

File System

Descriere

Răzvan e pasionat de sistemele de operare și dorește să își exerseze skill-urile de programare cu un exercițiu mai complex: implementarea minimalistă a unui sistem virtual de fișiere, prin care dorește să aibă câteva operații minimaliste la început, pe care le va realiza în linie de comandă:

1. `mkdir <nume>` - creează un director cu numele `nume`. Dacă directorul există, nu se întâmplă nimic.
2. `rmdir <nume>` - șterge directorul cu numele `nume`. Dacă directorul **NU** există, nu se întâmplă nimic
3. `touch <nume>` - creează un fișier cu numele `nume`. Dacă fișierul există, nu se întâmplă nimic.
4. `rm <nume>` - șterge fișierul cu numele `nume`. Dacă fișierul **NU** există, nu se întâmplă nimic.
5. `pwd` - afișează calea curentă.
6. `dir` - afișează toate numele de fișiere și directoare din directorul curent, sortate alfabetic crescător.
7. `cd <nume>` (change directory) - mută utilizatorul în directorul `nume`.
 - Dacă directorul `nume` nu există nu se întâmplă nimic.
 - Dacă directorul `nume` este `..`, ne vom muta un director în spate.
 - Dacă suntem în directorul rădăcină și încercăm să ne mutăm un director în spate, nu se întâmplă nimic.

Răzvan a făcut o implementare dar încă are ceva bug-uri la aceasta, așa că apelează și el la voi pentru a-l ajuta la vânătoarea de buguri.

Funcționalități noi

Răzvan s-a gândit că ar mai vrea să facă și un progres la acest proiect, așa că vă roagă să aduceți următoarele funcționalități:

1. să nu se poată crea în directorul curent un folder și un fișier cu același nume.
2. comanda `cd ../..` să poată merge două directoare în spate dacă este posibil acest lucru. (De exemplu dacă calea curentă este `/task1/sample/src`, după aplicarea comenzii calea curentă va fi `/task1`)
 - Dacă este posibil să se meargă maxim un director în spate, se va realiza acest lucru. (Adică dacă calea curentă este de tipul `/task`)
 - Dacă nu este posibil să se meargă nici măcar un director în spate, nu se va întâmpla nimic (Adică dacă calea curentă este de tipul `/`)

Răzvan vă mulțumește!

Observații

1. Se garantează că toate comenzile sunt valide.
2. Se garantează că parametrul primit de comanda `cd` este ori un șir format din litere mici sau mari ale alfabetului englez sau cifre, ori `..` ori `../..`

3. Pentru afișarea path-ului curent, se va adăuga un / între două nume de directoare din ierarhie.

Exemplu:

```
5
mkdir test1
cd test1
mkdir sol
cd sol
pwd
```

va afișa path-ul curent ca fiind /test1/sol și nu /test1sol.

Input Format

Pe prima linie, un număr **N** ce reprezintă numărul de comenzi ce vor fi date. Pe următoarele **N** linii se află o comandă având formatul specificat mai sus. Se garantează că toate comenzile sunt valide.

Output Format

Se vor afișa rezultatul pentru comenzile de tipul pwd și dir.

Example

Exemplu 1

Input

```
3
touch file1.txt
mkdir file1.txt
dir
```

Output

```
file1.txt
```

Explicație

1. Se va afișa doar o singură dată file1.txt, corespunzător fișierului (comanda mkdir file1.txt nu va avea niciun efect pentru că există deja un fișier cu același nume)

Exemplu 2

Input

```
6
mkdir test1
cd test1
mkdir test2
cd test2
cd ../../
pwd
```

Output

```
/
```

Explicație

1. Se creează directorul test1 și ne mutăm în el. Calea curentă va fi /test1.
2. Se creează directorul test2 și ne mutăm în el. Calea curentă va fi /test1/test2.
3. Ne mutăm două directoare în spate, deci calea curentă va fi /.

