山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202200130048	姓名： 陈静雯	班级： 6
实验题目：模拟文件目录系统		
实验学时：6	实验日期： 4.30	

实验目的：

设计并实现目录树 CatalogTree 的 ADT，用它来表达字符串集合组成的有序树。应用以上 CatalogTree 结构设计并实现一文件目录系统的模拟程序，并提供模拟操作界面。

软件开发工具：

Vscode

1. 实验内容

目录系统具有如下基本操作：

① dir ——列出当前目录下的所有目录项
② cd ——打出当前目录的绝对路经
③ cd ..——当前目录变为当前目录的父目录
④ cd str——当前目录变为 str 所表示路径的目录
⑤ mkdir str ——在(当前目录下)创建一个子目录(名为 str)
⑥ mkfile str ——在(当前目录下)创建一个文件(名为 str)
⑦ delete str ——删除(当前目录下)名为 str 的目录或文件

要求与内容：

（1）描述并实现 CatalogTree 的 ADT，包括其上的基本操作：如插入一个结点，寻找一个结点，返回一个结点的最左儿子等（具体情况依据应用自定）。

（2）应用 CatalogTree 的 ADT 实现一个模拟文件目录系统的应用程序。

（3）应用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（quit）。命令名及其含义如上所述。

（4）目录树结构可以保存（save）到文件中，也可从文件中读出（load *.dat）。

（5）dir 命令的结果应能够区分是子目录和还是文件。

（6）应对命令 ④~⑦中的 str 区分是绝对路径，还是相对路径。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

（1）node，有 parent 父指针，child 的 map，filetype（0 是文件，1 是目录），filename 文件名，childnum 孩子个数，filesize 文件大小，allsize 目录配额，son_used 孩子大小

（2）Load（），从 load 读取每一行，load 文件设置目录为“+文件名 孩子个数 配额”和普通文件“文件名 大小”，第一次肯定是根目录，调用 read_mulu（）函数，每次读取每行的第一个字母，判断是不是“+”，是则该项为目录，node 的 filetype 设为 1，并递归调用 read_mulu 函数，如果第一个字母不是+，则该项是文件，直接读取即可。

（3）save（），首先将 cout 改为输出到文件 mls.txt，从根目录开始，递归调用 dir，如果是目录在对应文件名前先输出一个“+”，递归调用 dir，如果是普通文件，直接输出，每一行通过传递的参数层级 n，输出空格，来进行排列。

（4）dir（），传递参数 n（层数），root1（当前的目录项），对 root1，输出它的每个孩子，根据 n 的大小

决定每行开头输出的空格数，如果孩子中有目录项，那么递归调用 dir（）进行输出。

（5）cd（），对当前项 cur，用一个 string 数组保存从 cur 开始的父节点直到 root 为止，然后倒序输出即可得到绝对路径

（6）cd_back（），把 cur 变成 cur 的父节点

（7）cd str，cd_change（），对于某路径 str，先对其进行分割，得到路径的字符串数组，判断它是绝对路径还是相对路径，是回到上一级，还是去下一级。对 path[0]进行讨论，如果是从根目录开始，那么是绝对路径，直接按照 path 从根开始找即可，如果 path[0]是“..”，那么就要回到上一级，如果都不是，那么就是去下一级，从 cur 的孩子节点里找即可。若到某一步发现孩子节点没有需要的路径名则查找失败，返回失败的路径名，cur 为失败的前一目录。若 path 都遍历完，说明找到需要的路径了，返回空字符串，cur 指向最后找到的项。

（8）mkdir（str,size），首先 cd str，判断 str（可能是相对路径也可能是绝对路径），因为 cd_change 返回的是查找失败的那一项文件名，即需要创建的文件名 thename，如果返回空说明重名，也有可能要创建的项不是目录项，也会创建失败，输出错误信息。对 thename，先判断 cur 及 cur 的所有父节点配额大小是否够创建文件，如果够，增加 cur 的孩子节点，并修改父节点的 son_used 大小，如果配额不够，创建失败，返回错误信息。

