

## aRS PoEtiCa

Termen de predare: **5.12.2019, ora 23:55**

Pentru fiecare zi (24 de ore) de întârziere, se vor scădea 10 puncte din nota acordată. Temele trimise după 7 de zile de întârziere vor putea fi notate cu maxim 30 de puncte.

În consecință, 🌐 deadline-ul hard este **12.12.2019, ora 23:55**.

### Întrebări

Dacă aveți nelămuriri, puteți să ne contactați pe forumul dedicat 🌐 temei de casă nr. 2.

La orice întrebare vom răspunde în maximum 24 de ore.

Nu se acceptă întrebări în ultimele 24 de ore înainte de deadline.

### Actualizări:

- **[24.11.2019 - 11:50]** funcțiile `get_friendly_word`, `get_synonym` întorc `null` dacă nu găsesc cuvântul respectiv
- **[26.11.2019 - 01:48]** added checker :), also fixed poem 3
- **[27.11.2019 - 02:00]** fixed checker :D

## Obiective Temă

---

- Utilizarea funcțiilor de manipulare a șirurilor de caractere.
- Utilizarea tablourilor.
- Formatarea unui text și procesarea după reguli date.
- Parsarea comenzilor primite la input și familiarizarea cu interactive console.

- Folosirea pointerilor pentru a transmite parametri.
- Folosirea funcțiilor din biblioteci externe.
- Utilizare random si implementare sampling.
- Folosirea utilitarului diff pe fișiere text.

## Introducere

Căpitanul Licurici primește postul de redactor al revistei clubului de literatură al UPB datorită experienței acumulate la Grădinița cu program prelungit nr. 42 ca editor de poezii pentru copii. Deși n-au trecut nici 2 luni de când a început lucrul, căpitanul Licurici e copleșit de volumul mare de muncă la care este supus, primind zeci de poezii în fiecare zi.

Acest lucru nu e deloc surprinzător, având în vedere că studenții de la Calculatoare, în special cei care sunt aici deja de 3, 4 sau chiar 5 ani, au realizat că viața lor nu începe cu #include și nu se termina cu return 0 și că pot fi persoane echilibrate care au timp și de alte activități. Planul sună foarte bine, însă asta nu face viața la Poli mai ușoară, o face doar mai frumoasă. Adevărul e că uneori unii studenți au fost nevoiți să nu doarmă câteva nopți pentru a termina o temă înainte de deadline, așa că și poeziile trimise de ei căpitanului Licurici indică semne de oboseală.

Căpitanul Licurici vă roagă să îi scrieți un program care să îi ușureze munca și să automatizeze acest proces de corectare a poeziilor trimise. Deoarece nu se pricepe să lucreze cu codul sursă, vă roagă să îi scrieți și o consolă interactivă prin care să poată preciza ce tip de procesare vrea să aplice și să obțină automat output-ul dorit. Mai multe detalii despre această consolă interactivă odată ce vor fi prezentate toate cerințele aplicației.

## Cerințe

### Cerința 1 - Uppercase

Poezia primită s-ar putea să nu respecte norma tradițională de a începe fiecare vers cu majusculă. E posibil ca în mijlocul versului să se fi apăsat shift din greșeală și să apară majuscule în mijlocul cuvintelor. În acest caz, programul scris va trebui să corecteze fiecare vers astfel încât primul cuvânt să înceapă cu majusculă, restul caracterelor alfabetice din vers să fie literă mică.

Exemplu:

```
trecE lebăDA pe ape
```

devine:

```
Trece lebăda pe ape
```

## Cerința 2 - Trimming

Din cauza oboselii, studenții au mai ațipit cu capul pe tastatură sprijiniți de cea mai mare tastă - Space. Acest lucru duce la spații prea mari între unele cuvinte, spații ce trebuie eliminate. Căpitanul Licurici vă roagă să identificați poeziile cu o spațiere nefirească și să eliminați acest whitespace redundant, astfel încât dacă între două cuvinte consecutive erau mai multe spații, acum va rămâne unul singur. În unele cazuri, studenții au mai adormit și pe alte taste, așa că pe lângă eliminarea a whitespace redundant se cere și eliminarea tuturor aparițiilor următoarelor caractere non-alfabetice: {.,'"/';:!\',?',''}

Exemplu:

```
Trece ;;; lebăda pe ....?? ape
```

devine:

```
Trece lebăda pe ape
```

## Cerința 3 - Make it silly

Căpitanul Licurici decide să îi înveselească puțin pe studenți și să le ia gândul de la iminența deadline-urilor, așa că decide să facă poeziile mai haioase. Pentru moment, doar va face schimbări aleatoare din literă mare în literă mica și invers, astfel încât versul "Trece lebăda pe ape" se va transforma în "tReCE LeBădA Pe aPe". Mai exact fiecare caracter are probabilitate  $p$  de a se schimba din majusculă / minusculă în complementara ei. Probabilitatea  $p$  va fi un parametru dat funcției.