Exemplu 3

Input

```
19
pwd
mkdir task1
mkdir task2
mkdir task3
pwd
cd task1
pwd
touch test1.in
touch test2.in
touch test4.in
rm test4.in
touch test3.in
mkdir solution
pwd
dir
cd solution
pwd
cd ../..
pwd
```

Output

```
/
/
/task1
/task1
solution test1.in test2.in test3.in
/task1/solution
/
```

Explicație

1. Se începe din directorul / și se afișează path-ul curent.
2. Se creează directoarele task1, task2 și task3.
3. Se afișează path-ul curent, care rămâne același.
4. În urma comenzii cd task1, ne mutăm în directorul task1 și path-ul curent va fi /task1.
5. Se creează fișierele test1.in, test2.in și test4.in. Se șterge task4.in și se creează task3.in
6. Se creează directorul solution și se afișează toate fișierele și directoarele din folderul curent.
7. În urma comenzii cd solution, ne mutăm în directorul task1 și path-ul curent va fi /task1/solution.
8. În urma comenzii cd ../.., ne mutăm în directorul task1 și path-ul curent va fi /.

Download sursă [aici](#).

Lazy Evaluation

Descriere

Mihai este un student la Facultatea de Automatică și Calculatoare. Întrucât este foarte pasionat de Programarea Calculatoarelor și Limbaje de Programare, el decide să înceapă un proiect personal, în care să studieze limbajele de programare, cu accent pe sintaxa acestora.

Pentru început, el realizează că are nevoie să dezvolte o structură de date specială în nodurile căreia să țină minte șiruri de caractere. Aceste șiruri de caractere au unele proprietăți speciale:

- sunt formate numai din caracterele 0 și 1.
- șirurile de caractere trebuie să **nu** conțină substringul **01**.

Mai mult, el își dă seama de faptul că nu trebuie să evalueze toate șirurile existente cu aceste proprietăți, pentru că ele sunt în număr infinit. Astfel, el decide să scrie un program care să primească din când în când un număr de șiruri pe care el dorește să le obțină (șirul de numere terminându-se cu un număr negativ) și printează rezultatele. Pentru o implementare eficientă, el aduce următoarea optimizare: atunci când trebuie să evalueze un număr de șiruri, el nu le redescoperă de fiecare dată, ci le folosește pe cele de la evaluările precedente.

Într-o manieră mai formală, șirurile descoperite de Mihai se pot determina folosind următorul model (numit în literatura de specialitate *gramatică independentă de context* - concret, aceasta reprezintă o mulțime de șabloane/patterns care permite generarea unor șiruri de caractere bazându-se pe anumite reguli):

```
A -> 1A | B
B -> B0 | ""
```

unde A și B sunt două simboluri externe (numite simboluri *neternale*), folosite pentru a putea fi înlocuite în continuare în mod recursiv, folosind setul de reguli de mai sus.

Spre exemplu, cea de-a doua regulă din gramatica de mai sus semnifică faptul că un simbol neterminal *B* (un simbol care poate fi înlocuit în continuare) poate fi înlocuit fie cu șirul vid, fie cu "B0", adică *B* poate genera, prin aplicări diferite ale regulilor de înlocuire, toate șirurile care conțin doar 0, cu oricâte zerouri.

Exemplu

Primele 10 șiruri obținute după regulile de mai sus sunt:

```
"" 1 0 11 10 00 111 110 100 000
```