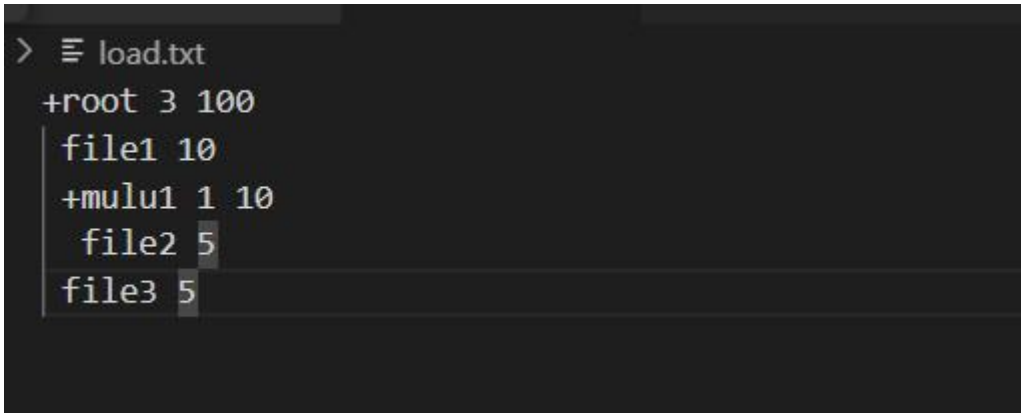
（9）mkdfile，过程同 mkdir，只是创建的文件类型变成文件。

（10）delete，先对 str 查找，判断返回值，如果返回空字符串，说明找到要删除掉文件，进行删除，如果没有则删除失败。

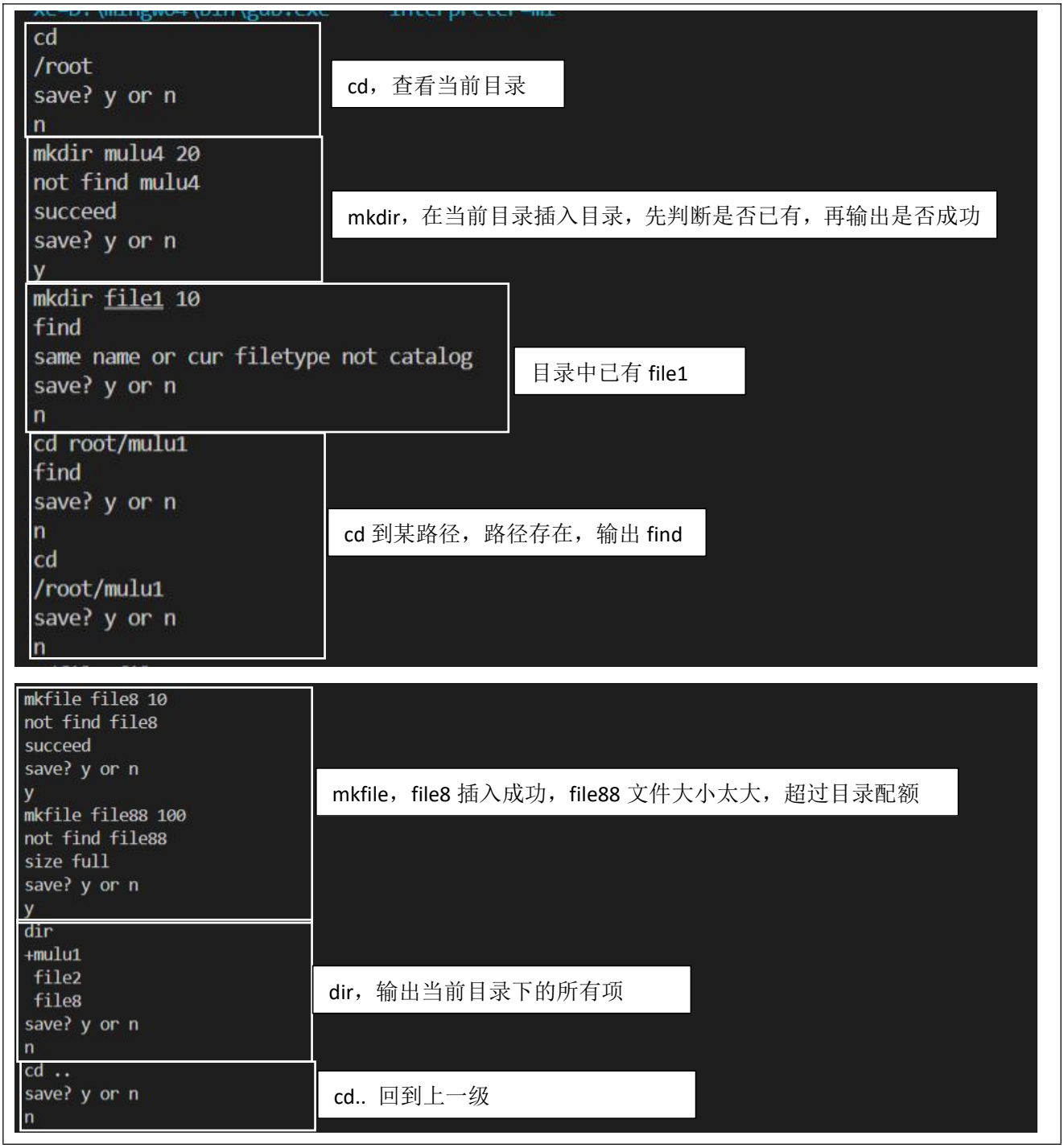
（11）main 函数中，用一个 while 不断接收输入的字符串，根据不同的命令调用不同的函数直到没有输入停止程序，每次命令询问是否需要保存到文件。

