Tema 2 – Structuri de date (seria CB)

Minimal Browser

Responsabili tema:	Vlad Botezatu, Cristian Condruz, Cosmin-Dumitru Oprea
Data publicarii:	**6.04.2020**
Termenul de predare:	**26.04.2017 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 şi navigare

Browserul nostru va putea crea şi închide taburi şi 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 şi 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 şi posibilitatea de a da forward după o astfel de operaţie, pentru a reveni la o pagină deschisă mai târziu decât cea curentă (forward). De asemenea, orice browser web reţine toate paginile accesate, astfel, trebuie implementată şi această funcţionalitate, ce nu ţine cont de tabul de pe care a fost deschisă pagina.

1.3. Descărcări

Browserul trebuie să permită descărcarea fişierelor de diverse tipuri şi dimensiuni de pe Internet într-un mod eficient. Pentru descărcarea acestora va fi pusă la dispoziţie 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 aştepta un anumit număr de secunde ca descărcările sa fie finalizate.

2. Cerinţă

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

- **`url`** - şir 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 funcţie deja implementată de noi şi 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 fişier (fie că este vorba de dimensiune iniţială sau cât a mai rămas de descărcat din fişier);

Taburile sunt structuri ce conţin informaţii despre pagina curentă precum şi despre paginile deschise anterior (stiva de undo / redo):

- **`current_page`** - referinţă 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 operaţie de forward;

3. Implementare

3.1. Starea iniţială a browserului

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

3.2. Reprezentarea structurală a componentelor

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

A. **`Lista de taburi`** - taburile se vor putea deschide şi închide în cadrul unei liste (simplu sau dublu înlănţuită, la alegere) conţinând 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. **`Coada istoricului global`** - istoricul paginilor web care au fost deschise va fi reţinut într-o coadă, pe care va fi posibilă operaţia de *`clear`*, care va goli primele ***`n`*** intrări din coadă, adică primele ***`n`*** pagini accesate din browser; nu este neapărat necesar să reţineţi obiecte de tip pagină web în această coadă, stringul reprezentând URL-ul fiind suficient;

C. **Coada de priorităţi a descărcărilor** - resursele care vor fi descărcate vor fi administrate folosind o coadă de priorităţi, prioritatea fiind dată de dimensiunea rămasă de descărcat; în momentul terminarii unei descărcări, aceasta este scoasă din coadă şi este adăugată într-o listă de descărcări finalizate. Coada de priorităţi poate conţine aceeaşi 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 înţelegem ce este o resursă. O resursă este o structură predefinită în scheletul de cod şi conţine drept informaţii:

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

O pagină web are în structura sa un vector dinamic de astfel de resurse, nealocat, dar care va fi iniţializat cu ajutorul funcţiei `get_url_resources`. Această funcţie primeşte ca parametru un string reprezentând URL-ul paginii curente şi returnează un pointer la începutul vectorului menţionat anterior.

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

- La introducerea comenzii `wait`, comandă la care se aşteaptă 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 şi o variabilă de `bandwidth`, care să ne spună câţi bytes putem descărca într-o secundă.

Astfel, într-o perioadă de timp `t`, având viteza de transfer `bandwidth`, putem şti sigur că am descărcat `t * bandwidth` Bytes, care vor fi distribuiţi 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 puţini octeţi decât ne-au fost oferiţi în aceasta perioada de timp `t`, atunci descărcăm această resursă complet, o adăugăm în lista de descărcări finalizate, iar cu restul de Bytes rămaşi “pe cablu”, reluăm procesul cu următoarea resursa din coadă, dacă aceasta există;

- În cazul în care coada de descărcări este goală, timpul aşteptat a fost consumat, iar la valoarea calculată anterior `t * bandwidth` se renunţă, aceasta ne mai fiind utilă ulterior;

În mod implicit bandwidth-ul este setat la 1 KBps (1024 B) şi poate fi modificat folosind o instrucţiune 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 in coadă.