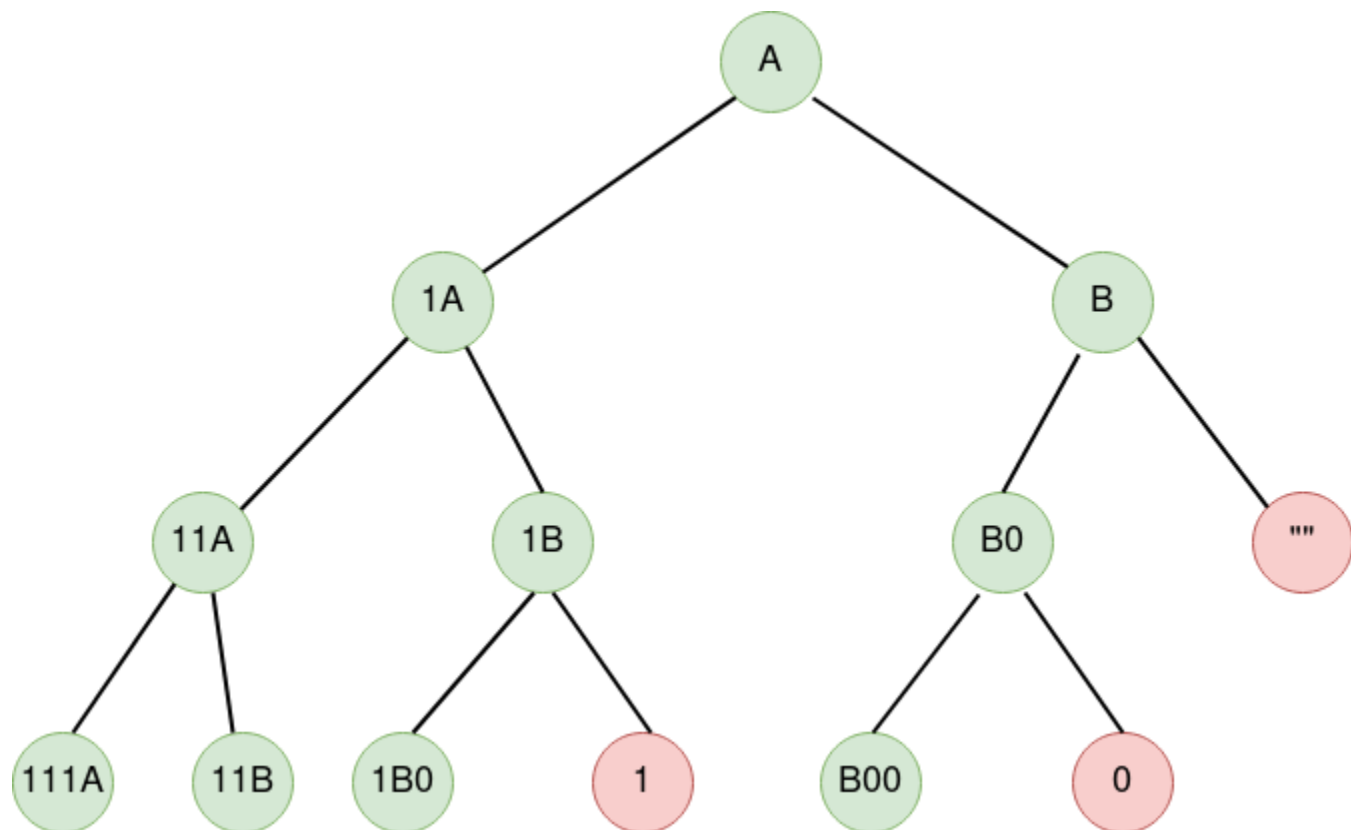
Să presupunem că programul nostru primește următoarele date de intrare:

```
2 3 7 4 -2
```

Atunci la ieșire se va afișa:

```
2 -> [ "", "1" ]
3 -> [ "", "1", "0" ]
7 -> [ "", "1", "0", "11", "10", "00", "111" ]
4 -> [ "", "1", "0", "11" ]
```

Aici este o imagine care evidențiază primele 4 nivele ale structurii de date modelată în problemă, numită arbore de derivare:



Din nefericire, codul scris de Mihai nu reușește să facă ceea ce el și-a propus. Ajutați-l pe Mihai să rezolve problemele și să facă primul pas înainte în proiectul său personal.

Download sursă [aici](#).

Reprezentanță Mașini

Descriere

O reprezentanță de mașini are un sistem de gestionare a parcului auto. Astfel, se menține o evidență a mașinilor care sunt în testare la client, a mașinilor care se vând și a celor care sunt în parcul auto. Fiecare a treia mașină va fi vândută după ce este testată în prealabil de client. Numerotarea mașinilor va începe de la 0.

Aceasta reprezentanta are însă niste bug-uri. Identificați bug-urile și rezolvați-le.

În cazul unei rezolvări de succes, programul vă trebui să afișeze id-urile mașinilor prezente în parcul auto și rularea să se încheie cu succes.

Precizări

1. Nu este permisă apelarea metodelor din clasa `Check` pentru alte utilizări decât cele din schelet.
2. Nu este permisă declararea unor simboluri (nume de variabile / metode / funcții / clase / structuri) cu numele `is_user_happy` și `get_counter`
3. Nu aveți voie să modificați liniile care conțin `You can NOT edit this.`

Input Format

Pe prima linie numărul natural N , reprezentând numărul total de mașini din parcul auto. Pe următoarele N linii câte un număr, reprezentând id-ul mașinilor aflate în gestiune.

