displayTree(TreeNode* D, int Depth, ofstream& outfile) ;//显示当前目录的树形结构
};

void CatalogTree::deleteChildren(TreeNode *D) {
    //删除的是文件
    if (D->isFile) {
        auto t = D->parent->FirstChild;
        //删除的是第一个孩子
        if (D == t) {
            D->parent->FirstChild = D->brother;
        }
        //删除的是非第一个孩子，则将其前一个孩子的 brother 指向其后一个孩子
        else {
            while (t->brother != D) {
                t = t->brother;
            }
            t->brother = D->brother;
        }
        delete D;
    }
    //删除的是目录
    else {
        //如果没有孩子，直接删除
        if (D->FirstChild == NULL) {
            //D 为根节点
            if (D == root) {

```



```

        return;
    }
    //处理 D 的父节点
    if (D->parent->FirstChild == D) {
        D->parent->FirstChild = D->brother;
    }
    else {
        auto t = D->parent->FirstChild;
        while (t->brother != D) {
            t = t->brother;
        }
        t->brother = D->brother;
    }
}
//有孩子, 递归删除
else {
    while (D->FirstChild != NULL) {
        auto f = D->FirstChild;
        while (f->brother != NULL) {
            f = f->brother;
        }
        deleteChildren(f);
    }
    deleteChildren(D);
}
}
}

```

```

void CatalogTree::mkdir(string name, TreeNode *t) {
    //不能产生相同的文件/目录
    TreeNode* temp = t->FirstChild;
    while (temp != NULL) {
        if (temp->fileName == name) {
            return;
        }
        temp = temp->brother;
    }
}

```

```

TreeNode *newNode = new TreeNode();
newNode->fileName = name;
newNode->parent = t;
newNode->depth = t->depth + 1;
newNode->isFile = false; //创建的是目录
newNode->childrenSize = 0;
// t->childrenSize++; 目录不需要增加孩子数目
if (t->FirstChild == NULL) {

```

```

        t->FirstChild = newNode;
    }
    else {
        auto temp = t->FirstChild;
        while (temp->brother != NULL) {
            temp = temp->brother;
        }
        temp->brother = newNode;
    }
}

void CatalogTree::mkfile(string name, TreeNode *t) {
    //不能产生相同的文件/目录
    TreeNode* temp = t->FirstChild;
    while (temp != NULL) {
        if (temp->fileName == name) {
            return;
        }
        temp = temp->brother;
    }

    TreeNode *newNode = new TreeNode();
    newNode->fileName = name;
    newNode->parent = t;
    newNode->depth = t->depth + 1;
    newNode->isFile = true; // 创建的是文件
    newNode->childrenSize = 0;
    if (t->FirstChild == NULL) {
        t->FirstChild = newNode;
    }
    else {
        auto temp = t->FirstChild;
        while (temp->brother != NULL) {
            temp = temp->brother;
        }
        temp->brother = newNode;
    }
    //更新孩子数目
    while (t != NULL) {
        t->childrenSize++;
        t = t->parent;
    }
}

void CatalogTree::ListDir(ofstream &outfile) {

```

```

auto t = nowPtr ->FirstChild;
set<string> fileSet, dirSet;
while (t != NULL) {
    string temp = t->fileName;
    if (t -> isFile) {
        temp += "*";
        fileSet.insert(temp);
    }
    else {
        dirSet.insert(temp);
    }
    t = t->brother;
}
for (auto it = fileSet.begin(); it != fileSet.end(); it++) {
    outfile << *it << endl;
}
for (auto it = dirSet.begin(); it != dirSet.end(); it++) {
    outfile << *it << endl;
}
}

```

```

void CatalogTree::Delete(string str) {
    TreeNode* t = nowPtr->FirstChild;
    while (t != NULL) {
        if (t->fileName == str) {
            deleteChildren(t);
//            cout << "删除成功" << endl;
            return;
        }
        t = t->brother;
    }
//    cout << "未找到该文件/目录" << endl;
}

```

```

void CatalogTree::cd(ofstream &outfile) {
    findPath(nowPtr, outfile, nowPtr);
}

```

```

void CatalogTree::cdPath(string str) {
    if (str == "/") {
        nowPtr = root;
        return;
    }
    //绝对路径
    if (str[0] == '/') {
        nowPtr = root;
    }
}

```

```

        str = str.substr(1); //去掉第一个字符
    }
    stringstream ss(str);
    string temp;
    auto t = nowPtr->FirstChild;
    auto pre = t;
    while (getline(ss, temp, '/')) { //按照'/'分割字符串
        bool flag = false;
        while (t != NULL) {
            if (t->fileName == temp) {
                pre = t;
                t = t->FirstChild;
                flag = true;
                break;
            }
            else {
                t = t->brother;
            }
        }
        if (flag == false) {
            return;
        }
    }
    nowPtr = pre;
}

void CatalogTree::cdFather() {
    if (nowPtr != root) {
        nowPtr = nowPtr->parent;
    }
    else {
//        cout << "已经是根目录了" << endl;
    }
}

void CatalogTree::saveCatalog(string filename) {
    ofstream inf(filename, ios::trunc);
    printTree2File(root, 0, inf);
    inf.close();
}

void CatalogTree::loadCatalog(string filename) {
    ifstream inf(filename, ios::in);
    printFile2Tree(inf);
    inf.close();
    cdPath("/");
}

```