4. Descrierea operaţiilor şi a datelor de intrare

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

4.1. Setare bandwidth

Sintaxă: `set_band <bandwidth>`

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

4.2. Deschidere tab

Sintaxă: `newtab`

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

4.5. Afişare taburilor deschise

Sintaxă: `print_open_tabs`

Mod de funcţionare: Afișează o listă cu informaţii 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. Afişare istoric global

Sintaxă: `history`

Mod de funcţionare: Afişează istoricul global al browserului, fiecare pagină pe cate o linie in ordinea accesării.

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

La rularea comenzii se va afişa:

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

4.10. Ştergere intrări din istoricul global

Sintaxă: `del_history <nr_entries>`

Mod de funcţionare: Şterge din istoricul global de navigare al browserului primele `<nr_entries>` adrese accesate. Se garantează că `<nr_entries>` este un număr natural mai mic sau egal decât numărul de elemente din istoric. Dacă `<nr_entries>` este 0, va fi şters întreg istoricul. Stivele de back şi forward ale taburilor vor rămâne nemodificate.

4.11. Afişare descărcări din pagina curentă

Sintaxă: `list_dl`

Mod de funcţionare: Afişează 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. Afişare istoric descărcări

Sintaxă: `downloads`

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

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

Unde `<status>` va fi inlocuit 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 ramas de descărcat din aceasta.

Exemplu: Pentru 2 resurse în curs de descărcare şi 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 funcţionare: 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 funcţionare: 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]`

Explicaţie: 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	Ieşire
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)" : 39468494] (0: facebook.com) (1: yahoo.com) (2: acs.pub.ro) [0 - "acs.pub.ro-(0)" : 60414401] [1 - "acs.pub.ro-(1)" : 52550309] [2 - "acs.pub.ro-(2)" : 44685884] [3 - "acs.pub.ro-(3)" : 44685415] [4 - "acs.pub.ro-(4)" : 99735639] [5 - "acs.pub.ro-(5)" : 91871546] facebook.com yahoo.com acs.pub.ro

Explicaţie: 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 afişarea istoricului, vedem fix această ordine. Apoi se comută la tabul 0, din care se dă `back`, fapt ce nu influenţează 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 afişează lista cu descărcările pentru pagina curentă (de reţinut 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 operaţie nu face nimic (întrucât eram deja acolo). Deschidem un nou tab în care accesăm pagina `acs.pub.ro`. Schimbarea la tabul 0 şi operaţia de `forward` nu au niciun efect asupra istoricului, însă, după cum se vede la apelul funcţiei de printare a taburilor, se modifică pagina deschisă în acel tab. Comutăm apoi la tabul 2 pentru a afişa resursele descărcabile de pe pagina `acs.pub.ro`. Operaţia de ştergere a unui element din istoric scoate din coadă `google.com`, efectul fiind observat la afişarea istoricului.

5.2.

Intrare	Ieşire
set_band 102400 goto URL1 goto URL2 list_dl download 1 downloads newtab goto URL3 change_tab 0 goto URL4 newtab goto URL5 print_open_tabs list_dl download 2 deltab downloads print_open_tabs wait 25 downloads	[0 - "URL2-(0)" : 2664505] [1 - "URL2-(1)" : 28878470] [2 - "URL2-(2)" : 42514] [3 - "URL2-(3)" : 55093015] [4 - "URL2-(4)" : 26257060] [5 - "URL2-(5)" : 43592] [6 - "URL2-(6)" : 55093927] ["URL2-(1)" : 28878470/28878470] (0: URL4) (1: URL3) (2: URL5) [0 - "URL5-(0)" : 55093015] [1 - "URL5-(1)" : 26257060] [2 - "URL5-(2)" : 43592] [3 - "URL5-(3)" : 55093927] … ["URL5-(2)" : 43592/43592] ["URL2-(1)" : 28571270/28878470] (0: URL4) (1: URL3) ["URL2-(1)" : 26054862/28878470] ["URL5-(2)" : completed]

