山东大学计算机科学与技术学院

数据结构与算法课程设计报告

学号：202000130143	姓名： 郑凯饶	班级： 20.1
上机学时： 6		日期：2022-5-5
课程设计题目：模拟文件目录系统		
软件环境：Windows 10 家庭中文版 64 位（10.0，版本 18363） Microsoft VS Code cmake version 3.23.0-rc3		

报告内容：

1.需求描述

1.1 问题描述

使用树结构实现一个简单文件目录系统的模拟程序。

1.2 基本要求

（1）设计并实现目录树 CatalogTree 的 ADT。

（2）应用以上 CatalogTree 结构设计并实现一文件目录系统的模拟程序。

（3）文件目录系统程序应该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。它支持基本操作：dir, cd, cd.., cd str, mkdir str, mkfile str, delete str, save str, load str.....

1.3 输入说明

输入界面设计

命令行输入

输入样例

```
cd
cd ..
cd ..
dir
mkdir eymdfzzcdpimmomk
cd
cd ..
mkdir cewbfowgaepoecuqkfpi
cd
dir
mkfile nxpmeqkcujygxvui
mkfile r
mkdir bgbhkznkiyjolvhgw
cd
cd bgbhkznkiyjolvhgw
mkdir tcgrocyrpdwomp
mkfile atpjcstt
cd
cd tcgrocyrpdwomp
cd
```

cd ..
delete atpjcstt
mkfile iycowkcutz
mkdir tlucktswmufx
cd ..
cd ..
cd
dir
quit

1.4 输出说明

输出界面设计

使用系统函数 system(“cls”)清屏后再输出

输出样例

/
/
/
cewbfowgaepoecuqkfpi
eymdfzzcdpimmomk
/
/bgbhkznkiyjolvhgw
/bgbhkznkiyjolvhgw/tcgrocyrpdwomp
/
nxpmeqkcujygxvui*
r*
bgbhkznkiyjolvhgw
cewbfowgaepoecuqkfpi
eymdfzzcdpimmomk

2.分析与设计

2.1 问题分析

系统主要问题的是目录树的设计，根据设计命令的需求我们归纳出目录树的设计应满足可以寻找节点的父亲节点以及孩子节点，正常可以维护指向父亲及孩子的指针，但我选择一种更满足目录树性质的存储方式，存储当前访问的绝对路径，即当前节点所有的祖先节点和它本身。

2.2 主程序设计

```
void run() {
    catalogTree tree;

    string s;

    // cin.sync();

    if (D)…

    while (getline(cin, s, '\n')) {
        if (D) cout << s << '\n';
        string cmd, str;
        stringstream ss(s);
        ss >> cmd;
        ss >> str;

        if (cmd == "dir") {…
        else if (cmd == "cd") {…
        else if (cmd == "mkdir") {…
        else if (cmd == "mkfile") {…
        else if (cmd == "delete") {…
        else if (cmd == "save") {…
        else if (cmd == "load") {…
        else if (cmd == "quit") {…
    }
}
```

主程序主要是进行字符串命令解析、调用目录树的相应方法。

2.3　设计思路

确定目录的结构之后设计主要根据指令需求进行即可。可以发现我们主要设计与文件节点和 filePtr 相关的操作。由于目录树实际分散地存储于内存中，修改操作问时应通过指针。

2.4 数据及数据类(型)定义

目录树节点 file 的设计：

```
struct file
{
    bool isDir;
    string fileName;
    list<file> children;
    ll filesize;

    bool operator < (const file &a) const {
        if (isDir != a.isDir) return a.isDir;
        return fileName < a.fileName;
    }
};
```

将文件分类为目录文件以及普通文件，通过 Isdir 标识。前者拥有孩子节点构成的线性表 children，后者拥有实际的文件内容，用文件大小 filesize 抽象描述。fileName 为文件名，也是文件的唯一标识符。

重载比较运算符，根据系统设计要求，输出当前目录下的全部孩子时要按照字典序以及普通文件在前的次序。字典序的比较可以直接使用 string 内置的比较方法。这也是为什么使用 list 作为 children 实现，可以高效地增加、删除孩子节点，并维持其有序，这里我采用 STL 的 list。

目录树的定义：

```
class catalogTree {
    public:
        catalogTree();
        ~catalogTree() { delete root; }

        void dir();
        void cd();
        void cd(ofstream &);
        void cd_();
        void cd_str(string &);
        void mkdir(string &);
        void mkfile(string &);
        void delet(string &);
        void save(string &);
        void load(string &);
        void quit() { exit(0); }

    private:
        file *root;                 // 根目录文件
        list<file *> filePtr;      // 保存当前路径
        vector<string> parsePath(string);
        void findPath(vector<string>);
        void dfs(ofstream &, file *);
};
```

Private 部分：

Root:根目录文件指针；

filePtr：保存当前绝对路径的 file 指针链表；

parsePath(string)的实现：

```
vector<string> catalogTree::parsePath(string path) {
    stringstream ss(path);
    string fileName;
    vector<string> res;

    while (getline(ss, fileName, '/')) {
        res.push_back(fileName);
    }
    return res;
}
```

该函数用于在执行 cd str 命令时将命令中的绝对路径参数解析为一级级文件名的 vector 数组；

FindPath(vector<string>)的实现：

```
void catalogTree::findPath(vector<string> fileNames) {
    filePtr.clear();

    // 重新解析当前路径
    filePtr.push_back(root);
    for (int i = 1; i < fileNames.size(); i++) {
        file *p = filePtr.back();
        for (auto &j : p->children) {       // 引用，否则将形参地址加入filePtr
            if (j.fileName == fileNames[i]) {
                filePtr.push_back(&j);
                break;
            }
        }
    }
}
```

