

Tema 2 – Structuri de date

Minimal Browser

Responsabili tema:	
Data publicarii:	6.04.2020
Termenul de predare:	26.04.2020 ora 23:55 Se accepta teme trimise cu penalizare de 10 puncte / zi (din maxim 100 puncte) pana la data de 29.04.2020 ora 23:55

1. Introducere

Dorim să implementăm functionalitatea unui browser web minimal în linie de comanda. Functionalitatea acestuia va fi apropiata de cea a unui browser clasic.

1.1. Taburi și navigare

Browserul nostru va putea crea și închide taburi și naviga printre acestea. Închiderea taburilor se va face în ordinea inversă deschiderii acestora. Astfel, ultimul tab deschis este primul care va fi închis. Totodată, din oricare dintre taburile deschise putem accesa pagini web.

1.2. Istoricul local al tabului și istoricul browserului

Fiecare tab va avea propriul istoric de navigare, întrucât ne dorim să avem posibilitatea de a ne întoarce într-o pagină vizualizată anterior (back), precum și posibilitatea de a da forward după o astfel de operație, pentru a reveni la o pagină deschisă mai târziu decât cea curentă (forward). De asemenea, orice browser web reține toate paginile accesate, astfel, trebuie implementată și această funcționalitate, ce nu ține cont de tabul de pe care a fost deschisă pagina.

1.3. Descărcări

Browserul trebuie să permită descărcarea fișierelor de diverse tipuri și dimensiuni de pe Internet într-un mod eficient. Pentru descărcarea acestora va fi pusă la dispoziție o anumită viteză de descărcare (KBps). Încărcarea oricărei pagini web va dura o secundă. Pentru trecerea timpului, se va folosi comanda “curge timpul” care înseamnă că utilizatorul va aștepta un anumit număr de secunde ca descărcările să fie finalizate.

2. Cerință

Scopul temei este de a implementa funcționalitățile browserului web descrise mai sus. O pagină web este descrisă prin URL-ul acesteia și prin resursele care pot fi descărcate de pe acea pagină:

- **url** - șir de maxim 20 de caractere;

- **num_res** - număr natural pozitiv reprezentând numărul de resurse ce pot fi descărcate de pe pagina respectivă;
- **resources** - vector de structuri de tip resursă ce poate fi descărcată de pe pagina respectivă;

Resursele vor fi generate pe baza numelui folosind o funcție deja implementată de noi și salvate într-o structură specifică:

- **id** - identificator unic pentru fiecare resursă ce poate fi descărcată de pe o pagină (în general are ca format "`<URL>-(index-ul_resursei)`");
- **size** - număr natural pozitiv reprezentând dimensiunea unui fișier (fie că este vorba de dimensiune inițială sau cât a mai rămas de descărcat din fișier);

Taburile sunt structuri ce conțin informații despre pagina curentă precum și despre paginile deschise anterior (stiva de undo / redo):

- **current_page** - referință către un obiect de tip pagină web reprezentând pagina pe care utilizatorul se află în tabul respectiv;
- **back_stack** - stivă ce salvează obiecte de tip pagini web pe care utilizatorul le-a vizualizat în trecut pe acest tab;
- **forward_stack** - stivă ce salvează obiecte de tip pagini web la care se poate reveni printr-o operație de forward;

3. Implementare

3.1. Starea inițială a browserului

În primă fază, browserul va avea un singur tab gol deschis, care nu ține referință către o pagina web, ci este un null pointer. Istoricul descărcărilor și istoricul de navigare vor fi goale. Singurele acțiuni posibile fiind deschiderea unui nou tab și accesarea unei pagini web.

3.2. Reprezentarea structurală a componentelor

Browserul trebuie să implementeze câteva dintre funcționalitățile de bază ale unui browser web obișnuit. Astfel, componentele de bază ale browserului vor fi:

- A. **Lista de taburi** - taburile se vor putea deschide și închide în cadrul unei liste (simplu sau dublu înlănțuită, la alegere) conținând elemente de tip tab; de fiecare dată când vom alege să deschidem un tab nou, acesta va fi cel mai din dreapta, iar atunci când vom dori să închidem un tab, acesta va fi tot cel mai din dreapta (adică cel mai recent deschis tab va fi cel închis);
- B. **Coadă istoricului global** - istoricul paginilor web care au fost deschise va fi reținut într-o coadă, pe care va fi posibilă operația de *clear*, care va goli primele *n* intrări din coadă, adică primele *n* pagini accesate din browser; nu este neapărat necesar să rețineți obiecte de tip pagină web în această coadă, stringul reprezentând URL-ul fiind suficient;

