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 课程实验报告

学号：202200130039	姓名：张向荣	班级： 22.3
上机学时：4 小时	实验日期：2024.4.22	

课程设计题目：
模拟文件目录系统

软件开发环境：
Dev-c++

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）。目录系统应支持如下基本操作：
(1)
dir ——列出当前目录下的所有子目录与文件项。
(2)
cd ——列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)，若存在则不进行任何操作。
(6)
mkfile str ——在(当前目录下)创建一个文件(名为 str) ，若存在则不进行任何操作。
(7)
delete str ——删除(当前目录下)名为 str 的目录或文件，若不存在则不进行任何操作。
(8)
save str—— 将从根节点开始的目录树结构保存到文件(名为 str)中。
(9)

load str —— 从文件 str 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
(10) quit —— 退出程序
选做功能
你可以自行充实目录树支持的指令，添加一些参数支持，这将有助于你对目录
系统的理解。提倡实现其他指令或带参数的现有指令。

1.3 输入说明
1. 输入有若干行，每行包含上述基本要求 3 的 10 条指令之一。
2. 数据文件 str 命名：[长度<=20 的随机字符串].tmp，随机字符串仅包含 a
z，不会出现其他字符。
输入界面设计
输入样例
1.4 输出说明
1. 根据输入指令按上述基本要求输出，没有则不输出。
2. “dir”命令的输出顺序按照字典序排列，先输出所有文件项再输出所有子
目录名，所有文件项后加“*”表示“这是一个文件项”。
输出界面设计
输出样例
2.分析与设计
2.1 问题分析
首先我们需要了解什么是目录树，目录树是一种数据结构，用于组织存储文件和目录，其结构类似于一棵倒挂的树。目录树中所有文件和目录从根目录开始，然后分支成不同的子目录和文件，这种结构使得文件和目录可以按照层次进行嵌套和组织，便于管理和访问。
2.2 主程序设计
首先打开输入文件，判断文件是否打开，如果未打开则重新输入。然后打开输出文件，通过输入文件中的指令引用相应的类成员函数。
2.3 设计思路
先定义一个结构体，并定义结构体成员表示节点。再定义一个目录树类，定义类成员函数实现目录树的相关功能，在主函数中根据指令调用类的相应函数。
2.4 数据及数据类型定义
在 CatalogTree.h 文件中首先定义目录树的节点结构体，其中包含目录名，节点指针类型的 vector 容器，表示当前目录的子目录，string 类型的 vector 容器存储当前目录的文件项，父节点指针，指向它的上一层目录。然后定义目录类。
2.5.算法设计及分析
在 CatalogTree.cpp 文件中，定义类成员函数，在构造函数中首先对目录树初始化，打开输出文件，将根节点文件名定义为空，当前节点从根节点开始。
在 Dir()函数中，利用迭代器将当前目录下的所有文件项和子目录输出。通过迭代器从 vector 中 begin()位置开始遍历至 end()，输出内容。
Cd1()函数表示列出当前目录的绝对路径，首先判断是否为根节点，若是则直接输出“/”，否则从当前节点开始，输出目录名，并指向其父节点，循环输出直到根节点。
Cd2（）函数表示当前目录变为当前目录的父目录，先判断当前节点是否为根节点，如果不是根节点，指向其父节点。
Cd3（）与 CD4（）函数表示 cd str 指令的两种情况。
第一种情况 cd 后面接的是当前节点的子目录，调用 Cd3()函数，传入要找的子目录名，使用迭代器遍历当前目录的子目录，若找到将当前节点变为找到的子目录所表示的路径目录。

第二种情况 cd 后面接的是一个新的路径，调用 Cd4() 函数，传入要找的目录的路径，字符串取第一个位置往后，也就是删去第一个“/”，当前节点从根节点开始。判断字符串中是否还存在'/'，如果不存在，说明只剩一个目录，循环直接结束，否则使用 s1 得到第一个'/'前的目录名，也就是父目录，s'表示'/'后的内容，遍历当前节点的孩子节点，直到找到 s1 目录名的节点。在循环结束后，表示只剩 1 个路径目录，从当前节点遍历子目录，直到找到对应的子目录，当前节点变为子目录节点。

Mkdir()函数表示在当前目录下创建一个子目录，传入要创建的目录名。首先遍历当前节点的子目录，如果目录名等于传入的目录名，表示该目录已经存在，直接返回，否则创建一个节点，目录名为传入的目录名，父节点为当前节点，使用 vector 中 push_back（）将其插入当前节点的子目录集合中，使用 sort 按照文件名的字典序排列。

Mkfile()函数表示在当前目录下创建一个文件，传入的参数表示要创建的文件名。与 Mkdir（）函数类似，首先通过迭代器遍历当前节点中的文件项，若找到传入的文件，则说明存在直接返回。否则将该字符串表示的文件名使用 vector 容器的函数 push_back（）插入 vector 中，按照文件名的字典序排列。