接收来自 parsePath()的 fileNames，根据文件名信息重新设置当前的绝对路径；
Dfs(ofstream &, file *)的实现：

```
void catalogTree::dfs(ofstream &f, file *fp) {
    f << fp->fileName << ' ' << fp->isDir << ' ';
    if (fp->isDir) {
        f << "{ ";
        for (auto &i : fp->children) {
            dfs(f, &i);
        }
        f << "} ";
    }
}
```

用于 save 命令时规整化输出目录树结构。

Public 部分是向外部开放的接口，直接对应具体的命令。
dir()：有序输出当前目录下的文件，直接访问当前目录的 children 输出；
cd()：输出 filePtrs 中文件的文件名；
cd(ostream &)：参数重载，向指定文件输出绝对路径，由于 save 命令要求不仅要还原目录树结构，还要恢复到最后访问的文件路径下，因此需要存储该信息并重新解析；
cd__()：返回父目录，根据定义直接对 filePtr 进行 pop_back()即可；
cd_str()：跳转到绝对路径 str 或者当前目录的孩子 str 处；
mkdir()的实现：

```
// 创建子目录
void catalogTree::mkdir(string &dirName) {
    file *p = filePtr.back();
    file *newDir = new file;
    newDir->isDir = true;
    newDir->fileName = dirName;

    // 插入至指定位置
    bool flag = false;
    for (auto i = p->children.begin(); i != p->children.end(); i++) {
        if (*newDir < *i) {
            p->children.insert(i, *newDir);
            flag = true;
            break;
        }
    }
    if (!flag) p->children.push_back(*newDir);
}
```

首先访问 filePtr.back()获取指向当前目录的文件指针 p，为新的目录文件*newDir 开辟空间，之后基于类似于插入排序的思想，将*newDir 插入至第一个大于它的文件之前。注意处理 newDir 为最大文件的情形；
Mkfile()类同 mkDir()；
Delet()：找到指定文件，调用 list 的 erase()方法，注意迭代器的丢失，删除后应立即返回；
Save()：调用私有函数 dfs()结构化输出目录树；
Load()的实现：

```cpp
// 重建树
void catalogTree::load(string &path) {
    ifstream fin(path);

    if (!fin.is_open()) {
        cout << "fail to open!\n";
        exit(0);
    }

    string p;
    fin >> p;

    filePtr.push_back(root);
    string c, fn;
    bool isD;
    fin >> isD;
    fin >> c;       // 根目录特殊处理，直接将第一个'{'读出
    while (fin >> c) {
        if (c == "{") {     // 切换为子目录
            cd_str(fn);
        }
        else if (c == "}") {
            cd_();
        }
        else {
            fn = c;
            fin >> isD;
            if (isD) {
                mkdir(fn);
            }
            else {
                mkfile(fn);
            }
        }
    }

    // 还原之前保存路径
    cd_str(p);

    fin.close();
}
```

解析结构化目录树输出。
我存储目录树的方式是空格分隔，输出文件名，文件类型（目录，普通），若为目录文件，则输出“{}”，大括号中包含当前文件的孩子对象。
因此读到“{”时调用 cd_str()切换至子目录 str；读到“}”时表示已经读完某个对象的孩子文件，调用 cd__()返回上一级目录；除此之外读到的是当前目录下的孩子，根据文件类型调用 mkdir()或者 mkfile()进行文件创建。

2.5.算法设计及分析
以下我们分析 filePtr 带来的优化：
Cd():可直接输出绝对路径，不必递归访问；
Cd__() or cd_str()：父子目录跳转时，直接对 filePtr 进行操作，不必通过指针在节点之间移动；

Save()与 load()算法分析：
目录树本质上是一棵树，而树的存储实际存储节点的内容外再存储节点之间的直接父子关系就可以还原树的结构，而我通过“{}”标识了父子关系。
算法的时间复杂度：设目录树的节点数量为 n（包含目录及普通文件），save 是深度优先遍历每个节点，因此复杂度为 O(n)。load 命令的执行中，cd__()为 O(1)，cd_str()为 O(当前目录的孩子个数) < O(n)，mkdir()以及 mkfile()操作涉及插入，极端情况也可能遍历当前目录的所有孩子，因此也为 O(n)。综上所述，load 命令的时间复杂度为 O(n^2)。

3. 测试
正确性验证：
运行法官程序 localJudge.cpp 将 moutx.txt 和标准输出 outx.txt 进行比对：

```
The result is as follows:
Test point 1 : Accept
Test point 2 : Accept
Test point 3 : Accept
Test point 4 : Accept
Test point 5 : Accept
Test point 6 : Accept
Test point 7 : Accept
Test point 8 : Accept
Test point 9 : Accept
Test point 10 : Accept
```

4. 分析与探讨

这次实验我尝试设计一个目录树模拟系统，实现了文件节点之间的跳转、文件创建、目录树持久化存储等等功能，并尝试创新使用 filePtr 优化访问。filePtr 可以免去指针的直接使用，转而调用 STL 的封装方法，更加安全高效。目录树是 OS 的基石，希望有机会可以深入学习成熟的目录树设计，而不是仅仅在文件操作的抽象层进行学习，进一步学习内存管理、系统调度等等知识。

5. 附录：实现源代码

（另附）