Laborator #6 : exerciții de laborator

Sumar:

I) Exerciții de programare C cu fișiere (diverse prelucrări de fișiere și implementarea unor comenzi uzuale):

a) *Exerciții propuse spre rezolvare:*

Intrați pe setul de exerciții propuse spre rezolvare, pe care vi-l va indica profesorul de laborator, în timpul laboratorului, și încercați să le rezolvați singuri:

Setul #1 | Setul #2

Setul 1

1. [#1: The filter NoVocals]
   Să se scrie un program C care primește de la linia de comandă numele a două fișiere, cu care va face următoarea procesare: va copia conținutul fișierului de intrare în cel de ieșire, eliminând vocalele întâlnite (majuscule și minuscule). În caz că fișierul de ieșire deja există, se va cere confirmare de suprascriere. Respectiv, va fi creat în cazul în care nu există, cu drepturi de citire și scriere doar pentru proprietar.
   Cerință: se vor utiliza apelurile de sistem din API-ul POSIX pentru accesarea fișierelor.
   (Indicație: printr-o singură parcurgere a fișierului de intrare, copiați fiecare caracter citit, aplicând transformarea cerută, în fișierul de ieșire.)
   Cerință suplimentară: dacă de la linia de comandă se primește un singur nume de fișier, sau dacă numele fișierului de ieșire coincide cu numele celui de intrare, atunci se va trata, într-un mod adecvat, această situație (i.e., nu se mai face copiere, ci supra-scriere).

2. [#2: The command MyCut]
   Să se scrie un program C ce implementează comanda `cut`, inclusiv cu opțiunile `-b`, `-f` și `-d`. Se va permite precizarea de argumente multiple de tip nume de fișiere, în linia de comandă a programului, pentru procesare.
   Cerință: se vor utiliza apelurile de sistem din API-ul POSIX pentru accesarea fișierelor.
   (Indicație: încercați să simulați cât mai exact comportamentul comenzii cut.)

1. [#1: The filter ROT13]
   Să se scrie un program C care primește de la linia de comandă numele a două fișiere, cu care va face următoarea procesare: va copia conținutul fișierului de intrare în cel de ieșire, aplicând o transformare de tip ROT13 pe fiecare literă (majusculă și minusculă). În caz că fișierul de ieșire deja există, se va cere confirmare de suprascriere. Respectiv, va fi creat în cazul în care nu există, cu drepturi de citire și scriere doar pentru proprietar.
   Cerință: se vor utiliza apelurile de sistem din API-ul POSIX pentru accesarea fișierelor.
   Despre ce este o transformare de tip ROT13 puteți citi aici.
   (Indicație: printr-o singură parcurgere a fișierului de intrare, copiați fiecare caracter citit, aplicând transformarea cerută, în fișierul de ieșire.)
   Cerință suplimentară: dacă de la linia de comandă se primește un singur nume de fișier, sau dacă numele fișierului de ieșire coincide cu numele celui de intrare, atunci se va trata, într-un mod adecvat, această situație (i.e., nu se mai face copiere, ci supra-scriere).

2. [#2: The command MyGrep]
   Să se scrie un program C ce implementează comanda `grep`, inclusiv cu opțiunile `-c`, `-v` și `-n`. Ca și *pattern* de căutat, veți considera doar cazul simplu al cuvintelor fixe/constante (deci fără expresii regulate, de niciun fel). Se va permite precizarea de argumente multiple de tip nume de fișiere, în linia de comandă a programului, pentru procesare.
   Cerință: se vor utiliza apelurile de sistem din API-ul POSIX pentru accesarea fișierelor.
   (Indicație: încercați să simulați cât mai exact comportamentul comenzii grep.)

b) *Exerciții suplimentare, propuse spre rezolvare pentru acasă:*

Iată alte câteva exerciții de programare C cu prelucrări diverse de fișiere, pe care să încercați să le rezolvați singuri în timpul liber, pentru a vă auto-evalua cunoștințele dobândite în urma acestui laborator:

1. [MyFind #2]
   Să se implementeze o clonă simplificată pentru comanda find, care funcționează astfel: programul primește un singur parametru în linia de comandă, reprezentând numele exact al fișierului care este căutat. Pornind din directorul *acasă* al utilizatorului curent, programul va parcurge recursiv întregul subarbore din sistemul de fișiere cu rădăcina în acesta, și va afișa calea, relativă față de directorul de start, spre fiecare fișier întâlnit în timpul parcurgerii, ce are numele căutat. În plus, în caz de eroare, programul va afișa un mesaj explicit și se va închide "elegant", returnând și un cod de terminare specific erorii apărute.
   (Indicație: parcurgeți directorul cu șablonul indicat în lecția practică și, pentru fiecare intrare din director ce are numele căutat, afișați calea ei, iar pentru fiecare intrare ce este de tip director, apelați recursiv funcția de parcurgere.)

   Show / Hide some suggestions for solving the problem

   Ideea de rezolvare: parcurgerea recursivă se implementează similar ca la exercițiul rezolvat [MyFind #1].
   Iar informațiile ce trebuie afișate sunt mult mai simple în cazul acestui exercițiu!

2. [MyFind #3]
   Să se scrie un program C care șterge toate legăturile simbolice "rupte" (i.e. legături ale căror destinații nu mai există) aflate într-un director, dat ca parametru, sau în subdirectoarele lui (parcurse recursiv).
   (Indicație: parcurgeți directorul cu șablonul indicat în lecția practică și, pentru fiecare intrare din director ce este o legătură simbolică, testați existența destinației cu funcția `access(legătură, F_OK)` și, în caz negativ, ștergeți legătura respectivă cu funcția `unlink(legătură)`, iar pentru fiecare intrare ce este de tip director, apelați recursiv funcția de parcurgere.)

   Show / Hide some suggestions for solving the problem

   Ideea de rezolvare: parcurgerea recursivă se implementează similar ca la exercițiul rezolvat [MyFind #1].
   Iar prelucrările cerute sunt mai simple în cazul acestui exercițiu; pentru implementarea lor folosiți apelurile de sistem `access()` și `unlink()` .

3. [MyMv]
   Să se scrie un program C ce implementează comanda `mv` , inclusiv cu opțiunile sale `-i` , `-u` și `-t` . Se va permite precizarea mai multor fișiere sursă și a unui singur fișier/director destinație.

4. [MyLs]
   Să se scrie un program C ce implementează comanda `ls` , inclusiv cu opțiunile sale `-l` și `-A` . Se va permite precizarea de argumente multiple de tip nume de fișiere sau directoare.

5. [MyRm]
   Să se scrie un program C ce implementează comanda `rm` , inclusiv cu opțiunile sale `-i` și `-r` . Se va permite precizarea de argumente multiple de tip nume de fișiere sau directoare.

6. [MyChmod]
   Să se scrie un program C ce implementează o variantă interactivă a comenzii `chmod` , i.e. programul va afișa meniuri text pentru interacțiunea cu utilizatorul, în vederea interogării acestuia pentru schimbarea permisiunilor fișierelor.
