**《程序设计基础(2)》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2020级计算机类01班** | | | **姓名** | **孙莹莹** |
| **实验题目** | **命令行文件管理器的设计与实现** | | | | | |
| **实验时间** | **2021.4.22** | | **实验地点** | **DS1408** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 ■综合性** | | |
| 教师评价：  □算法/实验过程正确；□源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  运用面向对象程序设计思想，基于Linux文件管理和C＋＋的I/O框架，实现命令行下的文件管理器。 | | | | | | |
| 二、实验项目内容  1、实现文件夹创建、删除、进入。  2、实现当前文件夹下的内容罗列。  3、实现文件拷贝和文件夹拷贝（文件夹拷贝指深度拷贝，包括所有子目录和文件）。 | | | | | | |
| 1. 实验过程或算法（源程序）   #include <iostream>  #include <iomanip>  #include <fstream>  #include <sstream>  #include <vector>  #include <string>  #include <sys/types.h>  #include <sys/stat.h>  #include <unistd.h>  #include <dirent.h>  using namespace std;  class godd  {  public:      virtual void doit() = 0;  };  class modd : public godd    //创建文件夹  {  private:      vector<string> mycommandArgs;  public:      modd(vector<string> commandArgs)      {          this->mycommandArgs = commandArgs;      }      virtual void doit()      {          if (mycommandArgs.size() < 2)          {              cout << "error: please enter the name of the folder" << endl;          }          else          {              string dirName = mycommandArgs[1];              mode\_t mode = 0775;              if (mkdir(dirName.c\_str()) == 0)              {                  cout << "succeed in creating the folder‘" << dirName << "’" << endl;              }              else              {                  cout << "error:fail in creating the folder‘" << dirName << "’" << endl;              }          }      }  };  class rodd : public godd     //删除文件夹  {  private:      vector<string> mycommandArgs;  public:      rodd(vector<string> commandArgs)      {          this->mycommandArgs = commandArgs;      }      virtual void doit()      {          if (mycommandArgs.size() < 2)          {              cout << "error:please enter the name of the folder" << endl;          }          else          {              string dirName = mycommandArgs[1];              if (rmdir(dirName.c\_str()) == 0)              {                  cout << "succeed in deleting the folder‘" << dirName << "’" << endl;              }              else              {                  cout << "error:fail in deleting the folder‘" << dirName << "’" << endl;              }          }      }  };  class roff : public godd   //删除文件  {  private:      vector<string> mycommandArgs;  public:      roff(vector<string> commandArgs)      {          this->mycommandArgs = commandArgs;      }      virtual void doit()      {          if (mycommandArgs.size() < 2)          {              cout << "error: please enter the name of the file" << endl;          }          else          {              string fileName = mycommandArgs[1];              if (remove(fileName.c\_str()) == 0)              {                  cout << "succeed in deleting the file‘" << fileName << "’" << endl;              }              else              {                  cout << "error: fail in deleting the file‘" << fileName << "’" << endl;              }          }      }  };  class codd : public godd      //更改目录  {  private:      vector<string> mycommandArgs;  public:      codd(vector<string> commandArgs)      {          this->mycommandArgs = commandArgs;      }      virtual void doit()      {          if (mycommandArgs.size() < 2)          {              cout << "error: please enter the name of the folder" << endl;          }          else          {              string dirName = mycommandArgs[1];              if (chdir(dirName.c\_str()) == 0)              {                  char curDir[300];                  getcwd(curDir, 300); // getcwd()会将当前工作目录的绝对路径复制到参数curDir所指的内存空间中。                  cout << "has change the working catalogue to： "<< curDir << endl;              }              else              {                  cout << "error: failure in changeing the working catalogue" << endl;              }          }      }  };  class lodd : public godd  //列出文件夹内容  {  private:      vector<string> mycommandArgs;  public:      lodd(vector<string> commandArgs)      {          this->mycommandArgs = commandArgs;      }      virtual void doit()      {          DIR \*curDir;          struct dirent \*ent; //目录项结构体          struct stat entStat; //文件属性结构体          if ((curDir = opendir(".")) == NULL)          {              cout << "error: failure in opening the folder" << endl;          }          else          {              while ((ent = readdir(curDir)) != NULL) //读取目录函数              {                  if (stat(ent->d\_name, &entStat) == 0)    //定义函数：int stat(const char \* file\_name, struct stat \*buf);                  {                                         // 函数说明：stat()用来将参数file\_name 所指的文件状态, 复制到参数buf 所指的结构中。                      