It's MyFS!!!!!

Background

You, Togawa Sakiko, an operations engineer at Bushiroad Inc., are busy preparing files and materials for the upcoming *MyGO!!!!!×Ave Mujica* joint live show. At work, your colleagues see you as a hard-working workaholic. What they don't know is your secret: you're also Oblivionis, the keyboardist for *Ave Mujica*.

Today, while reviewing the setlist for your rival band, *MyGO!!!!!*, something caught you off guard — *Haruhikage*, a song that belongs exclusively to *CRYCHIC*, your former band, was listed. A chill ran down your spine. "The weak me of the past is long gone," you thought. "I can't let the same tragedy repeat itself."

Determined, you accessed the song storage system, searching for *Haruhikage*. You had to erase it — to wipe away the final, lingering trace of that spring.

Description

> This question is designed to examine your proficiency of C++ class / C structure with memory management, and also a test to check whether you rely too much on GUI.

Linux/Unix systems have robust file systems sharing a similar set of utilities, like `cat`, `cd`, `find`, `ls`, `mkdir`, `rm` etc. As a operations engineer, Sakiko is very familiar with these tools, however a mousepad she bought when she was a newbie provides some basic introductions for these tools:

Sakiko's Mousepad

* Just for reference, you can also use `--help` option to check the help of these utils on your UNIX system. Note that only commands listed here are required to be implemented.

- `cat`

    - Concatenate FILE(s) to standard output.
    - Usage: `cat /path/to/file`
- `cd`
    - Change the shell working directory.
    - Usage: `cd /path/to/directory`
- `echo`
    - Repeat the following arguments, used as input source of files in this question.
    - Usage:
        - `echo <textcontent> > /path/to/file` replace the contents of given file to the arguments, with a line break.
        - `echo <textcontent> >> /path/to/file` append the arguments to the tail of given file, with a line break.
- `find`
    - Search for files in a directory hierarchy.
    - Usage: `find /path -name <pattern>`
- `ls`
    - List information about the FILEs (the current directory by default).
    - Usage: `ls -a /path`
- `mkdir`
    - Create the directory(ies), if they do not already exist.
    - Usage: `mkdir -p <dirname>`
- `mv`
    - Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.
    - Usage:
        - `mv /path/to/src /path/to/dst_file` move the source directory/file to be the destination file/directory
        - `mv /path/to/src /path/to/dst_dir/` move the source directory/file under the destination directory
- `rm`

    - Remove (unlink) the FILE(s).
    - Usage `rm -r /path/to/dir` or `rm /path/to/file`
- `pwd`
    - Print the name of the current working directory.
    - Usage: `pwd`

A bad news is, the file storage is nearly raw, meaning that you will need to implement a file system yourself, to support these functions of creating, deleting, modifying and searching, so that you can find *Haruhikage* and delete it.

Input

Since the materials are stored in a distributed cluster, meaning that *Haruhikage* could be stored in multiple replications, so you will need to look through 20 different file systems(pass 20 testcases).

For each testcase, Sakiko will first tell you the number of instructions T to be executed in a single line.

For the next T lines, each line is a command to be executed in sequential order, the possible types of commands are:

- `cat`
    - Concatenate FILE(s) to standard output.
    - Number of args: 1
        - The 1st argument states a path in the file system.
- `cd`
    - Change the shell working directory.
    - Number of args: 1
        - The 1st argument states a path in the file system.
- `echo`
    - Repeat the following arguments, used as input source of files in this question.
    - Number of args: 1 or 3
        - The 1st argument stands for the content to be echoed, quoted by `''` or not quoted.
        - If applicable, the 2nd argument could be `>` or `>>`, which is the **redirection operator**.
        - The 3rd argument states a path in the file system.

    - If the file given does not exist in that path, but its parent directory exists, create a new file there instead; if its parent directory also does not exist, throw an error.
- `find`
    - Search for files in a directory hierarchy.
    - Number of args: 3
        - The 1st argument states a path in the file system.
        - The 2nd argument is `-name`.
        - The 3rd argument is a regular expression quoted by `''`.
- `ls`
    - List information about the FILEs (the current directory by default).
    - Number of args: 2
        - The 1st argument is `-a`.
        - The 2nd argument states a path in the file system.
- `mkdir`
    - Create the directory(ies) and necessary parent directory(ies), if they do not already exist.
    - Number of args: 2
        - The 1st argument is `-p`
        - The 2nd argument states a path in the file system.
- `mv`
    - Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.
    - Number of args: 2
        - The 1st argument is the path of the source file/directory
        - The 2nd argument is the path of the destination, when it ends with `/`, meaning moving source under this destination directory.
    - We guarantee that the testcases will not move the working directory or its parent directories.
- `rm`
    - Remove (unlink) the FILE(s).
    - Number of args: 1 or 2

        - If 1 argument received, it states a path to a file in the file system.
        - If 2 arguments received, the 1st one is `-r`, the 2nd one states a path to a directory in the file system.
    - We guarantee that the testcases will not delete the working directory or its parent directories.
- `pwd`
    - print the absolute path of the current working directory.
    - Number of args: 0

* Note that "path" mentioned above could be either relative or absolute path, meaning that you need to maintain the path of the current working directory. Initially, the current working directory is the root directory `/`.