C. Coadă de prioritați a descărcărilor - resursele care vor fi descărcate vor fi administrate folosind o coadă de prioritați, prioritatea fiind dată de dimensiunea rămasă de descărcat; în momentul terminării unei descărcări, aceasta este scoasă din coadă și este adăugată într-o listă de descărcări finalizate. Coadă de prioritați poate conține aceeași resursă de mai multe ori dacă resursa a fost descărcată de mai multe ori;

3.3. Descărcări

Pentru a detalia mecanismul de descărcări, este necesar să înțelegem ce este o resursă. O resursă este o structură predefinită în scheletul de cod și conține drept informații:

- *char** - denumirea / id-ul resursei;
- *unsigned long* - dimensiunea resursei în octeți (Bytes);

O pagină web are în structura sa un vector dinamic de astfel de resurse, nealocat, dar care va fi inițializat cu ajutorul funcției *get_url_resources*. Această funcție primește ca parametru un string reprezentând URL-ul paginii curente și returnează un pointer la începutul vectorului menționat anterior.

După ce am accesat pagina web și cunoaștem resursele acesteia, putem alege să descărcăm o astfel de resursă. Descărcarea presupune copierea elementului de tip resursă și adăugarea lui în coada de prioritați a descărcărilor active. Descărcarea unei resurse necesită o anumită perioadă de timp. Astfel, definim două condiții de trecere a timpului:

- La introducerea comenzii *wait*, comandă la care se așteaptă un anumit număr de secunde pentru ca descărcările să avanseze;
- Încărcarea unei pagini durează o secundă;

Trecerea timpului nu este suficientă pentru a calcula cât putem descărca dintr-o resursă într-o perioadă de timp. Este necesar să definim și o variabilă de *bandwidth*, care să ne spună câți bytes putem descărca într-o secundă.

Astfel, într-o perioadă de timp *t*, având viteza de transfer *bandwidth*, putem ști sigur că am descărcat $t * bandwidth$ Bytes, care vor fi distribuiți astfel:

- Prima resursă din coadă este cea cu prioritatea cea mai mare, adică cu dimensiunea rămasă de descărcat cea mai mică, din care mai scădem $t * bandwidth$ Bytes, dacă acest număr este mai mic sau egal cu memoria rămasă de descărcat:
- Dacă mai avem de descărcat mai puțini octeți decât ne-au fost oferiți în aceasta perioadă de timp *t*, atunci descărcăm această resursă complet, o adăugăm în lista de descărcări finalizate, iar cu restul de Bytes rămași “pe cablu”, reluăm procesul cu următoarea resursă din coadă, dacă aceasta există;

- În cazul în care coada de descărcări este goală, timpul așteptat a fost consumat, iar la valoarea calculată anterior $t * \text{bandwidth}$ se renunță, aceasta ne mai fiind utilă ulterior;

În mod implicit bandwidth-ul este setat la 1 KBps (1024 B) și poate fi modificat folosind o instrucțiune de setare a bandwidth-ului.

3.4. Istoric global

Pentru a simplifica acest istoric global, este de precizat faptul că o pagină este adăugată în această coadă doar la momentul accesării ei directe, atunci când se dă `back` sau `forward` dintr-o pagină, pagina în care se ajunge nu va mai fi adăugată din nou în coadă.

4. Descrierea operațiilor și a datelor de intrare

Rezolvarea temei constă în efectuarea unui set de operații descrise în fișierul de intrare.

4.1. Setare bandwidth

Sintaxă: `set_band <bandwidth>`

Mod de funcționare: Setează valoarea bandwidth-ului (vitezei de transfer) la valoarea `<bandwidth>` KBps.

4.2. Deschidere tab

Sintaxă: `newtab`

Mod de funcționare: Creează un nou tab gol și îl adaugă în lista de taburi, pe care îl și setează ca fiind tabul curent.

4.3. Închidere tab

Sintaxă: `deltab`

Mod de funcționare: Șterge ultimul tab deschis. Dacă tabul pe care ne aflăm este cel care se închide, tabul curent va fi setat la noul ultim tab.

