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：202300130207 | 姓名：李臣思 | | 班级：23.5 |
| 实验题目：模拟文件目录系统 | | | |
| 实验学时：6 | | 实验日期：25.4.21 | |
| 实验目的：设计并实现目录树的ADT，表达字符串集合组成的有序树。应用CatalogTree结构设计并实现一文件目录系统模拟程序，提供模拟操作界面。 | | | |
| 软件开发工具：  Visual Studio Code | | | |
| 1. 实验内容   设计并实现目录树**CatalogTree的ADT**，用来表达**字符串组成的有序树**。对于目录系统，每个节点存放的数据内容为字符串，每个节点对应一个目录项（目录/文件）；从genie带你到该节点路径素有节点的字符串用“/”组合后的路径即为就绝对路径。实现一文件目录系统的**模拟程序**，并提供**模拟操作界面。**对于此目录系统应实现以下基本操作：  ① dir ——列出当前目录下的**所有目录项**  ② cd ——打出当前目录的**绝对路径**  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  基本要求：  (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**。   1. 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） 2. 数据结构：     图 1  如图，每一个CatalogTree节点node中包含一个指向父节点的指针(parent)；如果是目录的话，指向他的第一个子目录项的节点(leftchild)；以及指向该目录项的同级的下一个目录项节点(brother)；表明该目录项类别的filetype(1是目录，0是文件)；以及该目录项的名称(filename)；另外包括一个node构造函数。    图 2  此外在catalogtree类中首先包含私有成员node节点root以及cur，分别代表当前目录树的根节点以及当前系统进行时执行命令的节点位置；getAbsolutePath用来获得某一结点的绝对路径表达。  公有成员构造函数catalogtree初始化目录树，cleartree协助实现析构函数~catalogtree清空目录树；splitpath判断并分割输入的路径；findpath返回想要找到的节点在目录树中的节点位置；dir、cd等分别实现对应的功能；saveToFile以及loadFromFile分别实现将当前的目录保存在对应文件以及从对应文件中读取目录系统；getcurrentpath返回当前的路径，用于显示系统模拟界面。   1. 算法描述   getAbsolutePath(node\* node)：获得某一节点node的绝对路径，如果该节点node即为根节点，则直接返回“/”即可；否则，从该节点node开始向上寻找每一个父节点，并将每一个父节点parent放入栈中，随后开始出栈，后入栈的先出栈，则最终可以得到绝对路径。  **构造函数catalogtree()：**构造初始化根节点root的各类定义；cur=root。  **cleartree(node\* node1)**：如果node1为空，那么直接结束函数，否则递归调用函数cleartree(node1->leftchild);cleartree(node1->brother);删除node1.  **析构函数~catalogtree()：**调用cleartree(root);删除目录树。  **queue<string> splitpath(const string& path)：**首先判断是否为绝对路径，如果是绝对路径，那么路径开头会存在“/”,否则则不是绝对路径；如果是绝对路径，在队列初压入“”，用来表明是一个绝对路径，并且根据“/”将路径path分割成若干个字符串，分别压入队列中；如果是相对路径，那么直接将path压入队列中。最后返回队列。  **node\* findpath(queue<string>& parts)**：根据通过splitpath分割得到的parts，如果是绝对路径（即队列头存在“”）,那么从根节点开始查找；如果不是，则从当前节点cur开始查找。当队列不为空时，每次都从节点的左孩子开始查找，依次从该节点向brother查找，如果名称等于队首名称，则从该节点向下寻找，弹出队首。循环往复，直到队列为空，返回查找到的节点。  **功能(1)dir()：**首先判断当前节点的类型，如果不是目录，直接返回并报错；反之，继续向下，从当前节点cur的左孩子开始，向brother遍历，输出每个节点的名称以及类型。  **功能(2)-(4)cd(const string& path)：**首先判断将要执行的命令，如果输入的path为空，那么即要执行cd ——打出当前目录的**绝对路径，**直接执行同getAbsolutePath函数一样的逻辑输出绝对路径；如果输入的path为“..”，那么就要执行命令cd ..——当前目录变为当前目录的父目录，那么就将cur指向cur的父节点parent即可；除此之外，则path输入的即为将要进入的目录项的相对/绝对路径，那么根据splitpath(path)，分割path并且判断是绝对路径还是相对路径，再直接执行findpath找到该路径所指的位置，更新cur位置。  **功能(5)mkdir(const string& path)**：首先同样需要判断是绝对路径/相对路径，执行splitpath函数，但是与cd操作不同的是不能直接调用findpath找到对应的节点，而是要去掉队末的文件名再执行findpath，所以这里遍历整个队列将除队尾的元素外其他元素从新放到另一个队列中，而后判断如果是绝对路径就执行findpath；反之，直接指向cur。而后遍历这个节点的所有孩子，存在重名则返回错误；或者该节点不是目录，同样返回错误。最终创建一个新的节点放到将要插入的节点的leftchild处，brother指向原leftchild。  **功能(6)mkfile(const string& path)**：同mkdir逻辑一致，只是在最后创建目录项时，将新创建的目录项节点的类型改为0（文件）。  **功能(7)delete1(const string& path)**：前半部分逻辑与mkdir一致。在找到准备删除的目录项所在的目录位置后，则从这个目录的子节点中找到将要删除的节点的前一个节点位置，删除这个节点，如果这个将要删除的节点即为该目录的leftchild，那么直接指定leftchild为原leftchild的写一个brother。  **文件操作：**保存saveToFile(const string& filename)，遍历目录树中的所有节点，getAbsolutePath得到绝对路径并放入对应的文件中。