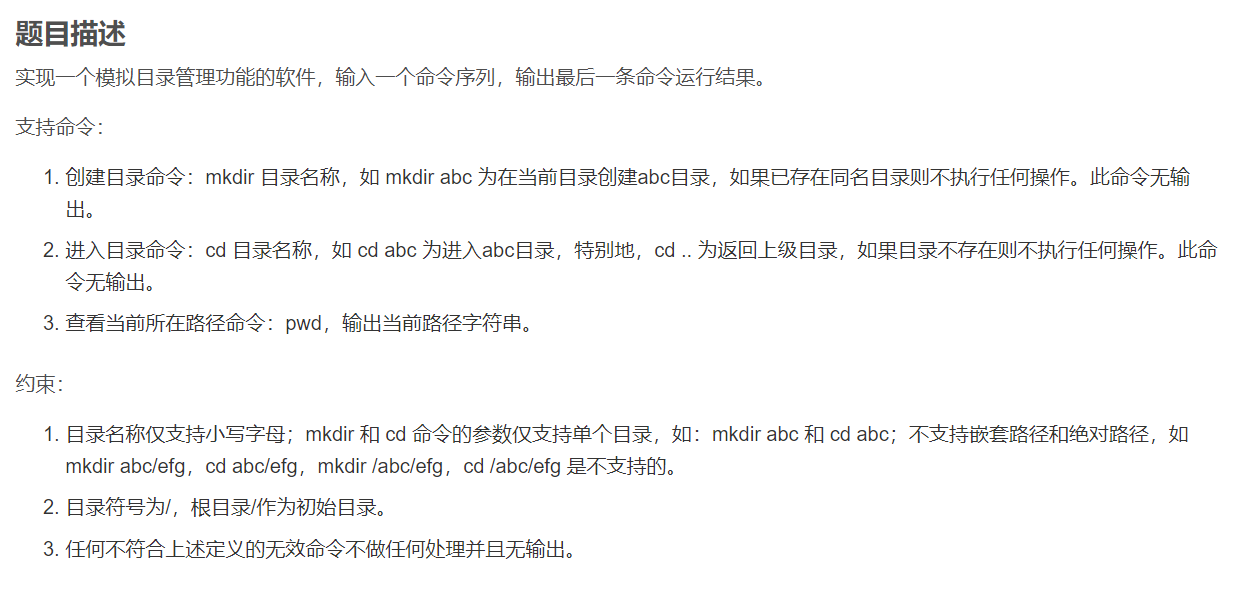
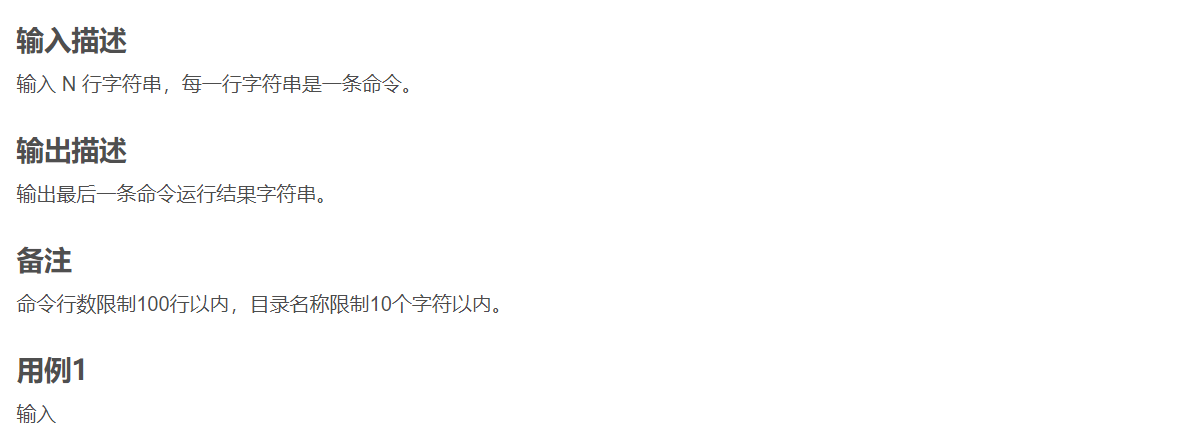
# **E卷-模拟目录管理[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go ）**