Output

In a single testcase, for each input instruction, generate necessary output to stdout. Below are the descriptions for output formats:

- `cat`
    - Output the content of a file, do not output any extra line breaks (because probably the file content contains line breaks since written by `echo`).
- `cd`
    - Output nothing unless any error occurs.
- `echo`
    - Output the argument with a line break, if no redirection operator specified.
    - Output nothing if any redirection operator specified, write into the file instead.
- `find`
    - If results found, output their **relative paths** to the given path according to the given pattern, each in a single line.
    - If no result found, do not output.
- `ls -a`
    - Different from standard UNIX output, output all the terms within the given path in a single line, separated by `\t`.
    - Note: "all the terms" include links like `.` and `..`, check it out by trying `find` on your own UNIX system.

- `mkdir`
  - Output nothing unless any error occurs.
- `mv`
  - Output nothing unless any error occurs.
- `rm`
  - Output nothing unless any error occurs.
- `pwd`
  - Output the absolute path of current working directory
- For all types of errors, just output `error` in a single line:
  - wrong number of arguments, like missing or too-many arguments.
  - invalid arguments, like `find whatever -wrong argument`.
  - accessing files/directories that does not exist.
  - wrong path, like `cd` into a **file path** or `echo` into a **directory path** etc.
  - name conflict, like making/renaming a file to the name of an existing file.
  - other possible errors.

Sample Input #1

```
20
pwd
mkdir -p /BanGDream/MyGO/Live20250426/songs
mkdir -p /BanGDream/AveMujica/Live20250426/songs
cd /BanGDream/MyGO/Live20250426/songs
pwd
echo 'Kajikanda kokoro furueru manazashi' > Haruhikage
echo 'Sekai de boku wa hitoribocchi datta' >> ./Haruhikage
echo 'Chiru koto shika shiranai haru wa' >> ../../Live20250426/songs/Haruhikage
echo 'Maitoshi tsumetaku ashirau' >> /BanGDream/MyGO/Live20250426/songs/Haruhikage
find /BanGDream -name 'Haru.*'
mkdir -p /BanGDream/CRYCHIC/songs
cd /BanGDream/CRYCHIC/songs
mv /BanGDream/MyGO/Live20250426/songs/Haruhikage ./
ls -a .
cat Haruhikage
cd /BanGDream
rm -r ./CRYCHIC
find /BanGDream -name 'Haru.*'
echo 'Haruhikage deleted'
cd CRYCHIC
```

Sample Output #1

```
/
/BanGDream/MyGO/Live20250426/songs
/BanGDream/MyGO/Live20250426/songs/Haruhikage
.   ..  Haruhikage
Kajikanda kokoro furueru manazashi
Sekai de boku wa hitoribocchi datta
Chiru koto shika shiranai haru wa
Maitoshi tsumetaku ashirau
Haruhikage deleted
error
```

Testcases

There are 20 testcases in total, each takes up 5 points. You are encouraged to implement part of the functions and submit your semi-finished code to test if the implemented part works correctly. For testcase 1-18, we guarantee that no errors happen. You don't need to worry about the efficiency too much, simple depth-first-search is all you need for `find`.

Testcase No.	Commands used	$T \leq$	Description
1	echo, cat	1e4	
2	mkdir, cd, pwd	1e4	Absolute paths only
3	mkdir, cd, pwd	1e4	Relative/absolute paths
4	cat, cd, echo, ls, mkdir, rm	1e3	
5	cat, cd, echo, ls, mkdir, rm	1e4	
6	cat, cd, echo, ls, mkdir, mv	1e3	
7	cat, cd, echo, ls, mkdir, mv	1e4	
8	cat, cd, echo, ls, mkdir, find	1e3	
9	cat, cd, echo, ls, mkdir, find	1e4	
10	cat, cd, echo, ls, mkdir, rm, mv	1e5	
11	cat, cd, echo, ls, mkdir, rm, find	1e5	
12	cat, cd, echo, ls, mkdir, mv, find	1e4	
13	Mixture of all above - 1	1e3	
14	Mixture of all above - 2	1e3	
15	Mixture of all above - 3	1e4	
16	Mixture of all above - 4	1e4	
17	mkdir, echo, find, rm	2e4	Efficiency test of `find`
18	cd, mkdir, echo, rm	1e4	Memory leak test of `rm`
19	Mixture of all above - 3(with err)	1e4	Arguments can be wrong
20	Mixture of all above - 4(with err)	1e4	Paths, names and dependencies can be wrong

Note that some of the testcases contain only part of the commands, so don't hesitate to submit and see if your implementation of that part is correct (even if you have not implemented all types of commands)!

Hints

- Note that there are many details about the path format, for example, the path of a directory **may** ends with `/`, but some times this slash could be omitted.
- Descriptions and documents for most of the commands above can be found at **GNU Coreutils**.
- For regular expressions of `find` commands, try **regular expression library** since C++11.
- Try the commands out on your UNIX PC is a good way to understand how these commands interact with the file system.

- Be careful when handling with memory allocation. Not properly Allocating, accessing and releasing resources will lead to various runtime errors and memory-limit-exceeded errors.

Implementing this using C++ with derived classes and smart pointers is easier than using C. However, you are still encouraged to give it a try. I believe that no matter how much you know about UNIX, memory and pointers before, you can be a master of them after solving this problem ;)