Explicaţie: Bandwidth-ul e setat la 102400 Bps (100 KBps). Am accesat `URL1` şi `URL2`, după care am afişat descărcările pentru URL2. Apoi am ales să descărcăm resursa 1, pe care o adăugăm în coada de descărcări, care urmează să fi afişată. Deschidem două noi taburi, deschidem diverse site-uri, după care afişăm informaţii legate de taburile deschise, din care se observă paginile accesate. Ne aflăm în tabul 2 şi afişăm descărcările pentru această pagină, mai precis URL5, dintre care descărcăm resursa 2, iar apoi închidem tabul, fapt ce nu influenţează descărcarea deoarece, după cum se observă la afişarea descărcărilor, ambele resurse descărcate se află în coadă, iar din URL2-(1) au fost descărcaţi 307200 B (300 KB, datorită). După afişarea taburilor rămase deschise se aşteaptă 25 de secunde, timp în care se descarca 2560000 B (2.5 MB), împărţiţi astfel:
43592 B sunt folosiţi pentru a finaliza descărcarea resursei URL5-(2), care se află în vârful cozii. Restul de 2516408 B sunt folosiţi pentru a reduce dimensiunea rămasă de descărcat a resursei URL2-(1) de la 28571270 B la 26054862 B. Aceste valori sunt observate la afişarea cozii de descărcări.

6. Restricţii şi precizări

- `0 <= bandwidth <= 1 MB (1,048,576 B)`
- `0 <= linii în fişierul de intrare <= 1.000`
- `0 <= dimensiunea unei linii <= 256`
- `0 <= nr resurse pe o pagina web <= 12`
- `0 <= dimensiunea unei resurse <= 100 MB (104,857,600 B)`
- `0 <= seconds <= 1.000`, unde seconds reprezintă parametrul funcţiei `wait`
- Se garantează că datele de intrare vor fi corecte
- Programul va fi rulat astfel:

  `./tema2 <in_file> <out_file>`
- Comenzile se citesc din fişierul `in_file`, iar rezultatele se scriu în fişierul `out_file`
- Stivele si cozile vor fi implementate folosind liste generice simplu înlănţuite - puteţi folosi implementarea cozilor/stivelor din laborator (implementare cu vectori), caz în care veţi fi penalizaţi cu 25 puncte (din max 85 puncte)
- Nu aveţi voie cu variabile globale, statice
- Nu aveţi voie să iteraţi prin structurile de tip stivă şi coadă; folosiţi doar operaţiile specifice pentru stive şi cozi
- Pentru sortări, afişări se vor folosi doar stive/cozi auxiliare (nu se permite iterarea prin stive/cozi)
- Se poate itera doar prin lista de taburi

7. Notare:

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

8. Reguli de trimitere a temelor

- Temele vor trebui încărcate atât pe `vmchecker` (în secţiunea **Structuri de Date Seria CB**) cât şi pe `acs.curs.pub.ro` în secţiunea aferentă temei 2.
- Arhiva cu rezolvarea temei trebuie să fie **.zip** şi să conţină:
    - fişiere sursă (fiecare fişier sursă va trebui sa înceapă cu un comentariu de forma: `/* NUME Prenume - grupa */`
    - fişier **README**, denumit obligatoriu astfel, care să conţină detalii despre implementarea temei
    - fişier **Makefile**, denumit obligatoriu astfel, cu două reguli:
        - **build**, care va compila sursele şi va obţine executabilul cu numele **tema2**
        - **clean**, care va şterge executabilele şi alte fişiere obiect generate
- Arhiva nu trebuie sa contina fişiere obiect sau executabile
- Dacă arhiva nu respectă specificaţiile de mai sus nu va fi acceptată la upload şi tema nu va fi luată în considerare.