在 Delete()函数中，传入一个字符串，删除名为传入字符串的文件或子目录。首先使用迭代器遍历当前节点的子目录，如果找到字符串对应的子目录，使用 vector 容器中 erase()删除该子目录。同样使用迭代器遍历当前节点的文件项，若找到对应的文件，使用 erase()将其删除。

在 Save（）函数中，将从根节点开始的目录树结构保存到文件中，首先打开.tmp 文件，定义一个栈，从根节点开始遍历将非空子目录（及存在子目录或者文件的目录）压入栈中。并记录下每步操作的指令存入文件中。若是保存子目录，则用 Mkdir 指令，若是保存文件项，使用 Mkfile 指令，若是遍历子目录，使用 cd name(对应子目录名)，若是返回父节点，使用 cd ..。当栈不为空时开始循环，首先判断当前节点的孩子节点是否为栈顶，如果是则当前节点变为该栈顶节点，将该节点从栈中删除，记录下路径的改变，然后遍历该节点的子目录，将目录名、文件名指令都存入文件，将非空子目录压入栈中；如果当前节点的孩子节点不是栈顶元素，则返回当前节点的父节点，记录指令，直到栈空循环结束，存入“quit”指令。关闭文件。

在 Load()函数中，表示从文件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，在 save（）函数中记录下了每步操作的指令。在 Load（）函数中使用 getline（）读取文件中每行的指令，执行对应的操作，如果是“quit”表示退出；“cd ..”表示返回父节点，执行 Cd2()函数；若以'c'开头，执行 Cd3()函数；若是“Mkfile”执行 Mkfile()；若是“Mkdir”，执行 Mkdir（）。

选做部分：

实现了三个功能：返回根节点，重命名，搜索。

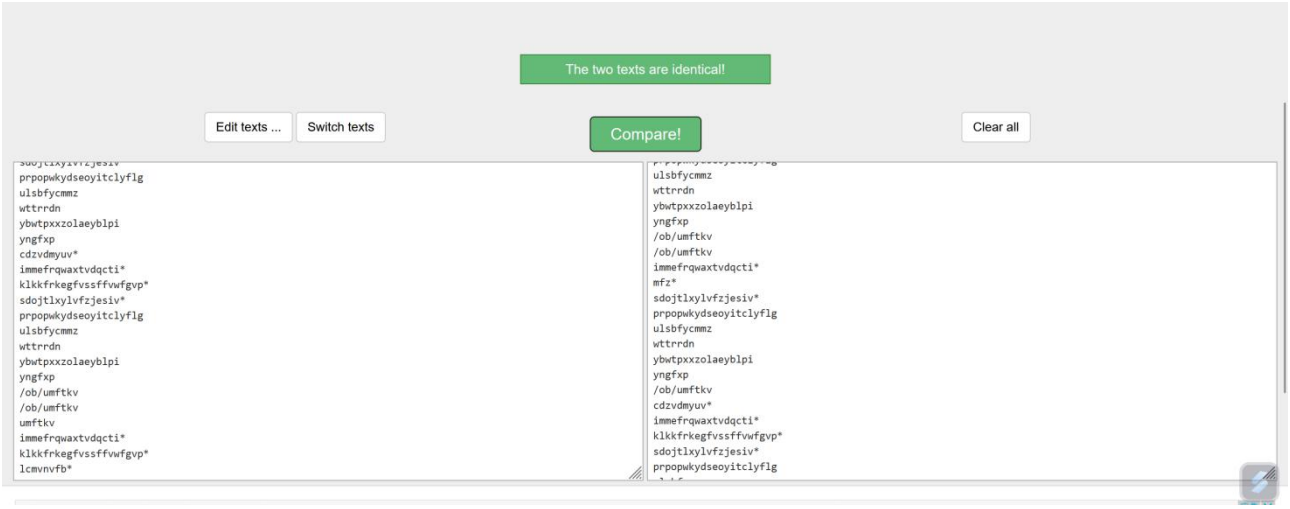
在 ReturnRoot()函数中，表示直接返回根节点。当前节点等于根节点。

在 Rename（）函数中，表示将文件项重命名。首先要找到需要修改的文件项的位置，传入要找的目录的路径，字符串取第一个位置往后，也就是删去第一个“/”，当前节点从根节点开始。判断字符串中是否还存在'/'，如果不存在，说明只剩一个目录，循环直接结束，否则使用 s1 得到第一个'/'前的目录名，也就是父目录，s'表示'/'后的内容，遍历当前节点的孩子节点，直到找到 s1 目录名的节点。在循环结束后，表示只剩 1 个路径目录，从当前节点遍历子目录，直到找到对应的子目录，当前节点变为子目录节点。定义一个节点表示父节点。将当前文件项名修改为新的文件名，并对父节点指向的所有子节点文件项重新按字典序排列。