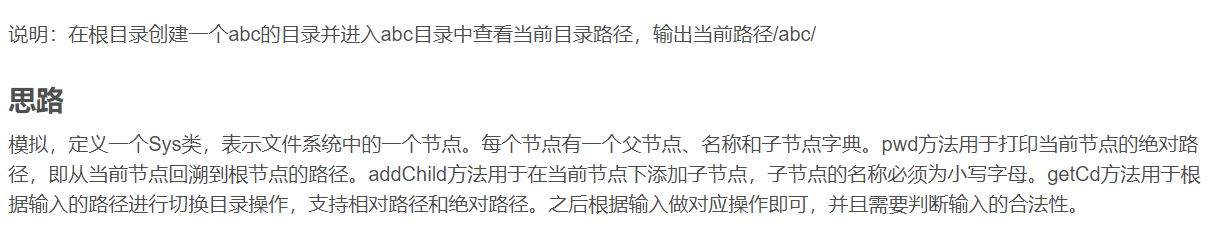
mkdir abc

cd abc

Pwd



/abc/





import java.util.\*;

class Sys {

Sys par; // 父目录

String x; // 当前目录名称

Map<String, Sys> son; // 子目录映射

Sys(String x) {

this.par = null; // 初始化父目录为null

this.x = x; // 设置当前目录名称

this.son = new HashMap<>(); // 初始化子目录映射

}

void pwd() {

List<String> res = new ArrayList<>(); // 用于存储路径的列表

Sys now = this; // 从当前目录开始

while (now.par != null) { // 循环直到根目录

res.add(now.x); // 将当前目录名加入结果列表

now = now.par; // 向上移动到父目录

}

Collections.reverse(res); // 反转列表以得到正确的路径顺序

if (res.isEmpty()) { // 如果结果列表为空，表示在根目录

System.out.println("/"); // 输出根目录

} else {

System.out.println("/" + String.join("/", res) + "/"); // 输出完整路径

}

}

void addChild(String c) {

for (char x : c.toCharArray()) {

if (x < 'a' || x > 'z') { // 检查目录名称是否合法

return; // 如果不合法，直接返回

}

}

if (!son.containsKey(c)) { // 如果目录不存在

Sys temp = new Sys(c); // 创建新的目录对象

temp.par = this; // 设置新目录的父目录

son.put(c, temp); // 将新目录加入当前目录的子目录中

}

}

Sys getCd(String temp) {

if (temp.equals("..")) { // 如果命令是返回上级目录

if (this.par != null) { // 如果有父目录

return this.par; // 返回父目录

}

}

return son.getOrDefault(temp, this); // 返回指定的子目录或当前目录

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in); // 创建Scanner对象以读取输入

Sys rt = new Sys(""); // 初始化根目录

String t = ""; // 用于存储最后一条命令

while (scanner.hasNextLine()) { // 读取输入直到结束

String inp = scanner.nextLine().trim(); // 去除输入的前后空格

if (inp.startsWith("cd")) { // 如果命令是cd

String[] parts = inp.split(" ");

if (parts.length != 2) continue; // 检查命令格式是否合法

if (!inp.contains("/")) { // 检查是否为简单目录名

rt = rt.getCd(parts[1]); // 进入指定目录

}

} else if (inp.startsWith("mkdir")) { // 如果命令是mkdir

String[] parts = inp.split(" ");

if (parts.length != 2) continue; // 检查命令格式是否合法

rt.addChild(parts[1]); // 创建新目录

}

t = inp; // 更新最后一条命令

}

rt.pwd(); // 输出当前路径

scanner.close(); // 关闭Scanner对象

}

}



import sys # 导入sys模块以读取标准输入

# 定义一个类Sys来模拟目录管理系统

class Sys:

def \_\_init\_\_(self, x):

self.par = None # 父目录初始化为None

self.x = x # 当前目录名称

self.son = {} # 子目录字典，存储当前目录下的所有子目录

def pwd(self):