if (entStat.st\_mode & S\_IFDIR)        //S\_IFDIR判断一个路径是否为目录                      {                          cout << setw(6) << left << "d";                          cout << right << setw(10) << entStat.st\_size << left << setw(8) << " bytes";                          cout << ent->d\_name << "/" << endl;                      }                      else if (entStat.st\_mode & S\_IFREG)                      {                          cout << setw(6) << left << "-";                          cout << right << setw(10) << entStat.st\_size << left << setw(8) << " bytes";                          cout << ent->d\_name << endl;                      }                  }              }              closedir(curDir);          }      }  };  class copp : public godd  {  private:      vector<string> mycommandArgs;  public:      copp(vector<string> commandArgs)      {          this->mycommandArgs = commandArgs;      }      virtual void doit()      {          if (mycommandArgs.size() < 3)          {              cout << "error: please enter the name of the source document/folder and the name of the target document/folder. " << endl;          }          else          {              string sourceFileName = mycommandArgs[1];              string targetFileName = mycommandArgs[2];              ifstream sourceFile(sourceFileName.c\_str(), ios::binary);              ofstream targetFile(targetFileName.c\_str(), ios::binary);              targetFile << sourceFile.rdbuf();    // rdbuf()可以实现一个流对象指向的内容用另一个流对象来输出              targetFile.close();              sourceFile.close();              cout << "copy successfully" << endl;          }      }  };  vector<string> split\_str(string s)  {      vector<string> tokens;      istringstream iss(s);      //istringstream类用于执行C++风格的串流的输入操作。      do                             //istringstream::istringstream(string str);它的作用是从string对象str中读取字符。      {          string sub;          iss >> sub;          tokens.push\_back(sub);      } while (iss);      tokens.erase(tokens.end() - 1); // 删除最后的换行回车符      return tokens;  }  void handle(vector<string> commandss);  void show();  int main()  {      show();      string command;      vector<string> commandArgs;      getline(cin, command);      commandArgs = split\_str(command);      while (commandArgs.size() > 0 && "q" != commandArgs[0])      {          handle(commandArgs);          cout << "?";          getline(cin, command);          commandArgs = split\_str(command);      }      return 0;  }  void handle(vector<string> commandArgs)  {      if ("rf" == commandArgs[0])      {          roff p(commandArgs);          godd \*q = &p;          q->doit();      }      else if ("md" == commandArgs[0])      {          modd p(commandArgs);          godd \*q = &p;          q->doit();      }      else if ("rd" == commandArgs[0])      {          rodd p(commandArgs);          godd \*q = &p;          q->doit();      }      else if ("ld" == commandArgs[0])      {          lodd p(commandArgs);          godd \*q = &p;          q->doit();      }      else if ("cd" == commandArgs[0])      {          codd p(commandArgs);          godd \*q = &p;          q->doit();      }      else if ("cp" == commandArgs[0])      {          copp p(commandArgs);          godd \*q = &p;          q->doit();      }      else      {          cout << "error: flase command。" << endl;          return;      };  }  void show()  {     cout << "===================================" << endl;      cout << " welcome to the Doszip Commander   ver: 1 by LUY" << endl;      cout << "==============================================" << endl;      cout << "plese enter with： ? xx1 xx2 …… xxN" << endl;      cout << "   ? means what you want to do，xx1~xxN means the object you want to work with" << endl;      cout << "\ncommands are as followed：" << endl;      cout << "   md  create a folder             rd  delete a folder" << endl;      cout << "   cd  change the working folder   ld  list the folder" << endl;      cout << "   cp  copy the document/folder    rf  delete the document" << endl;      cout << "   q   quit\n"           << endl;      cout << "?";  } | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程 2. 正确的输入   SKZDG)`K~DFNZRBGN3LPOMD  CF3U]~`I}7V@5M}L83){3E8]B2PCU$VU]}K_RWGGKAKHD5   1. 错误的输入（会显示出错）   UL`S%24IKL@[(]`$B9~6UZ1  3.实验结果分析：  该算法的设计依赖于对每个命令创建一个类来实现该命令，如rf指令创建一个roff类去实现它，而这些类都属于同一个基类godd,其中有一个虚方法doit(),handle函数根据命令的不同赋值给g\*,然后调用g的doit()去实现这个命令。  这个算法的好处就是当需要添加新的命令时只需要再添加一个类，相应地更改handle()函数即可，如果需要更改某一个命令的具体是西安，只需要更改相应的类即可。 | | | | | | |