loadFromFile(const string& filename)，打开文件，按行读取各个目录项的绝对路径，根据不同的类别分别执行mkdir和mkfile。  每次会读取一个命令（cd/mkdir/mkfile/save等），与一个跟在命令后的一个字符串（路径或为空），根据命令，分别执行对应再catalogtree中的函数，并且输入对应的参数。   1. 测试结果（测试输入，测试输出）     图 3  如图，最后save之后可以在out.txt中看到    图 4  而后再将这个load进系统中：    图 5   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   结果符合预期。在图3的运行中，首先在根节点中执行mkdir与mkfile，再进入到path1中，执行mkdir与mkfile，并且这里/path1> mkdir path1；/path1> mkdir /path1/path2  均成功，说明mkdir中区分绝对路径与相对路径成功。而后再次尝试delete分别输入相对路径与绝对路径，说明delete区分绝对路径与相对路径成功。同样cd与mkfile区分绝对路径与相对路径成功。  最后将该目录系统存入文件out.txt中，得到图5中的内容，与预期的一致。  再次打开运行系统，load进out.txt，通过dir验证，与out中对应的目录项一致。   1. 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<string>  #include<sstream>  #include<stack>  #include<queue>  #include<fstream>  using namespace std;  typedef long long ll;  //文件系统节点结构体  struct node  {  node\* parent;//父节点指针（指向上一级目录）  node\* leftchild;//左孩子（指向第一个子节点）  node\* brother;//兄弟节点指针9指向同级的下一个节点）  int filetype;//1是目录，0是文件  string filename;//目录/文件名  node(node\* p=NULL,node\* left=NULL,node\* bro=NULL,int type=1,string name=""):  parent(p),leftchild(left),brother(bro),filetype(type),filename(name){}  };  class catalogtree{//CatalogTree  private:  node\* root;//根节点  node\* cur;//当前目录项指针  string getAbsolutePath(node\* node) {//得到node的绝对路径  if (node == root) return "/";  stack<string> pathStack;  while (node != root) {  pathStack.push(node->filename);  node = node->parent;  }  string path;  while (!pathStack.empty()) {  path += "/" + pathStack.top();  pathStack.pop();  }  return path.empty() ? "/" : path;  }  public:  catalogtree(){  root=new node();//初始化根节点  root->filename="";//名为空  root->filetype=1;  cur=root;//当前目录指针指向根节点  }  void cleartree(node\* node1){  if(!node1)  return;  cleartree(node1->leftchild);//递归删除左子树  cleartree(node1->brother);//递归删除兄弟子树  delete node1;//释放当前节点  }  ~catalogtree(){  cleartree(root);//释放整个目录树的内容  }  queue<string> splitpath(const string& path){  queue<string> parts;  bool isabsolute=false;  if(!path.empty()&&path[0]=='/'){//判断是否为绝对路径  isabsolute=true;  }  int i=0;  if(path[0]=='/'){  i=1;//跳过绝对路径开头的“/”  }  string current;  for(i;i<path.size();i++){//遍历path，按照“/”分割  if(path[i]=='/'){  if(!current.empty()){  parts.push(current);  current="";  }  }  else{  current+=path[i];  }  }  if(!current.empty()){//将最后一个没有压入的current压入  parts.push(current);  }  if(isabsolute==1){//如果为绝对路径  queue<string> absparts;  absparts.push("");//在开头压入一个空字符串，标记其为绝对路径  while(!parts.empty()){  absparts.push(parts.front());  parts.pop();  }  return absparts;  }  return parts;//不是绝对路径则直接返回  }  node\* findpath(queue<string>& parts);//根据路径队列确定目标节点的位置  void dir();//列出当前目录下的所有子项  void cd(const string& path);//实现功能2-4  bool mkdir(const string& name);//创建目录  bool mkfile(const string& name);//创建文件  bool delete1(const string& name);//删除指定目录/文件  void saveToFile(const string& filename);//保存到文件中  void loadFromFile(const string& filename);//从文件中读取目录结构  string getcurrentpath(){//用于系统的模拟显示  return getAbsolutePath(cur);  }  };  node\* catalogtree::findpath(queue<string>& parts){  node\* cur1;  if(parts.front()==""){//是绝对路径，从根节点开始  cur1=root;  parts.pop();  }  else{//相对路径从当前节点开始  cur1=cur;  }  while(!parts.empty()&&cur1!