res = [] # 用于存储路径的列表

now = self # 从当前目录开始

while now.par: # 循环直到根目录

res.append(now.x) # 将当前目录名加入结果列表

now = now.par # 向上移动到父目录

res.reverse() # 反转列表以得到正确的路径顺序

if len(res) == 0: # 如果结果列表为空，表示在根目录

print("/") # 输出根目录

else:

print("/" + "/".join(res) + "/") # 输出完整路径

def addChild(self, c):

for x in c:

if ord(x) > ord('z') or ord(x) < ord('a'): # 检查目录名称是否合法

return # 如果不合法，直接返回

if c not in self.son: # 如果目录不存在

temp = Sys(c) # 创建新的目录对象

temp.par = self # 设置新目录的父目录

self.son[c] = temp # 将新目录加入当前目录的子目录中

def getCd(self, temp):

if temp == "..": # 如果命令是返回上级目录

if self.par is not None: # 如果有父目录

return self.par # 返回父目录

if temp not in self.son: # 如果目录不存在

return self # 返回当前目录

return self.son[temp] # 返回指定的子目录

rt = Sys("") # 初始化根目录

t = "" # 用于存储最后一条命令

for inp in sys.stdin: # 读取标准输入

inp = inp.strip() # 去除输入的前后空格

if inp.startswith("cd"): # 如果命令是cd

if len(inp.split()) > 2 or len(inp.split()) == 1: # 检查命令格式是否合法

continue # 不合法则跳过

if inp.find("/") == -1: # 检查是否为简单目录名

rt = rt.getCd(inp.split()[1]) # 进入指定目录

if inp.startswith("mkdir"): # 如果命令是mkdir

if len(inp.split()) > 2 or len(inp.split()) == 1: # 检查命令格式是否合法

continue # 不合法则跳过

rt.addChild(inp.split()[1]) # 创建新目录

t = inp # 更新最后一条命令

rt.pwd() # 输出当前路径



#include <iostream>

#include <string>

#include <unordered\_map>

#include <vector>

#include <algorithm>

#include <numeric>

class Sys {

public:

// 指向父目录的指针

Sys\* par;

// 当前目录名称

std::string x;

// 子目录映射

std::unordered\_map<std::string, Sys\*> son;

// 构造函数，初始化当前目录名称和子目录映射，父目录初始化为 nullptr

Sys(std::string x) {

par = nullptr;

this->x = x;

}

// 输出当前目录的完整路径

void pwd() {

std::vector<std::string> res;

Sys\* now = this;

// 从当前目录向上遍历到根目录，将路径上的目录名称存入 res 中

while (now->par!= nullptr) {

res.push\_back(now->x);

now = now->par;

}

if (res.empty()) {

// 如果 res 为空，说明当前在根目录，输出根目录表示

std::cout << "/";

} else {

// 反转 res，使目录名称顺序正确

std::reverse(res.begin(), res.end());

// 使用 accumulate 函数将路径上的目录名称连接起来输出

std::cout << "/" << std::accumulate(res.begin() + 1, res.end(), res[0], [](std::string a, std::string b) {

return a + "/" + b;

}) << "/";

}

}

// 添加子目录

void addChild(std::string c) {

for (char x : c) {

// 检查目录名称是否合法，只允许小写字母

if (x < 'a' || x > 'z') return;

}

if (son.find(c) == son.end()) {

// 如果子目录不存在，创建新的 Sys 对象作为子目录

Sys\* temp = new Sys(c);

temp->par = this;

// 将新子目录存入子目录映射中

son[c] = temp;

}

}

// 根据输入切换目录或返回当前目录

Sys\* getCd(std::string temp) {

if (temp == "..") {

// 如果输入是".."，返回父目录（如果存在）

if (this->par!= nullptr) return this->par;

}

// 如果输入是有效的子目录名称，返回对应的子目录，否则返回当前目录

if (son.find(temp)!= son.end()) return son[temp];

return this;

}

};

int main() {

std::string inp;