3. 测试结果（测试输入，测试输出）

初始 load 文件，加载到目录树



对应的输入输出


cd
/root
save? y or n
n
cd，查看当前目录
mkdir mulu4 20
not find mulu4
succeed
save? y or n
y
mkdir，在当前目录插入目录，先判断是否已有，再输出是否成功
mkdir file1 10
find
same name or cur filetype not catalog
save? y or n
n
目录中已有 file1
cd root/mulu1
find
save? y or n
n
cd
/root/mulu1
save? y or n
n
cd 到某路径，路径存在，输出 find
mkfile file8 10
not find file8
succeed
save? y or n
y
mkfile file88 100
not find file88
size full
save? y or n
y
mkfile，file8 插入成功，file88 文件大小太大，超过目录配额
dir
+mulu1
file2
file8
save? y or n
n
dir，输出当前目录下的所有项
cd ..
save? y or n
n
cd.. 回到上一级

```
cd
/root
save? y or n
n
delete file1
find
succeed
save? y or n
n
dir
+root
 file3
 +mulu1
  file2
  file8
 +mulu4
save? y or n
y
```

delete 删除项

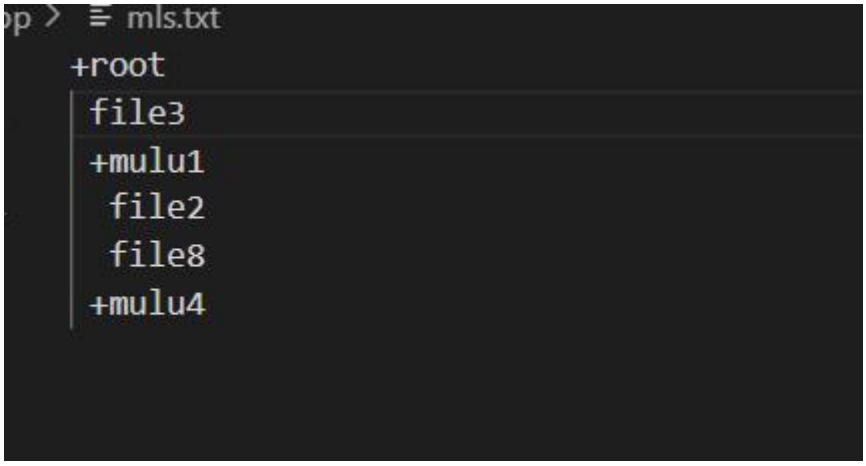
对应保存后的文件
最开始在 root 下插入 mulu4

```
+root
| file1
| file3
| +mulu1
|  file2
| +mulu4
```

在 mulu1 下插入 file8，（file88 不能插入，大小过大）

```
+root
| file1
| file3
| +mulu1
|  file2
|  file8
| +mulu4
```

删除 file1



4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）
已增加配额项，修改配额用的是 0（n）的复杂度，就是从根到对应文件的路径上都改一遍，应该可以降低修改配额的时间复杂度