4.4. Schimbare tab

Sintaxă: `change_tab <tab_index>`

Exemplu: `change_tab 0`

Mod de funcționare: Schimbă tabul curent cu o referință la tabul de la indexul `<tab-index>`; Se garantează că funcția nu va schimba la un tab inexistent;

4.5. Afișare taburilor deschise

Sintaxă: `print_open_tabs`

Mod de funcționare: Afișează o listă cu informații despre taburile deschise în formatul:

```
(<index_tab>: <url>)\n
```

Dacă tabul este gol, în loc de <url> se va afișa "empty".

Exemplu: Avem 4 taburi deschise, dintre care 3 conțin pagini web deschise:

```
(0: url1.com)
(1: url2.com)
(2: empty)
(3: url3.com)
```

4.6. Accesare adresă

Sintaxă: goto <URL>

Exemplu: goto google.com

Mod de funcționare: Deschide în tabul curent pagina cu numele URL. În momentul deschiderii paginii web, o structură de pagină web va fi creată, în care vor fi populate câmpurile reprezentând **URL** (informație primită ca argument al comenzii), precum și numărul și vectorul de resursele, vector obținut folosind funcția `get_url_resources` din cadrul scheletului de laborator. Dacă exista o pagină deschisă în tabul curent, aceasta va fi pusă în stiva de back (printr-o operație de **push**). **Atunci când această instrucțiune este folosită va trece o unitate de timp (o secundă).** De asemenea, după cum se observă în orice browser, accesarea unei pagini noi dezactivează opțiunea de forward. Practic, un apel `goto` golește stiva de forward.

4.7. Accesarea paginii anterioare

Sintaxă: back

Mod de funcționare: Scoate din vârful stivei de back (printr-o operație de **pop**) a tabului curent un obiect de tip pagină web, care va fi acum pagina deschisă în tabul curent. Fosta pagină deschisă în tabul curent va fi pusă în stiva de forward (printr-o operație de **push**). Această operație este posibilă doar dacă există un obiect în stiva de back, în caz contrar operația nu va fi efectuată și se va afișa mesajul de atenționare "*can't go back, no pages in stack*".

4.8. Accesarea paginii următoare

Sintaxă: forward

Mod de funcționare: Scoate din vârful stivei de forward (printr-o operație de **pop**) a tabului curent un obiect de tip pagină web, care va fi acum pagina deschisă în tabul curent. Fosta pagină deschisă în tabul curent va fi pusă în stiva de back (printr-o operație de **push**). Această operație este posibilă doar dacă există un obiect în stiva de forward, în caz contrar operația nu va fi efectuată și se va afișa mesajul de atenționare "*can't go forward, no pages in stack*".

4.9. Afișare istoric global

Sintaxă: `history`

Mod de funcționare: Afișează istoricul global al browserului, fiecare pagină pe câte o linie în ordinea accesării.

Exemplu: Presupunem că am accesat paginile: `url1.com`, `url2.net`, `url3.ro`

La rularea comenzii se va afișa:

```
url1.com
url2.net
url3.ro
```

4.10. Ștergere intrări din istoricul global

Sintaxă: `del_history <nr_entries>`

Mod de funcționare: Șterge din istoricul global de navigare al browserului primele `<nr_entries>` adrese accesate. Dacă `<nr_entries>` este 0 sau un număr mai mare decât numărul de intrări totale din istoric, va fi șters întreg istoricul. Stivele de back și forward ale taburilor vor rămâne nemodificate.

4.11. Afișare descărcări din pagina curentă

Sintaxă: `list_dl`

Mod de funcționare: Afișează resursele descărcabile din pagina curentă, câte una pe linie, în formatul:

```
[<index_res> - "<res_name>" : <dimensiune>]\n
```

Exemplu: Pe pagina `url1.com` se află două resurse de 1MB, respectiv 25 KB:

```
[0 - "url1.com-(0)" : 1048576]
[1 - "url1.com-(1)" : 25600]
```

4.12. Afișare istoric descărcări

Sintaxă: `downloads`

Mod de funcționare: Afișează istoricul descărcărilor din browser, mai întâi pe cele nefinalizate, iar apoi pe cele finalizate, cu formatul:

```
["<res_name>" : <status>]\n
```

Unde `<status>` va fi înlocuit cu stringul `"completed"`, în cazul în care descărcarea resursei a fost finalizată, sau cu stringul `"<remaining>/<total>"`, unde `<total>` este dimensiunea resursei, iar `<remaining>` este cât a mai rămas de descărcat din aceasta.