Sys\* rt = new Sys("");

while (std::getline(std::cin, inp)) {

std::string command;

std::string arg;

size\_t spacePos = inp.find(' ');

if (spacePos!= std::string::npos) {

command = inp.substr(0, spacePos);

arg = inp.substr(spacePos + 1);

} else {

command = inp;

}

if (command == "cd") {

if (arg.find('/') == std::string::npos) {

// 根据输入切换目录

rt = rt->getCd(arg);

}

} else if (command == "mkdir") {

// 创建新目录

rt->addChild(arg);

}

}

// 输出当前目录的路径

rt->pwd();

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// 定义目录结构体

typedef struct Sys {

struct Sys\* par; // 父目录

char\* x; // 当前目录名称

struct Sys\* son[100]; // 子目录数组

int son\_count; // 子目录数量

} Sys;

// 创建新的目录

Sys\* createSys(const char\* x) {

Sys\* sys = (Sys\*)malloc(sizeof(Sys));

sys->par = NULL;

sys->x = strdup(x);

sys->son\_count = 0;

return sys;

}

// 输出当前路径

void pwd(Sys\* sys) {

if (sys->par == NULL) { // 如果在根目录

printf("/\n");

return;

}

char\* path[100];

int count = 0;

while (sys->par != NULL) { // 循环直到根目录

path[count++] = sys->x;

sys = sys->par;

}

printf("/");

for (int i = count - 1; i >= 0; i--) {

printf("%s/", path[i]);

}

printf("\n");

}

// 创建子目录

void addChild(Sys\* sys, const char\* c) {

for (int i = 0; c[i] != '\0'; i++) {

if (c[i] < 'a' || c[i] > 'z') { // 检查目录名称是否合法

return; // 如果不合法，直接返回

}

}

for (int i = 0; i < sys->son\_count; i++) {

if (strcmp(sys->son[i]->x, c) == 0) {

return; // 如果目录已存在，直接返回

}

}

Sys\* child = createSys(c); // 创建新的目录对象

child->par = sys; // 设置新目录的父目录

sys->son[sys->son\_count++] =

child; // 将新目录加入当前目录的子目录中

}

// 进入子目录

Sys\* getCd(Sys\* sys, const char\* temp) {

if (strcmp(temp, "..") == 0) { // 如果命令是返回上级目录

if (sys->par != NULL) { // 如果有父目录

return sys->par; // 返回父目录

}

}

for (int i = 0; i < sys->son\_count; i++) {

if (strcmp(sys->son[i]->x, temp) == 0) {

return sys->son[i]; // 返回指定的子目录

}

}

return sys; // 返回当前目录

}

int main() {

Sys\* rt = createSys(""); // 初始化根目录

char inp[256];

while (fgets(inp, sizeof(inp), stdin)) { // 读取输入直到结束

char\* newline = strchr(inp, '\n');

if (newline) \*newline = '\0'; // 去除输入的换行符

if (strncmp(inp, "cd", 2) == 0) { // 如果命令是cd

char\* arg = strtok(inp + 3, " ");

if (arg != NULL && strchr(arg, '/') == NULL) { // 检查是否为简单目录名

rt = getCd(rt, arg); // 进入指定目录

}

} else if (strncmp(inp, "mkdir", 5) == 0) { // 如果命令是mkdir

char\* arg = strtok(inp + 6, " ");

if (arg != NULL) { // 检查命令格式是否合法

addChild(rt, arg); // 创建新目录

}

}

}

pwd(rt); // 输出当前路径

return 0;

}



class Sys {

// 构造函数，初始化父目录、当前目录名称和子目录映射

constructor(x) {

this.par = null;

this.x = x;

this.son = {};

}

// 输出当前目录的完整路径

pwd() {

let res = [];

let now = this;

// 从当前目录向上遍历到根目录，将路径上的目录名称存入 res 中

while (now.par!== null) {

res.push(now.x);

now = now.par;

}

if (res.length === 0) {

// 如果 res 为空，说明当前在根目录，输出根目录表示

console.log('/');

} else {

// 反转 res，使目录名称顺序正确

res.reverse();

// 输出完整路径

console.log(`/${res.join('/')}/`);