在 Search()函数中，表示找到所有传入字符表示的子目录名与文件项。首先从根节点开始，将根节点存入栈中，当栈不为空时，开始循环。遍历栈顶节点的子目录，如果子目录名为传入字符串，则通过循环指向节点的父节点从而表示该子目录所在的路径，输出到文件中。

如果节点的子目录非空（存在子目录或文件项），则将其压入栈中。继续遍历节点的所有文件项，如果存在文件项名为传入的字符，则通过循环遍历节点的父节点从而表示出该节点的路径。

3. 测试



测试结果与样例相同。

4. 分析与探讨

在 Save()和 Load（）函数中，通过栈将目录树存到文件中，我们在输出路径时是向父节点寻找，我们是将一个目录中的所有内容遍历完，再遍历父节点的其他子目录，因此采用栈这种先进后出的数据结构。这种方法也减少了 Load()函数中的操作。

在代码中多处使用 string 中 substr（）函数选取字符串的某个部分。

5. 附录：实现源代码

```
#include"CatalogTree.h"


CatalogTree::CatalogTree(string s)
{
    outfile.open(s,ios::out);
    root=new Node("");
    now=root;
}

void CatalogTree::Dir() //利用迭代器将该目录下的所有子目录和文件输出
{
    for(vector<string>::iterator it=now->word.begin();it!=now->word.end();it++) //将 vector 中全部文件输出  string 类型
    {
        outfile<<*it<<'*'<<endl;
    }
    for(vector<Node *>::iterator ib=now->next.begin();ib!=now->next.end();ib++)//将 vector 中全部子目录输出  node*->name
    {
        outfile<<(*ib)->name<<endl;
```

```
    }
}

void CatalogTree::Cd1()//列出当前目录的绝对路径
{
    if(now==root)
    {
        outfile<<'/'<<endl;
         return ;
    }
     string s="";
     Node* p=now;
    while(p!=root)//如果 p 不是根节点，则不断将 p=p->father 并且将 p 的 name 输出
    {
        s="/"+p->name+s;
        p=p->father; //向上寻找
    }
    outfile<<s<<endl;
}


void CatalogTree::Cd2()//当前目录变为当前目录的父目录
{
    if(now==root)
       return ;
    now=now->father;//返回其父目录
}

void CatalogTree::Cd3(string s)//s 为该目录的一个子目录
{
    for(vector<Node*>::iterator it=now->next.begin();it!=now->next.end();it++)
    {
        if((*it)->name==s)
        {
            now=*it;//当前目录变成 str 所表示的路径的的目录
            break;
        }
    }
}

Node *CatalogTree::find(string s)//找到对应目录的位置
{
    int u=s.length();
    s=s.substr(1,u);//从/后开始
    Node *p=root;
```

```
    while(1)
    {
        if(s.rfind('/')==string::npos)//s 中是否存在'/'，逆向查找
          break;//没有子目录
        int op=s.length();
        string s1=s.substr(0,s.find('/',0));//得到该目录下一个子目录名称       /前的内容
        s=s.substr(s.find('/',0)+1,op); //s 为/后的内容

        for(vector<Node*>::iterator it=p->next.begin();it!=p->next.end();it++)
        {
            if((*it)->name==s1)//找到对应的 p 的子目录
            {
                p=*it;
                break;
            }
        }
    }
    //     /a/b/c
    //   a    b/c       b c
    for(vector<Node*>::iterator it=p->next.begin();it!=p->next.end();it++)
    {
        if((*it)->name==s)//找到要求的最后一个子目录对应的位置
        {
            p=*it;
            return p;
            //break;
        }
    }
    return NULL;


}

void CatalogTree::Cd4(string s)
{
    if(find(s))
    {
        now=find(s);
    }

}

void CatalogTree::Mkdir(string s)//在当前目录下创建一个子目录
{
    for(vector<Node *>::iterator it=now->next.begin();it!=now->next.end();it++)
    {
```

```
        if((*it)->name==s)
        return ;//存在直接返回
    }
    //不存在
    Node *p=new Node(s,now);//创建目录，父节点 w 为 now
    now->next.push_back(p);
    sort(now->next.begin(),now->next.end(),judge);//排序

}


void CatalogTree::Mkfile(string s)//在当前目录下创建一个文件
{
    for(vector<string>::iterator ib=now->word.begin();ib!=now->word.end();ib++)
    {
        if((*ib)==s)
         return ;//存在直接返回
    }
    now->word.push_back(s);//不存在  存入
    sort(now->word.begin(),now->word.end());//按字典序排序
 }


void CatalogTree::Delete(string s)//删除当前目录下名为 s 的文件或目录
{
    for(vector<Node *>::iterator it=now->next.begin();it!=now->next.end();it++)
    {
        if((*it)->name==s)
        {
            it=now->next.erase(it);
            break;
        }
    }

    for(vector<string>::iterator ib=now->word.begin();ib!=now->word.end();ib++)
    {
        if((*ib)==s)
        {
            ib=now->word.erase(ib);
            break;
        }
    }
}


void CatalogTree::Save(string s)//将从根节点开始的目录树结构保存到文件中
```

```
{
    ofstream out(s+".tmp",ios::out);//打开文件
    stack<Node *>S;//要将一条目录输出到底，所以用栈
    for(vector<Node*>::iterator it=root->next.begin();it!=root->next.end();it++)
    {
        out<<"mkdir "<<(*it)->name<<endl;
        if((*it)->next.size()!=0||(*it)->word.size()!=0)