Pentru a face sampling cu probabilitate  $p$ , generăm un număr random între 0 și 1. Dacă este mai mic decât  $p$ , se va face schimbarea majusculă ↔ minusculă. Pentru a face conversia probabilității primite ca parametru în consolă din `char*` în `float/double` check  [atof](#).



Facem sampling doar atunci când întâlnim un caracter din alfabet, restul caracterelor se ignoră.



Pentru a facilita corectarea automată, se fixează seed-ul pentru random la începutul programului cu numărul grădiniței lui Licurici, astfel 2 rulări diferite vor genera același output.

Pentru a genera un număr random între 0 și 1 folosiți următoarea secvență de cod:



```
float sample = rand() % 100 / 99.0;
```

Nu uitați de: **srand(42);** la începutul programului!

## Cerința 4 - Make it friendlier

Tot cu intenția de a mai distra studenții, redactorul vostru preferat vrea să înlocuiască unele cuvinte cu diminutivele lor. Pentru că formarea diminutivelor se bazează pe reguli mai complicate, colegii voștri mai mari au decis să dea o mână de ajutor și să ofere o bibliotecă helper ce implementează funcția de conversie către diminutiv:

```
void get_friendly_word(const char *word, char **friendly_word);
```

Această funcție primește ca parametru cuvântul `word` și returnează în `friendly_word` diminutivul corespunzător sau NULL în cazul în care nu s-a găsit un diminutiv. Implementarea voastră trebuie să treacă pe rând prin toate cuvintele din poezie, să apeleze această funcție și dacă rezultatul întors este diferit de NULL să se facă replace cu acesta în locul cuvântului aflat inițial în poezie.

De ex, versul "Trece lebăda pe ape" se va transforma în "Trece lebăduța pe ape" considerând că doar apelul pentru cuvântul "lebăda" a întors un rezultat.

## Cerința 5 - Make it rhyme

Cu toate încercările de a aduce o urmă de zâmbet, căpitanul Licurici nu e convins că a fost suficient de eficient. Dar ce poate bucura pe cineva mai tare decât un generator automat de rime? Pentru o poezie primită se va încerca formarea de rime fără a schimba semantica.

Pentru catrene, există 3 tipuri tradiționale de rimă:

- rimă împerecheată (versurile 1-2, 3-4)
- rimă încrucișată (versurile 1-3, 2-4)
- rimă îmbrățișată (versurile 1-4, 2-3)

Presupunem că această funcție primește ca parametru și tipul de rimă pe care dorește să îl obțină.

Pentru tipul de rimă "încrucișată" și următoarea poezie, algoritmul va fi:

A fost odata ca-n **basme**

A fost ca niciodată,

Din rude mari împăratești,

O prea frumoasă **duduie**.

Se observă că "basme" nu rimează cu perechea lui - "împăratești". Se cere de la biblioteca helper lista de sinonime și se primește {"zvonuri", "povești"} și se alege "povești" pentru că e mai mic lexicografic. Cuvântul "niciodată" nu rimează cu corespondentul lui - "duduie", însă în lista de sinonime întoarsă de bibliotecă nu s-a găsit nici un sinonim pentru "niciodată". Astfel, vom încerca să formăm rima din partea cealaltă. Pentru "duduie" primim lista {"puștoaică", "fată"}. Alegem "fată" pentru că este cuvântul cel mai mic lexicografic care asigură rima.

Rezultatul final obținut în urma înlocuirilor va fi:

A fost odată ca-n povești

A fost ca niciodată,

Din rude mari împăratești,

O prea frumoasă fată.



Poeziile primite sunt formate doar din catrene despărțite de un singur rând liber.



Pentru simplitate, se va considera că două cuvinte rimează dacă se termină cu același caracter. De asemenea, toate poeziile primite vor fi scrise **fără diacritice**, la fel și sinonimele și diminutivele.

Rima se verifică pentru ultimul caracter alfabetic din vers. După înlocuire punctuația trebuie să se păstreze.

De exemplu, pentru versurile:

A fost ca **nicicând**,

O prea frumoasă fată.



Se va înlocui "nicicând" cu "niciodată", dar virgula de la finalul primului vers va fi păstrată. Se obține:

A fost ca niciodată,

O prea frumoasă fată.

## Cerința 0 - Interactive Console

Se va implementa un parser al liniei de comandă pentru a putea lucra interactiv cu transformările descrise în cerințele de mai sus și pentru a putea vedea la fiecare pas output-ul rezultat. Programul scris va citi în continuu de la standard input și va putea primi următoarele comenzi, pentru fiecare din ele trebuind să apeleze funcția de la cerința corespunzătoare.