}

}

// 添加子目录

addChild(c) {

// 检查目录名称是否合法，只允许小写字母

for (let char of c) {

if (char < 'a' || char > 'z') return;

}

if (!this.son[c]) {

// 如果子目录不存在，创建新的 Sys 对象作为子目录

let temp = new Sys(c);

temp.par = this;

// 将新子目录存入子目录映射中

this.son[c] = temp;

}

}

// 根据输入切换目录或返回当前目录

getCd(temp) {

if (temp === '..') {

// 如果输入是".."，返回父目录（如果存在）

return this.par || this;

}

// 如果输入是有效的子目录名称，返回对应的子目录，否则返回当前目录

return this.son[temp] || this;

}

}

let rt = new Sys('');

let readline = require('readline').createInterface({

input: process.stdin,

output: process.stdout

});

readline.on('line', (inp) => {

let command, arg;

let spacePos = inp.indexOf(' ');

if (spacePos!== -1) {

command = inp.slice(0, spacePos);

arg = inp.slice(spacePos + 1);

} else {

command = inp;

}

if (command === 'cd') {

if (arg.indexOf('/') === -1) {

// 根据输入切换目录

rt = rt.getCd(arg);

}

} else if (command === 'mkdir') {

// 创建新目录

rt.addChild(arg);

}

});

process.on('exit', () => {

// 输出当前目录的路径

rt.pwd();

});



package main

import (

"bufio"

"fmt"

"os"

"strings"

)

// 定义目录结构体

type Sys struct {

par \*Sys // 父目录

x string // 当前目录名称

son map[string]\*Sys // 子目录映射

}

// 创建新的目录

func createSys(x string) \*Sys {

return &Sys{

par: nil,

x: x,

son: make(map[string]\*Sys),

}

}

// 输出当前路径

func (sys \*Sys) pwd() {

var res []string // 用于存储路径的切片

now := sys // 从当前目录开始

for now.par != nil { // 循环直到根目录

res = append(res, now.x) // 将当前目录名加入结果切片

now = now.par // 向上移动到父目录

}

for i, j := 0, len(res)-1; i < j; i, j = i+1, j-1 {

res[i], res[j] = res[j], res[i] // 反转切片以得到正确的路径顺序

}

if len(res) == 0 { // 如果结果切片为空，表示在根目录

fmt.Println("/") // 输出根目录

} else {

fmt.Println("/" + strings.Join(res, "/") + "/") // 输出完整路径

}

}

// 创建子目录

func (sys \*Sys) addChild(c string) {

for \_, x := range c {

if x < 'a' || x > 'z' { // 检查目录名称是否合法

return // 如果不合法，直接返回

}

}

if \_, exists := sys.son[c]; !exists { // 如果目录不存在

child := createSys(c) // 创建新的目录对象

child.par = sys // 设置新目录的父目录

sys.son[c] = child // 将新目录加入当前目录的子目录中

}

}

// 进入子目录

func (sys \*Sys) getCd(temp string) \*Sys {

if temp == ".." { // 如果命令是返回上级目录

if sys.par != nil { // 如果有父目录

return sys.par // 返回父目录

}

}

if child, exists := sys.son[temp]; exists {

return child // 返回指定的子目录

}

return sys // 返回当前目录

}

func main() {

rt := createSys("") // 初始化根目录

scanner := bufio.NewScanner(os.Stdin)

for scanner.Scan() { // 读取输入直到结束

inp := strings.TrimSpace(scanner.Text()) // 去除输入的前后空格

if strings.HasPrefix(inp, "cd") { // 如果命令是cd

parts := strings.Split(inp, " ")

if len(parts) != 2 {

continue // 检查命令格式是否合法

}

if !strings.Contains(parts[1], "/") { // 检查是否为简单目录名

rt = rt.getCd(parts[1]) // 进入指定目录

}

} else if strings.HasPrefix(inp, "mkdir") { // 如果命令是mkdir

parts := strings.Split(inp, " ")

if len(parts) != 2 {

continue // 检查命令格式是否合法

}

rt.addChild(parts[1]) // 创建新目录

}

}

rt.pwd() // 输出当前路径

}