        {
            S.push((*it));
        }
    }

    for(vector<string>::iterator ib=root->word.begin();ib!=root->word.end();ib++)
    {
        out<<"mkfile "<<(*ib)<<endl;
    }


    Node *t=root;
    while(!S.empty())//一层一层存储
    {
        bool b=false;
        for(vector<Node *>::iterator it=t->next.begin();it!=t->next.end();it++)
        {
            if((*it)==S.top())
            {
                t=(*it);
                b=true;
                break;
            }
        }

        if(b==false)
        {
            t=t->father;
            out<<"cd .."<<endl;
        }
        else
        {
            Node* p=S.top();
            S.pop();
            bool b=false;
            out<<"cd "<<p->name<<endl;
            for(vector<Node*>::iterator it=p->next.begin();it!=p->next.end();it++)
            {
                out<<"mkdir "<<(*it)->name<<endl;
```

```
                    if((*it)->next.size()!=0||(*it)->word.size()!=0)
                        S.push((*it));
                }
                for(vector<string>::iterator ib=p->word.begin();ib!=p->word.end();ib++)
                {
                    out<<"mkfile "<<(*ib)<<endl;
                }
            }
        }

        out<<"quit"<<endl;
        out.close();

}


void CatalogTree::Load(string s)//从文件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
{
    root=new Node("");
    now=root;
    ifstream in(s+".tmp",ios::in);
    while(1)
    {
        string s;
        getline(in,s);
        if(s=="quit")
           return ;
        else if(s=="cd ..")
        {
            Cd2();
        }
        else if(s[0]=='c')
        {
            int u=s.length();
            s=s.substr(3,u);
            Cd3(s);
        }
        else if(s[0]=='m'&&s[2]=='d')
        {
            int u=s.length();
            s=s.substr(6,u);
            Mkdir(s);
        }
        else if(s[0]=='m'&&s[2]=='f')
        {
```

```
                int u=s.length();
                s=s.substr(7,u);
                Mkfile(s);
            }
        }
        now=root;
        in.close();

    }

void CatalogTree::Search(string s)//包含 s 目录名或文件名的路径输出
{
    Node *p=root;
    Node *q=NULL;
    queue<Node *>Q;
    Q.push(p);
    while(!Q.empty())
    {
        Node* node=Q.front();
        Q.pop();
        for(vector<Node *>::iterator it=node->next.begin();it!=node->next.end();it++)
        {
            if((*it)->name==s)
            {
                q=node;
                if(q==root)
                {
                    outfile<<'/'<<endl;
                }
                else
                  {
                    string s="";
                    while(q!=root)
                        {
                         s="/"+q->name+s;
                         q=q->father;
                    }
                    outfile<<s<<endl;//将目录名输出
                }
            }

            if((*it)->next.size()!=0||(*it)->word.size()!=0)//存子目录
              Q.push((*it));
        }//所有包含 s 目录名的路径输出

        for(vector<string>::iterator ib=node->word.begin();ib!=node->word.end();ib++)
```

```
        {
            if((*ib)==s)//文件名为 s
            {
                q=node;
                if(q==root)
                {
                    outfile<<"/*"<<endl;
                }
                else
                {
                    string s="*";
                    while(q!=root)
                    {
                        s="/"+q->name+s;
                        q=q->father;
                    }
                    outfile<<s<<endl;
                }
            }
        }//包含 s 为文件名的路径名输出

    }
}


void CatalogTree::ReturnRoot()
{
    now=root;//返回根节点
}

void CatalogTree::Rename(string s1,string s2)//将目录名 s1 改成 s2
{
    if(find(s1))
    {
     now=find(s1);
     if(now==root)
        return ;

        Node *q=now->father;

         now->name=s2;
         sort(q->next.begin(),q->next.end(),judge);//重新排序
         cout<<now->name<<endl;
    }
```

```
}
```