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

```
#include<iostream>
#include<fstream>
#include<map>
#include<vector>
#include<sstream>
using namespace std;
typedef long long ll;

struct node{
    node* parent;   //父节点
    map<string, node*> child;//孩子节点
    int filetype;   //文件类型，0 是文件，1 是目录
    string filename;//名
    int childnum;
    ll filesize;   //该文件大小

    ll all_size;   //配额
    ll son_used;   //孩子大小

    node(int type, string fname, ll size=0){
        filetype=type;
        filesize=size;
        parent=NULL;
        child.clear();
        filename=fname;
        all_size=size;
        son_used=0;
    }
```

```
    bool preadd(ll size){  //预分配
        if(all_size<=0||son_used+size>all_size) return false;
        return true;
    }
    void addsize(ll size){
        son_used+=size;
    }
    bool setsize(ll size){
        if(filetype&&size>=son_used){
            all_size=size;
            return true;
        }
        return false;
    }
};

class catalogtree{
public:
    catalogtree(ll size){
        root=new node(1,"root",size);
        root->all_size=size;
        cur=root;
    }
    void save(){
        auto st=cout.rdbuf();
        string fileto="mls.txt";
        ofstream fileout(fileto);
        fileout<<"+"<<root->filename<<'\n';
        cout.rdbuf(fileout.rdbuf());
        dir1(1,root);
        cout.rdbuf(st);
    }
    void load();
    void read_mulu(ifstream &filein,  int num,int n);
    void dir(){
        cout<<'+'<<cur->filename<<'\n';
        dir1(1,cur);
    }
    void dir1(int n,node* root1);//写出当前目录所有目录和文件
    void cd();  //写出当前目录的绝对路径
    void cd_back();//返回上一级
    string cd_change(string str);//更改当前目录为路径 str
    void mkdir(string str,ll size);//创建子目录
    void mkfile(string str,ll size);//创建子文件
    void deletestr(string str);//删除 str 文件或目录
private:
```

```
    node* root;
    node* cur;
};

void catalogtree::load(){
    ifstream filein("load.txt");  //load 文件中每一行有名字 子目录项数目 配额大小
    string temp;
    char c;
    filein>>c;
    filein>>temp;
    root->filename=temp;
    cur=root;
    int num,size;
    filein>>num>>size;
    cur->childnum=num;
    cur->filetype=1;
    cur->all_size=size;
    read_mulu(filein,num,1);
}

void catalogtree::read_mulu(ifstream &filein, int num, int n){
    char c;
    for(int i=0;i<num;i++){
        filein>>c;
        string name;int nn;ll ss;
        node* cc=new node(0,name,ss);
        if(c=='+'){
            filein>>name>>nn>>ss;
            cc->filetype=1;
            cc->filename=name;
            cc->all_size=ss;
            cc->parent=cur;
            cc->childnum=nn;
            cur->child.insert({name,cc});
            cur=cc;
            read_mulu(filein,nn,++n);       //+表示目录项，需要递归读取
            cur=cur->parent;
            n--;
        }
        else{
            string name2;
            filein>>name2>>ss;
            name=c+name2;
            cc->parent=cur;
            cc->filename=name;
            cc->all_size=ss;
```

```
            cur->child.insert({name,cc});
        }
        cur->addsize(cc->son_used);
    }
}

void catalogtree::dir1(int n,node*root1){  //n 为行首空格数
    if(root1==NULL) return;
    auto ip=root1->child.begin();
    for(;ip!=root1->child.end();ip++){
        for(int j=0;j<n;j++) cout<<' ';
        if(ip->second->filetype){
            cout<<'+'<<ip->first<<'\n';
            dir1(++n,ip->second);
            n--;
        }
        else{
            cout<<ip->first<<'\n';
        }
    }
}

void catalogtree::cd(){
    vector<string>str;
    node* pre=cur;
    while(pre!=root){
        str.push_back(pre->filename);
        pre=pre->parent;
    }
    cout<<'/'<<root->filename;
    for(int i=str.size()-1;i>=0;i--){
        cout<<'/'<<str[i];
    }
    cout<<'\n';
}

void catalogtree::cd_back(){
    cur=cur->parent;
}

string catalogtree::cd_change(string str){   //对某路径 str，若路径正确，返回空，
若路径中有找不到的目录名，cur 变为缺失名的前一项，返回缺失值
    vector<string>path;
    stringstream ss(str);
    string temp;
    while(getline(ss,temp,'/')){
```

```
        path.push_back(temp);
    }
    if(path[0]!=root->filename){  //相对路径
        if(path[0]==".."){ //回到上级
            for(int i=0;i<path.size();i++){
                if(path[i]==".."){
                    cur=cur->parent;
                }
                else{
                    auto ip=cur->child.find(path[i]);