=nullptr){//开始遍历找寻  string part=parts.front();  parts.pop();  if(cur1->filetype==0){//是文件，返回  return nullptr;  }  node\* child=cur1->leftchild;//指向左孩子  while(child!=nullptr&&child->filename!=part)//遍历所有孩子，直到找到对用的孩子  child=child->brother;  if(child==nullptr) return nullptr;//没有找到匹配项  cur1=child;//指向该孩子，继续向下找，直到队列为空  }  return cur1;//返回节点  }  void catalogtree::dir(){  if(cur->filetype==0){//当前节点是文件  cout<<"不是一个文件夹"<<endl;  return;  }  node\* child=cur->leftchild;  while(child!=nullptr){//遍历所有孩子节点  cout << child->filename << (child->filetype ? " (dir)" : " (file)") << endl; // 显示所有子节点以及类别  child=child->brother;  }  }  void catalogtree::cd(const string& path){  if(path.empty()){//输出当前的绝对路径  if(cur==root){  cout<<"/";  }  else{  node\* cur1=cur;  stack<string> path1;  while(cur1!=root){  path1.push(cur1->filename);  cur1=cur1->parent;  }  cout<<'/';  while(!path1.empty()){  cout<<path1.top()<<"/";  path1.pop();  }  }  cout<<endl;  return;  }  else if(path==".."){//更新cur指向cur的父节点  if(cur!=root){  cur=cur->parent;  }  return;  }  else{  queue<string> parts=splitpath(path);//解析路径  node\* target=findpath(parts);//找到将要切换的位置  if(target!=nullptr&&target->filetype==1){  cur=target;//指向该位置  }  else{  cout<<"没有该文件"<<endl;  }  }  }  bool catalogtree::mkdir(const string& path){  queue<string> parts1=splitpath(path);//分析路径  queue<string> parts;  while(parts1.size()>1){//如果是绝对路径，那么长度一定大于2  //将除了将要创建目录项的名称之外的所有节点名再放入parts中  parts.push(parts1.front());  parts1.pop();  }  string name=parts1.front();//将要创建的目录名  //parts.push(parts1.front());  node\* target;  if(parts.empty()){//如果是相对路径  target=cur;//从当前的节点开始  }  else{  target=findpath(parts);//找到相对路径中的位置  }  //排除错误情况  if(!target){  cout<<"该路径不存在"<<endl;  }  if(target->filetype==0){  cout<<"应在目录下创建目录"<<endl;  return false;  }  node\* child=target->leftchild;//遍历孩子，避免出现重名  while(child!=nullptr){  if(child->filename==name){  cout<<"存在重名"<<endl;  return false;  }  child=child->brother;  }  // 创建新节点并更新链表  node\* newnode=new node(target, nullptr, target->leftchild, 1, name);  target->leftchild = newnode;  return true;  }  bool catalogtree::mkfile(const string& path){//与mkdir基本一致  queue<string> parts1=splitpath(path);  queue<string> parts;  while(parts1.size()>1){  parts.push(parts1.front());  parts1.pop();  }  string name=parts1.front();  //parts.push(parts1.front());  node\* target;  if(parts.empty()){  target=cur;  }  else{  target=findpath(parts);  }  if(!target){  cout<<"该路径不存在"<<endl;  }  if(target->filetype==0){  cout<<"应在目录下创建文件"<<endl;  return false;  }  node\* child=target->leftchild;  while(child!