Output Format

Pe prima linie se va afișa numărul de apeluri pentru `isUserHappy`. Pe următoarea linie id-urile mașinilor aflate în parcul auto.

Exemplu

Input

```
10
5
7
8
3
4
6
2
1
10
9
```

Output

```
11
7 8 4 6 1 10
```

Explicație

1. Inițial avem în gestiune mașinile cu id-urile 5, 7, 8, 3, 4, 6, 2, 1, 10, 9
2. Mașinile care sunt pe poziții multiplu de 3 vor fi vândute către client (indexare de la 0 adică mașina cu id = 5 se consideră pe poziția 0)
3. Rămân în parcul auto mașinile 7, 8, 4, 6, 1, 10

Download sursă [aici](#).

Wire without phone

Descriere

Clienții unui serviciu de comunicație secret au la dispoziție din partea managerului rețelei un API (application programming interface) uniform, prin care se poate programa un protocol peste canalul de comunicație dat.

Fiecare client, când este înregistrat la început în rețea, primește un număr de identificare, de la 0 la $n-1$, pe care îl știe doar managerul rețelei.

API-ul permite transmiterea de informație într-un singur sens. Cu toate astea, Florin a găsit o modalitate de a facilita și comunicația în sens invers, dar a dispărut în mod misterios.

Cerință

Se alege un emițător și un receptor de la capete opuse firului. Emițătorul vrea să îi transmită receptorului un număr într-un anumit interval.

Scopul problemei este să rezolvați problema pentru cazul când emițătorul este clientul $n-1$ și receptorul este clientul 0. Va trebui să facilitați transmiterea numerelor de la 1 la $2^{32} - 1$.

Florin, expert în securitate, a găsit o soluție prin care se poate transmite informație în sens invers, dar nu a apucat să termine de scris codul. A rămas de scris funcția `step3()`, restul codului fiind corect.

Pentru verificare, veți avea de încărcat **doar** fișierul `client.cpp`. Acesta se va verifica alături de `manager.cpp` și `client.h`, care sunt la fel cu cele pe care le primiți.

Detalii de implementare

Transmiterea de informații

În rețea clientul i poate să transmită *doar* numere naturale *doar* clientului $i+1$, prin funcția `send()`.

Ca să se poată stabili un protocol peste acele numere, fiecare client primește și o unitate de stocare, numită registru. Spre deosebire de numărul de identificare, clienții își pot citi unitatea de stocare prin funcțiile `get_register()` și `set_register()`. Deci dacă oricare client ar socoti potrivit să își scrie în registru orice valoare, poate face asta cu singura restricție să fie între 0 și `MAX_SIZE`, inclusiv.

În acest sens, `send(x)`, adaugă x la registru destinatarului.

Mai jos avem ilustrat un exemplu în care clienții fac acțiuni aleatoare, cu scop demonstrativ.

sequenceDiagram

```
participant A as Ana <br/> #35;0
participant B as Bogdan <br/> #35;1
participant C as Cristi <br/> #35;2
participant M as Manager
autonumber
```

Note over A,C: Toate registrele sunt 0.
 MAX_SIZE = 500

A->>B: send(71)

B->>A: OK

```

activate B
note right of B: registru(Bogdan) += 71
deactivate B

B--xC: send(501)
C->>B: ERR

B->>B: set_register(499)

C--xM: send(0)
M->>C: ERR

A--xB: send(2)
activate B
note right of B: 2 + registru(Bogdan) > MAX_SIZE
deactivate B
B->>A: ERR

A->>B: send(1)
activate B
note right of B: 1 + registru(Bogdan) == MAX_SIZE <br/> registru(Bogdan) = 500
deactivate B
B->>A: OK

A->>B: send(0)
activate B
note right of B: 0 + registru(Bogdan) == MAX_SIZE
deactivate B
B->>A: OK

```

Descrierea pașilor din diagrama de mai sus:

1. Clientul #0 trimite 71 clientului #1.
2. Clientul #1 are registrul 0, așa că poate adăuga 71 cu succes.
3. Clientul #1 trimite 501 clientului #2.
4. Clientul #2 are registrul 0, dar dacă s-ar adăuga 501, ar depăși MAX_SIZE. Operația nu se execută și se întoarce ERR. Valoarea din registrul clientului #2 rămâne 0.
5. Clientul #1 își setează propriul registru cu valoarea 499. Valoarea anterioară e suprascrisă.
6. Clientul #2 trimite 0 la dreapta lui.
7. Clientul #2 este ultimul și nu are cui să trimită, așa că managerul întoarce ERR.
8. Clientul #0 trimite 2 clientului #1.
9. Clientul #1 are registrul 499. Dacă ar adăuga 2, ar depăși MAX_SIZE. Se întoarce ERR.
10. Clientul #0 trimite 1 clientului #1.
11. Clientul #1 are registrul 499. Se adaugă 1 cu succes.
12. Clientul #0 trimite 0 clientului #1.
13. Operația are succes.

Observații:

1. Transmiterea de numere prin send() nu este restricționată de valoarea din registrul expeditorului, ci de cea a destinatarului.

2. Atribuirea oricărei valori în propriul registru, `set_register()`, se poate desfășura oricând îi vine rândul clientului.

Fluxul de mesaje în rețea

Pentru că API-ul este uniform, managerul a socotit că ar fie bine să secvențializeze acțiunile clienților astfel:

1. Apelează `begin_message(msg)` din partea emițătorului
2. Ciclează pentru `MAX_CYCLES`
 1. Pentru fiecare client de la 0 la `nr_clients-1`
 1. Apelează `update()` din partea acelui client

Încă un exemplu ilustrat se găsește mai jos, pentru cazul în care emițătorul este clientul 0 și receptorul este clientul `n-1`, când valoarea de transmis este mică.

```
sequenceDiagram
    participant M as Manager
    participant A as Ana <br/> #35;0
    participant B as Bogdan <br/> #35;1
    participant C as Cristi <br/> #35;2
    autonumber

    note over A,C: Toate registrele sunt 0.
    note over M: msg = 30

    M->>A: begin_message(msg)

    activate A
    note right of A: set_register(msg)
    deactivate A

    note over M,C: Încep ciclul de update().

    A->>B: send(get_register())
    B->>A: OK

    activate B
    note right of B: registru(Bogdan) += 30
    deactivate B

    B->>C: send(get_register())
    C->>B: OK

    activate C
    note right of C: registru(Cristi) += 30
    deactivate C

    note over C: Determin că sunt ultimul client. <br/> Deci acționez write().
    C-->>C: write(get_register())
```

Starea fiecărui client

Din interiorul funcției `update()` numărul de identificare al clientului curent este inaccesibil. Ca să puteți totuși reține în ce stare se află fiecare client, aveți la dispoziție structura `state`. Fiecărui client `i` se va aloca o stare, pe care o poate accesa folosindu-se de parametru al funcției `update()`, numit `state`.

Intrare

Managerul citește două variabile: numărul de clienți și numărul pe care ultimul client vrea să îl transmită primului, separate pe câte un rând nou, în această ordine.

```
nr_clients
msg
```

Ieșire

Fiecare client dispune de posibilitatea scrierii de numere la ieșirea standard, apelând `write()`. Pentru fiecare apel de `write(x)`, managerul va scrie o linie de forma `<id> write: <x>`. Vom verifica dacă receptorul, și doar el, va scrie numai și numai `msg`.

Observație: Pentru implementarea oferită deja în `client.cpp`, cea în care primul client transmite un mesaj ultimului, **NU** se va acorda niciun punctaj.

Restricții și precizări

- `MAX_SIZE` este 500
- `MAX_CYCLES` este 200
- Numărul maxim de clienți este 20
- Fișierul `client.cpp` pe care îl încărcați va fi rulat alături de `manager.cpp` și `client.h`. Acestea din urmă nu se vor schimba.

Următoarele acțiuni sunt **interzise**, fiind penalizate cu pierderea punctajului pe problemă.

- Includerea în fișierul `client.cpp` a altor headere/biblioteci decât `client.h`
- Folosirea de variabile globale sau statice

Exemplu intrare

```
5 12239
```

Exemplu ieșire

```
stdout: > 0 write: 12239
```

Ieșiri considerate invalide:

```
> 0 write: 12239
```

```
> 0 write: 12239
```

```
> 0 write: 12239
```

```
> 1 write: 12239
```

```
> 1 write: 12239
```

Download sursă [aici](#).