- load path
  - pe lângă skelet-ul oferit, va exista și o bibliotecă auxiliară pentru citirea poeziei, încărcarea listei de diminutive și a listei de sinonime
- uppercase
- trimming
- make\_it\_silly prob
- make\_it\_friendlier
- make\_it\_rhyme rhyme-type
- print
  - afișează output-ul rezultat în urma secvenței curente de transformări aplicate
- quit
  - iese din program



Se garantează faptul că atât operațiile cât și parametrii trimiși la input sunt corecte.

## Restricții și precizări



Studentii au voie să modifice structura internă a programului în orice fel doresc. Pot formata și salva poezia în orice mod le face implementarea mai ușoară (în cod!). De aceea, semnăturile funcțiilor nu vor veni în schelet și fiecare le va alege în funcție de reprezentarea folosită. La apelarea funcției print programul va trebui să transforme poezia din reprezentarea internă și să o printeze la output. Toate cerințele sunt deterministe, funcția print trebuie să producă același output pentru toată lumea, indiferent de reprezentarea internă aleasă.

În cadrul scheletului este oferit și o bibliotecă statică (mai multe detalii în anul 3 SO). Trebuie să adăugați header-ul `task_helper.h` din folderul `util` în programul vostru.

```
/*
 * Load a file inside a given buffer
 * @param file - name of the file to be read
 * @param buffer - initialised vector to hold the content of the file
 */
void load_file(const char *file, char *buffer);

/*
 * Returns a list of synonym words and its length based on a given word.
 * @param word - the given word
 * @param n - the length of the returned list
 * @param word_list - returns the pointer to a list filled with synonym words having
 */
void get_synonym(const char *word, int *n, char ***word_list);

/*
 * Receives a word and returns the pointer to a location of memory that holds a friend
 * @param word - the given word
 * @param friendly_word - returns the pointer to a memory where the word is located
 */
void get_friendly_word(const char *word, char **friendly_word);
```

## Evaluare



Se va incepe implementarea obligatoriu cu **Cerința 0 - Interactive Console**. Checker-ul va trimite comenzi pe care sursa voastră trebuie să le suporte și să ia acțiunile corespunzătoare. E suficient să implementați handler in consola interactivă doar pentru subsetul de cerințe pe care le-ați implementat.

## Trimitere temă

Tema va fi trimisă folosind

Găsiți poeme pentru temă [aici](#).

Găsiți scheletul temei si makefile-ul [aici](#).


Găsiți arhiva cu checker-ul [aici](#).

Punctajul:

- 125p - testele
- 20p - coding style
- 5p - readme

După cum probabil ați observat, task-urile au un total de **150 de puncte**, dar motivul pentru acest lucru este ca fiecare test să valoreze **2.5 punct**. Dacă soluția voastră trece toate testele veți acumula un total de **1.25 puncte** (din nota finală). Celelalte **0.25 puncte** se vor acorda pentru **coding style** și **README**.

Formatul arhivei va fi următorul:

1. Fișierele necesare compilării.
  - a. Folderul util care conține biblioteca data de noi.
2. Fișierul **Makefile** dat de noi care să conțină următoarele reguli:
  - a. **build**: creează executabilul aferent (numele executabilului: **tema2**)
  - b. **run**: rulează executabilul aferent
  - c. **clean**: șterge fișierele obiect/executabile create.
3. Un fișier  **README** în care vă descrieți rezolvarea fiecărui task.



1. Arhiva trebuie să fie de tipul **zip**.
2. Inputul se va fi citit de la **stdin (tastatura)**, iar output-ul va fi afișat la **stdout (ecran)**. Testarea se face cu ajutorul redirectării acestora din linia de comandă/bash.

## Listă depunctări

Lista nu este exhaustivă.

- o temă care nu compilează și nu a rulat pe **vmchecker** nu va fi luată în considerare
- o temă care nu rezolvă cerința și trece testele prin alte mijloace nu va fi luată în considerare
- [-1.0]: warning-uri la compilare (este obligatorie folosirea în fișierul **Makefile** a flag-ului de compilare **-Wall** pentru regula **build**)
- [-1.0]: numele variabilelor nu sunt sugestive
- [-1.0]: linii mai lungi de 80 de caractere
- [-1.0]: cod nemodular, funcții prea lungi (inclusiv main)
- [-2.5]: variabile globale
- [-5.0]: abordare ineficientă
  - în cadrul cursului de programare nu avem ca obiectiv rezolvarea în cel mai eficient mod posibil a programelor; totuși, ne dorim ca abordarea să nu fie una ineficientă, de genul să nu folosiți instrucțiuni repetitive acolo unde clar nu era cazul, etc.