Exemplu: Pentru 2 resurse în curs de descărcare și două finalizate, istoricul va arăta astfel:

```
["res4" : 25600/102400]
["res3" : 76800/76800]
["res2" : completed]
```

```
["res1" : completed]
```

4.13. Descărcare resursă din pagina curentă

Sintaxă: `download <index_resursă>`

Mod de funcționare: Realizează o copie a obiectului de tip resursă aflat la poziția `<index_resursă>` în vectorul de resurse a paginii curente, pe care o adaugă în coada de priorități a descărcărilor active / nefinalizate. Se garantează că nu se va încerca descărcarea unei resurse care nu există.

4.14. Comandă de wait pentru a permite avansarea descărcării resurselor

Sintaxă: `wait <seconds>`

Mod de funcționare: Realizează o trecere fictivă a timpului, care oferă timp descărcărilor să avanseze.

Exemplu: Presupunem că avem următoarea listă de descărcări:

```
["res4" : 25600/102400]
```

```
["res3" : 76800/76800]
```

Dacă avem un bandwidth setat la 1KBps, adică 1024 Bps și rulăm comanda:

```
wait 75
```

Lista de descărcări va arăta acum astfel:

```
["res3" : 25600/76800]
```

```
["res4" : completed]
```

Explicație: Comanda de wait a realizat descărcarea a $75 * 1024 = 76800$ Bytes care au fost distribuiți astfel: în vârful cozii se află `res4` pentru care s-au folosit 25600 B din cei 76800 B, astfel `res4` a fost descărcată complet și scoasă din coadă, noul vârf fiind `res3`, pentru care se reia operația cu cei $76800 - 25600 = 51200$ B rămași, care vor fi folosiți pentru descărcarea parțială a `res3`, din care vor rămâne doar 25600 B de descărcat ($76800 - 51200 = 25600$).

5. Exemple

5.1.

Intrare	Ieșire
goto google.com goto facebook.com newtab goto yahoo.com history change_tab 0 back print_open_tabs list_dl change_tab 0 newtab goto acs.pub.ro change_tab 0 forward print_open_tabs change_tab 2 list_dl del_history 1 history	google.com facebook.com yahoo.com (0: google.com) (1: yahoo.com) [0 - "google.com-(0)" : 102383015] (0: facebook.com) (1: yahoo.com) (2: acs.pub.ro) [0 - "acs.pub.ro-(0)" : 102414475] [1 - "acs.pub.ro-(1)" : 10664427] [2 - "acs.pub.ro-(2)" : 65714521] [3 - "acs.pub.ro-(3)" : 65713844] [4 - "acs.pub.ro-(4)" : 78820986] [5 - "acs.pub.ro-(5)" : 91928538] facebook.com yahoo.com acs.pub.ro

Explicație: În exemplul anterior se accesează din tabul 0 paginile google.com, iar apoi facebook.com, după care se deschide un nou tab, în care va fi accesată pagina yahoo.com. Astfel, la **afișarea istoricului**, vedem fix această ordine. Apoi se comută la tabul 0, din care se dă back, fapt ce nu influențează istoricul, dar care schimbă pagina curentă a tabului 0 înapoi în google.com. Efectul comenzii se observă la **printarea stării curente a taburilor**. Apoi se **afișează lista cu descărcările pentru pagina curentă** (de reținut că ne aflăm în tabul 0, deci pagina este google.com, care are o singură resursă de dimensiune aprox 38 MB). Schimbăm la tabul 0, iar această operație nu face nimic (întrucât eram deja acolo). Deschidem un nou tab în care accesăm pagina acs.pub.ro. Schimbarea la tabul 0 și operația de forward nu au niciun efect asupra istoricului, însă, după cum se vede la apelul funcției de **printare a taburilor**, se modifică pagina deschisă în acel tab. Comutăm apoi la tabul 2 pentru a **afișa resursele descărcabile** de pe pagina acs.pub.ro. Operația de ștergere a unui element din istoric scoate din coadă google.com, efectul fiind observat la **afișarea istoricului**.

5.2.

