

Esercizio 1

Scrivere un programma C che implementi un `find` semplificato (nel seguito, quando si parla di “file” si intende un file o una directory). In particolare, il programma potrà essere lanciato con i seguenti argomenti:

1. lista di file (almeno uno), separati da spazi (assumere che nessuno di questi file possa cominciare con un dash -);
2. argomento opzionale `-maxdepth N`, con lo stesso significato che ha nel `find` di sistema (ovvero: un file viene considerato solo se il suo attuale path, relativo alla directory data come argomento, contiene al più N directory);
3. lista di test, separati da spazi;
4. lista di azioni, separate da spazi.

Dei test, vanno implementati solo i seguenti:

- `-type t`, con lo stesso significato che ha nel `find` di sistema, ma con t che può essere solo uno tra `l`, `d` o `f`;
- `-size N`, con lo stesso significato che ha nel `find` di sistema, ma senza suffissi ed assumendo che N indichi sempre i bytes (quindi, come il suffisso `c` di sistema);
- `-perma i`, con lo stesso significato che ha nel `find` di sistema, ma i è sempre fatto di sole 4 cifre ottali (quindi, niente permessi definiti in modo simbolico o caratteri speciali come `-`).

Quando più test sono presenti, assumere di dover effettuare l’OR degli stessi. Se sono presenti più istanze dello stesso test, considerare solo l’ultima istanza. Se non c’è nessun test, allora tutti i file passano la fase di test. Non è possibile effettuare altre combinazioni logiche dei test.

Delle azioni, vanno implementate solo le seguenti:

- `-ls`, con un significato simile a quello che ha nel `find` di sistema: se il file matchato ha path `file`, dovrà restituire una selezione dell’output del comando `ls -ld --color=never file`: vanno stampati solo i permessi, l’hard link count, la dimensione ed il nome; come separatore tra questi campi, usare sempre il tab. Se il file è un link simbolico, va mostrata la destinazione, come con l’opzione `-l` di `ls`;
- `-print`, con lo stesso significato che ha nel `find` di sistema.

Se non c’è nessuna azione, effettuare l’azione `-print`. Se c’è più di una azione, effettuare l’ultima azione data.

Si possono manifestare solamente i seguenti errori:

- Non esiste una delle directory D date nella lista iniziale dei file. Il programma dovrà allora proseguire con la prossima directory, scrivendo su standard error **Unable to open D because of e** , dove e è la stringa di sistema che spiega l'errore. L'exit status del programma dovrà essere il numero di directory saltate per via di questo motivo.
- Non viene dato nessun file nella lista dei file. Il programma dovrà allora terminare con exit status 20 (senza eseguire alcuna azione), e scrivendo su standard error **Usage: p dirs [-maxdepth N] [tests] [actions]**, dove p è il nome del programma stesso.
- Fallisce una qualsiasi altra system call. Il programma dovrà allora terminare con exit status 100 (senza eseguire alcun'altra azione), e scrivendo su standard error **System call s failed because of e** , dove e è la stringa di sistema che spiega l'errore ed s è la system call che ha fallito.

Attenzione: non è permesso usare le system call **system**, **exec**, **sleep** e **popen**. Il programma non deve scrivere nulla sullo standard error, a meno che non si tratti di un errore nelle opzioni da riga di comando come descritto sopra. Per ogni test definito nella valutazione, il programma dovrà ritornare la soluzione dopo al più 10 minuti.

Esempi

Da dentro la directory `grader.2`, dare il comando `tar xfzp all.tgz input_output.1 && cd input_output.1`. Ci sono 6 esempi di come il programma `./1/1` possa essere lanciato, salvati in file con nomi `inp_out.i.sh` (con $i \in \{1, \dots, 6\}$). Per ciascuno di questi script, la directory di input è `inp.i`. La directory con l'output atteso è `check/out.i`. La directory `check/out_tmp.i` contiene dei log di esempio di `valgrind`. Quelli prodotti dalla soluzione proposta dovranno essere simili a questi, ovvero non contenere messaggi d'errore (questo controllo viene fatto in `inp_out.i.sh`).