=nullptr){  if(child->filename==name){  cout<<"存在重名"<<endl;  return false;  }  child=child->brother;  }  node\* newnode =new node(target,nullptr,target->leftchild,0,name);  target->leftchild=newnode;  return true;  }  bool catalogtree::delete1(const string& path){  queue<string> parts1=splitpath(path);  queue<string> parts;  while(parts1.size()>1){  parts.push(parts1.front());  parts1.pop();  }  string name=parts1.front();  //parts.push(parts1.front());  node\* target;  if(parts.empty()){  target=cur;  }  else{  target=findpath(parts);  }  if(!target){  cout<<"该路径不存在"<<endl;  }  if(target->filetype==0){  cout<<"应在目录下删除"<<endl;  return false;  }//同样是找到对应的目录位置  node\* prev=nullptr;  node\* child=target->leftchild;  while(child!=nullptr&&child->filename!=name){//找到将要删除的节点的位置  prev=child;//指向将要删除节点的前一个节点  child=child->brother;  }  if(child==nullptr)  {  cout<<"不存在该文件或者目录"<<endl;  return false;  }  if(prev==nullptr){//即将要删除的节点位置恰好为父节点的第一个子节点  target->leftchild=child->brother;  }  else{  prev->brother=child->brother;  }  cleartree(child->leftchild);//删除对应的子树  delete child;  return true;  }  void catalogtree::saveToFile(const string& filename) {  ofstream file(filename);  stack<node\*> s;  s.push(root);  while (!s.empty()) {  node\* node1 = s.top();  s.pop();    // 跳过根节点，写入其他节点的绝对路径和类型  if (node1 != root) {  file << getAbsolutePath(node1) << " " << (node1->filetype ? "dir" : "file") << endl;  }  // 将子节点逆序压入栈（保证加载顺序正确）  stack<node\*> temp;  node\* child = node1->leftchild;  while (child != nullptr) {  temp.push(child);  child = child->brother;  }  while (!temp.empty()) {  s.push(temp.top());  temp.pop();  }  }  file.close();  }  void catalogtree::loadFromFile(const string& filename) {  ifstream file(filename);  cleartree(root);//清空现有目录树  root = new node();//重建根节点  root->filetype = true;  cur = root;  string line;  while (getline(file, line)) {  istringstream iss(line);  string path, type;  iss >> path >> type;  if (path == "/") continue;//跳过根节点  if(type=="dir"){mkdir(path);}//直接调用mkdir/mkfile重建目录结构  if(type=="file"){mkfile(path);}  }  file.close();  }  //拆分命令行输入为令牌队列  queue<string> splitCommand(const string& line) {  queue<string> tokens;  istringstream iss(line);  string token;  while (iss >> token)  tokens.push(token);  return tokens;  }  int main() {  catalogtree fs;  string commandLine;  while (true) {  cout <<fs.getcurrentpath()<<"> ";//显示当前路径以及提示符  getline(cin, commandLine);  queue<string> tokens = splitCommand(commandLine);  if (tokens.empty()) continue;  string cmd = tokens.front();//命令  tokens.pop();  if (cmd == "quit") break;  else if (cmd == "dir") fs.dir();  else if (cmd == "cd") {  string arg = tokens.empty() ? "" : tokens.front();  fs.cd(arg);  } else if (cmd == "mkdir") {  if (tokens.empty()) cout << "Missing argument" << endl;  else fs.mkdir(tokens.front());  } else if (cmd == "mkfile") {  if (tokens.empty()) cout << "Missing argument" << endl;  else fs.mkfile(tokens.front());  } else if (cmd == "delete") {  if (tokens.empty()) cout << "Missing argument" << endl;  else fs.delete1(tokens.front());  } else if (cmd == "save") {  if (tokens.empty()) cout << "Missing argument" << endl;  else fs.saveToFile(tokens.front());  } else if (cmd == "load") {  if (tokens.empty()) cout << "Missing argument" << endl;  else fs.loadFromFile(tokens.front());  }else {  cout << "Unknown command" << endl;  }  }  return 0;  } | | | |