Intrare	Ieșire
set_band 102400	[0 - "URL2-(0)" : 86570845]
goto URL1	[1 - "URL2-(1)" : 7927001]
goto URL2	[2 - "URL2-(2)" : 62482]
list_dl	[3 - "URL2-(3)" : 76084628]
download 1	[4 - "URL2-(4)" : 5305591]
downloads	[5 - "URL2-(5)" : 21035621]
newtab	[6 - "URL2-(6)" : 13171437]
goto URL3	["URL2-(1)" : 7927001/7927001]
change_tab 0	(0: URL4)
goto URL4	(1: URL3)
newtab	(2: URL5)
goto URL5	[0 - "URL5-(0)" : 76084628]
print_open_tabs	[1 - "URL5-(1)" : 5305591]
list_dl	[2 - "URL5-(2)" : 21035621]
download 2	[3 - "URL5-(3)" : 13171437]
deltab	...
downloads	["URL2-(1)" : 7619801/7927001]
print_open_tabs	["URL5-(2)" : 21035621/21035621]
wait 25	(0: URL4)
downloads	(1: URL3)
	["URL2-(1)" : 5059801/7927001]
	["URL5-(2)" : 21035621/21035621]

Explicație: Bandwidth-ul e setat la 102400 Bps (100 KBps). Am accesat URL1 și URL2, după care am **afișat descărcările pentru URL2**. Apoi am ales să descărcăm resursa 1, pe care o adăugăm în **coada de descărcări, care urmează să fi afișată**. Deschidem două noi taburi, deschidem diverse site-uri, după care **afișăm informații legate de taburile deschise**, din care se observă paginile accesate. Ne aflăm în tabul 2 și **afișăm descărcările pentru această pagină**, mai precis URL5, dintre care descărcăm resursa 2, iar apoi închidem tabul, fapt ce nu influențează descărcarea deoarece, după **cum se observă la afișarea descărcărilor**, ambele resurse descărcate se află în coadă, iar din URL2-(1) au fost descărcați 307200 B (300 KB, datorită celor 3 goto-uri). După **afișarea taburilor rămase deschise** se așteaptă 25 de secunde, timp în care se descarcă 2560000 B (2.5 MB), folosiți pentru a reduce dimensiunea rămasă de descărcat a resursei URL2-(1), care se află în vârful cozii, la 5059801. Aceste valori sunt observate la **afișarea cozii de descărcări**.

6. Restricții și precizări

- $0 \leq \text{bandwidth} \leq 1 \text{ MB}$ (1,048,576 B)
- $0 \leq \text{linii în fișierul de intrare} \leq 1.000$
- $0 \leq \text{dimensiunea unei linii} \leq 256$
- $0 \leq \text{nr resurse pe o pagina web} \leq 12$
- $0 \leq \text{dimensiunea unei resurse} \leq 100 \text{ MB}$ (104,857,600 B)
- $0 \leq \text{seconds} \leq 1.000$, unde seconds reprezintă parametrul funcției wait
- Se garantează că datele de intrare vor fi corecte
- Programul va fi rulat astfel:

```
./tema2 <in_file> <out_file>
```
- Comenzile se citesc din fișierul in_file, iar rezultatele se scriu în fișierul out_file
- Stivele și cozile vor fi implementate folosind liste generice simplu înlănțuite - puteți folosi implementarea cozilor/stivelor din laborator (implementare cu vectori), caz în care veți fi penalizați cu 25 puncte (din max 85 puncte)
- Nu aveți voie cu variabile globale, statice
- Nu aveți voie să iterați prin structurile de tip stivă și coadă; folosiți doar operațiile specifice pentru stive și cozi
- Pentru sortări, afișări se vor folosi doar stive/cozi auxiliare (nu se permite iterarea prin stive/cozi)
- Se poate itera doar prin lista de taburi
- Implementarea funcției ce returnează resursele ce pot fi descărcate și a structurii unei astfel de resurse se regăsesc în fișierele **utils.h** și **utils.c**

7. Notare:

- **85 de puncte** obținute din testele de pe vmchecker;
- **10 puncte** pentru coding style;
- **5 puncte** pentru README - va conține detaliile de implementare a temei, precum și **punctajul obținut la teste** (la rularea pe calculatorul propriu);
- **Bonus 20 de puncte** pentru soluțiile care nu au memory leak-uri (bonusul se acordă doar dacă testul a trecut cu succes);
- Temele care nu compilează, nu rulează sau obțin punctaj 0 pe vmchecker, indiferent de motive, vor primi punctaj 0;
- Se depunțează pentru:
 - warninguri la compilare (compilati folosind -Wall);
 - linii mai lungi de 80 de caractere;
 - folosirea incorectă de pointeri;
 - ne verificarea codurilor de eroare;
 - alte situații nespecificate aici, dar considerate inadecvate;

