Test practic la SO – varianta nr. 4

Realizați o aplicație formată din următoarele patru componente cooperante, definite în continuare, care vor fi dispuse în sistemul de fișiere conform ierarhiei de mai jos:

```
.
├── main.sh
└── app
    └── control_byte.sh
    └── recurse.sh
    └── source
        └── first_and_last_byte.c
```

1. Să se scrie un program C, numit "first_and_last_byte.c", conform specificației următoare:
Programul va primi cel puțin un argument în linia de comandă; în caz contrar își va încheia execuția cu un cod de terminare unic, nenul. Programul va considera acel argument ca fiind un nume de fișier și va verifica dacă are access de citire asupra acestuia, folosind doar un apel de funcție din API-ul POSIX, iar în caz negativ își va încheia execuția cu un cod de terminare unic, nenul.
Altfel, programul va determina valoarea (fără semn) atât a primului octet, cât și a ultimului octet din fișierul dat, folosind strict apeluri către funcții din API-ul POSIX. Fiecare apel către funcții din API-ul POSIX va trata cazul de eroare prin încheierea execuției programului cu un cod de terminare unic, nenul. În cazul în care fișierul este gol (i.e., are dimensiunea zero), valorile primului și ultimului octet se consideră a fi zero.
Apoi, se va afișa pe ieșirea stdout, folosind o funcție din biblioteca stdio, un mesaj format din: numele fișierului primit ca argument, separatorul '=' , numărul întreg fără semn reprezentând suma celor doi octeți citiți din fișier, și terminat cu caracterul '\n'. Apoi programul își va încheia execuția cu codul de terminare 0.

2. Să se scrie un script bash, numit "main.sh", conform specificației următoare:
Scriptul va primi un argument în linia de comandă și va face următoarele verificări, în ordinea următoare:
i) va verifica dacă în directorul în care se află "main.sh", există un subdirector numit "app" ce conține un script numit "control_byte.sh" și că are permisiunea de execuție a acestuia, iar în caz negativ va afișa pe ieșirea stderr un mesaj de eroare adecvat și se va termina cu codul 1;
ii) va verifica dacă în subdirectorul numit "app", există un subdirector numit "source" ce conține un program numit "first_and_last_byte.c" și că are permisiunea de citire a acestuia, iar în caz negativ va afișa pe ieșirea stderr un mesaj de eroare adecvat și se va termina cu codul 2;
iii) va verifica dacă argumentul primit reprezintă un fișier existent de tip fie director, fie normal ("regular"), iar în caz negativ va afișa pe ieșirea stderr un mesaj de eroare adecvat și se va termina cu codul 3.
Apoi scriptul "main.sh" va invoca scriptul "control_byte.sh", transmițându-i în linia de comandă, argumentul pe care l-a primit el în linia de comandă, și la final va afișa codul de terminare a execuției acestuia.

3. Să se scrie un al doilea script bash, numit "control_byte.sh", conform specificației următoare:
Scriptul va primi un argument în linia de comandă; în caz contrar se va termina cu codul de exit 2.
Mai întâi scriptul va verifica dacă în directorul în care se află "control_byte.sh" există și un script numit "recurse.sh" și că are permisiunea de execuție a acestuia, iar în caz negativ va afișa pe ieșirea stderr un mesaj de eroare adecvat și se va termina cu codul 1.

Apoi scriptul "control_byte.sh" va invoca scriptul "recurse.sh", transmițându-i în linia de comandă, argumentul pe care l-a primit el în linia de comandă. Invocarea scriptului "recurse.sh" se va face printr-o comandă compusă de tip pipeline, care să proceseze outputul produs prin execuția scriptului "recurse.sh" în maniera următoare: se va înlocui fiecare cifră zecimală cu "următoarea" cifră (i.e., 0 va fi înlocuit cu 1, 1 va fi înlocuit cu 2, …, 9 va fi înlocuit cu 0), iar apoi, în outputul produs în urma înlocuirii, se vor sorta liniile numeric crescător după al doilea câmp delimitat de caracterul "=".
La finalul procesării pipeline-ului, scriptul "control_byte.sh" se va termina cu codul de exit a pipeline-ului.