                    if(ip!=cur->child.end()) cur=ip->second;
                    else{
                        cout<<"not find "<<path[i]<<'\n';
                        return path[i];
                    }
                }
            }
        }
        else{  //去下一级
            for(int i=0;i<path.size();i++){
                auto ip=cur->child.find(path[i]);
                if(ip!=cur->child.end()) cur=ip->second;
                else{
                    cout<<"not find "<<path[i]<<'\n';
                    return path[i];
                }
            }
        }
    }
    else{                //绝对路径
        cur=root;
        for(int i=1;i<path.size();i++){
            auto ip=cur->child.find(path[i]);
            if(ip!=cur->child.end()) cur=ip->second;
            else{
                cout<<"not find "<<path[i]<<'\n';
                return path[i];
            }
        }
    }
    cout<<"find"<<'\n';
    return "";
}

void catalogtree::mkdir(string str,ll size=10){
    string thename=cd_change(str);
```

```
    if(cur->filetype==0){
        cout<<"same name or cur filetype not catalog"<<endl; //无重名且插入到的文
件时目录
        return;
    }
    auto ip=cur->child.find(thename);
    if(ip==cur->child.end()){
        node* temp=new node(1,thename,size); //文件没有重名
        temp->parent=cur;
        temp->setsize(size);
        node* t=cur;
        int check=0;
        while(t!=root){
            if(t->preadd(size)==false){  //判断配额大小够不够
                check=1;
                break;
            }
            t=t->parent;
        }
        if(t->preadd(size)==false) check=1;
        if(check){
            cout<<"size full"<<endl;
        }
        else{
            node* tempt=cur;             //插入到目录下，并修改子目录项总大小
            while(tempt!=root){
                temp->addsize(size);
                tempt=tempt->parent;
            }
            root->addsize(size);
            temp->parent=cur;
            cur->child.insert({thename,temp});
            cout<<"succeed"<<endl;
        }
    }
    else{
        cout<<"same name"<<endl;
    }
}

void catalogtree::mkfile(string str,ll size=10){
    string thename=cd_change(str);
    if(cur->filetype==0){
        cout<<"same name or cur filetype not catalog"<<endl;
        return ;
    }
```

```
    auto ip=cur->child.find(thename);
    if(ip==cur->child.end()){
        node* temp=new node(0,thename,size);
        temp->parent=cur;
        node* t=cur;
        int check=0;
        while(t!=root){
            if(t->preadd(size)==false){
                check=1;
                break;
            }
            t=t->parent;
        }
        if(t->preadd(size)==false) check=1;
        if(check){
            cout<<"size full"<<endl;
        }
        else{
            node* tempt=cur;
            while(tempt!=root){
                temp->addsize(size);
                tempt=tempt->parent;
            }
            root->addsize(size);
            temp->parent=cur;
            cur->child.insert({thename,temp});
            cout<<"succeed"<<endl;
        }
    }
    else{
        cout<<"same name"<<endl;
    }
}

void catalogtree::deletestr(string str){
    string sss=cd_change(str);
    if(sss==""){
        node* pre=cur->parent;
        auto id=pre->child.find(cur->filename);
        pre->child.erase(id);
        cout<<"succeed"<<endl;
        cur=pre;
    }
    else{
        cout<<"faild"<<endl;
    }
```

```
}

int main(){
    catalogtree mulushu(100);
    mulushu.load();
    mulushu.save();
    string p;
    while(cin>>p){
        if(p=="cd"){
            if(cin.peek()!='\n'){
                cin>>p;
                if(p==".."){
                    mulushu.cd_back();
                }
                else{
                    mulushu.cd_change(p);
                }
            }
            else{
                mulushu.cd();
            }
        }
        if(p=="dir") mulushu.dir();
        if(p=="mkdir"){
            cin>>p;
            ll ss;
            cin>>ss;
            mulushu.mkdir(p,ss);
        }
        if(p=="mkfile"){
            cin>>p;
            ll ss;
            cin>>ss;
            mulushu.mkfile(p,ss);
        }
        if(p=="delete"){
            cin>>p;
            mulushu.deletestr(p);
        }
        cout<<"save? y or n"<<'\n';
        char c;
        cin>>c;
        if(c=='y') mulushu.save();
    }
}
```