```

}

void CatalogTree::findPath(TreeNode *D, ofstream &outfile, TreeNode* origin) {
    if (D == root) {
        outfile << "/";
        return;
    }
    else {
        findPath(D->parent, outfile, origin);
        outfile << D->fileName;
        if (D != origin) {
            outfile << "/";
        }
    }
}

```

```

void CatalogTree::printTree2File(TreeNode *D, int Depth, ofstream &outfile) {
    if (D == NULL) {
        return;
    }
    for (int i = 0; i < Depth; i++) {
        outfile << "\t";
    }
    outfile << D->fileName;
    if (D->isFile) {
        outfile << ".f" << endl;
    }
    else {
        outfile << ".d" << endl;
        auto t = D->FirstChild;
        while (t != NULL) {
            printTree2File(t, Depth + 1, outfile);
            t = t->brother;
        }
    }
}

```

```

void CatalogTree::printFile2Tree(ifstream &infile) {
    string temp;
    string preDir = "/";
    int preDepth = 0;
    while (getline(infile, temp)) {
        int depth = 0;
        while (temp[depth] == '\t') {
            depth++;
        }
    }
}

```

```

        if (depth == 0) {
            continue;
        }
        temp = temp.substr(depth); //去掉前面的\t
        //子目录
        if (depth > preDepth) {
            cdPath(preDir);
        }
        //父目录
        else if (depth < preDepth) {
            for (int i = 0; i < preDepth - depth; i++) {
                cdFather();
            }
        }
        //默认在同一层
        string fileName = temp.substr(0, temp.size() - 2);
        if (temp[temp.size() - 1] == 'f') {
            mkfile(fileName, nowPtr);
        }
        else if (temp[temp.size() - 1] == 'd'){
            mkdir(fileName, nowPtr);
        }
        preDir = fileName;
        preDepth = depth;
    }
}

```

```

CatalogTree::CatalogTree() {
    root = new TreeNode();
    root->fileName = "root";
    root->depth = 0;
    root->isFile = false;
    root->childrenSize = 0;

    nowPtr = root;
}

```

```

int CatalogTree::size() {
    return nowPtr->childrenSize;
}

```

```

void CatalogTree::clear() {
    auto t = nowPtr->FirstChild;
    while (t != NULL) {
        deleteChildren(t);
        t = t->brother;
    }
}

```

```

    }
    nowPtr->FirstChild = NULL;
    nowPtr->childrenSize = 0;
}

void CatalogTree::clearAll() {
    deleteChildren(root);
    root->FirstChild = NULL;
    root->childrenSize = 0;
    nowPtr = root;
}

void CatalogTree::displayTree(TreeNode* D, int Depth, ofstream& outfile) {
    printTree2File(D, Depth, outfile);
}

#endif //CatalogTree_CatalogTree_H
Console.h
#ifndef CATALOGTREE_CONSOLE_H
#define CATALOGTREE_CONSOLE_H
#include "bits/stdc++.h"
#include "CatalogTree.h"
using namespace std;
void menu()
{
    cout << "目录树系统\n";
    cout << "dir          列出当前目录下的所有子目录与文件项\n";
    cout << "cd            列出当前目录的绝对路径\n";
    cout << "cd ..        当前目录变为当前目录的父目录\n";
    cout << "cd str       当前目录变为 str 所表示路径的目录\n";
    cout << "mkdir str    在 (当前目录下 )创建一个子目录";
    cout << "mkfile str   在 (当前目录下 )创建一个文件";
    cout << "delete str   删除当前目录下名为 str 的目录或文件\n";
    cout << "save str     从根节点开始保存虚拟目录到 str 文件中\n";
    cout << "load str     载入 str 文件中的虚拟目录重新构建目录树\n";
    cout << "help        显示帮助信息\n";
    cout << "size        显示当前目录下的文件数目\n";
    cout << "display     显示当前目录的树形结构\n";
    cout << "clear       清空当前目录下的所有文件\n";
    cout << "clearall    清空整个目录树\n";
    cout << "quit       退出\n";
}

void exec() {
    CatalogTree T;
    menu();
}

```

```

// while (1) {
//     cout << "->";
string s, op, str = "";
ifstream ifile ("input.txt", ios::in);
ofstream ofile ("output.txt", ios::trunc);
while (getline(ifile, s)) {
    stringstream ss(s);
    str = "";
    getline(ss, op, ' ');
    getline(ss, str, ' ');
    if (op == "dir") {
        T.ListDir(ofile);
    }
    else if (op == "cd") {
        if (str == "") {
            T.cd(ofile);
//            cout << endl;
            ofile << endl;
        }
        else if (str == "..") {
            T.cdFather();
        }
        else {
            T.cdPath(str);
        }
    }
    else if (op == "mkdir") {
        T.mkdir(str, T.nowPtr);
    }
    else if (op == "mkfile") {
        T.mkfile(str, T.nowPtr);
    }
    else if (op == "delete") {
        T.Delete(str);
    }
    else if (op == "save") {
        T.saveCatalog(str);
    }
    else if (op == "load") {
        T.loadCatalog(str);
    }
    else if (op == "quit") {
//        ofile << "Bye!\n";
        return ;
    }
    else if (op == "help") {

```



```
        menu();
    }
    else if (op == "size") {
        ofile << T.size() << endl;
    }
    else if (op == "display") {
        T.displayTree(T.root, 0, ofile);
    }
    else if (op == "clear") {
        T.clear();
    }
    else if (op == "clearall") {
        T.clearAll();
    }
    else {
        ofile << "Invalid command\n";
    }
}

#endif //CATALOGTREE_CONSOLE_H
```