4. Să se scrie un al treilea script bash, numit "recurse.sh", conform specificației următoare:
Scriptul va primi un argument în linia de comandă; în caz contrar se va termina cu codul de exit 2.
Mai întâi scriptul va verifica dacă în subdirectorul "source" se mai află și un fișier numit "first_and_last_byte"

(executabilul obținut prin compilare din programul "first_and_last_byte.c"), iar în caz negativ va apela compilatorul gcc pentru a-l produce și, doar dacă vor fi erori la compilare, va afișa pe ieșirea stderr un mesaj de eroare adecvat și se va termina cu codul de exit 3.
Apoi, scriptul "recurse.sh" va apela o funcție recursivă cu acel argument pe care l-a primit el în linia de comandă. Dacă argumentul funcției este un fișier de tip normal ("regular"), atunci se va apela executabilul "first_and_last_byte", transmițându-i acestuia ca argument în linia de comandă, calea fișierului primit ca argument. În schimb, dacă argumentul funcției este un fișier de tip director, atunci funcția va itera, folosind o structură repetitivă, prin toate intrările directe din acel director și se va apela recursiv cu fiecare intrare găsită, drept argument. (Atenție: așadar, aici trebuie să implementați o recursie explicită!)

Barem de corectare

Observații:
- programul C și cele trei scripturi trebuie plasate într-o ierarhie de directoare conform celor descrise în enunțul problemei (și apelate în mod corespunzător poziției lor în ierarhia respectivă).
- conform specificației date, programul C poate folosi biblioteca standard de C doar pentru afișarea rezultatelor; interacțiunea cu fișierul (aflarea tipului, lungimii, etc.) se va face folosind doar API-ul POSIX.
- pentru implementarea parcurgerii recursive se va respecta specificația dată pentru scriptul "recurse.sh".
- **dacă nu respectați toate condițiile precedente** (de exemplu, dacă plasați vreunul dintre cele trei fișiere apelate pornind de la “main.sh” într-un alt director, sau dacă folosiți comenzi precum find sau ls -R pentru parcurgerea recursivă în cadrul "recurse.sh"), atunci vom evalua corectitudinea rezolvării, dar **punctajul total acordat va fi înjumătățit**.
- fiecare punctaj din barem se acordă integral, sau deloc.

1. Baremul pentru programul " first_and_last_byte.c":
a) 0.5p – implementarea corectă a verificării existenței argumentui primit.
b) 0.5p – implementarea corectă, conform specificației date, a verificării dreptului de access de citire asupra fișierului.
c) 0.5p – deschiderea fișierului în mod correct.
d) 0.5p – citirea corectă a primului octet, conform specificației date.
e) 1p – citirea corectă a ultimului octet, conform specificației date.
f) 0.5p – afișarea rezultatului pe ieșirea stdout, conform specificației date.
g) 1p – tratarea tuturor erorilor posibile, conform specificației date.
h) 1p – programul se compilează fără erori și fără warning-uri.
Total: 5.5p

2. Baremul pentru scriptul "main.sh":
a) 3 x 0.5p – implementarea corectă, conform specificației date, a verificărilor descrise la punctul i) din specificație.
b) 3 x 0.5p – implementarea corectă, conform specificației date, a verificărilor descrise la punctul ii) din specificație.
c) 2 x 0.5p – implementarea corectă, conform specificației date, a verificărilor descrise la punctul iii) din specificație.
d) 0.5p + 0.5p – invocarea corectă a scriptului "control_byte.sh" și afișarea codului de exit, conform specificației date.
Total: 5p

3. Baremul pentru scriptul "control_byte.sh":
a) 0.5p – implementarea corectă, conform specificației date, a verificării referitoare la argumentul primit.
b) 1p – implementarea corectă, conform specificației date, a verificării referitoare la scriptul "recurse.sh".
c) 0.5p – invocarea corectă a scriptului "recurse.sh", în cadrul pipeline-ului, conform specificației date.
d) 1p – apelul comenzii potrivite, în cadrul pipeline-ului, pentru înlocuirea cifrelor, conform specificației date.
e) 1p – apelul comenzii potrivite, în cadrul pipeline-ului, pentru a sorta liniile, conform specificației date.
f) 0.5p – terminarea scriptului cu codul de exit a pipeline-ului, conform specificației date.
Total: 4.5p

4. Baremul pentru scriptul "recurse.sh":
a) 0.5p – implementarea corectă, conform specificației date, a verificării referitoare la argumentul primit.
b) 0.5p + 1p – implementarea corectă, conform specificației date, a verificării referitoare la programul executabil "first_and_last_byte" și apelul compilatorului, dacă este cazul, conform specificației date.
c) 0.5p – implementarea corectă a funcției recursive de parcurgere a unui director, plus:
c_1) 0.5p – apelarea corectă a programului executabil "first_and_last_byte", conform specificației date.

c_2) 1.5p – luarea deciziei de procesare, dacă argumentul primit este un director, prin parcurgerea iterativă a intrărilor directe din acel director și apelarea recursivă a funcției pentru fiecare intrare din acesta, conform specificației date.
d) 0.5p – apelarea funcției de parcurgere pentru argumentul primit.
Total: 5p