Codificator de șiruri

Programarea calculatoarelor (2015-2016)
Tema 1

20.11.2015
Deadline: 12.12.2015, 23:55

1 Cerință

Scopul acestei probleme este de a compune un mesaj codificat plecând de la datele de intrare și algoritmul descris mai jos. Programul citește de la tastatură caractere, cuvinte sau numere, fiecare pe câte o linie, și afișează la consolă mesajul rezultat în urma codificării.

1.1 Task 1 - Citire (25p)

Implementați citirea corectă a input-ului primit și identificați tipul de termen citit de pe fiecare rând (caracter, cuvânt sau număr).

Tipul termenului este important deoarece acesta determină modul în care va fi prelucrat termenul respectiv pentru a putea fi adăugat la mesajul final codificat.

Ca input se primesc mai mulți termeni, fiecare pe câte o linie, însă fără a se ști numărul acestora. Se știe însă că termenii sunt urmați de un număr de componente n (care nu va fi prelucrat pentru a fi adăugat la mesaj; semnificația lui va fi explicată ulterior), și de cuvântul `END`, care oprește citirea (nu se mai citește nimic după întâlnirea lui și, de asemenea, nu va fi prelucrat).

Observație: Termenii din input vor conține numai caractere reprezentabile, care au codurile ASCII în intervalul $[32, 127)$. Acestea sunt litere, cifre, semne de punctuație și alte câteva caractere ușor identificabile pe tastatură.

1.1.1 Tipurile de termeni

- **Cuvinte.** Acestea nu sunt cuvinte propriu-zise, ele pot conține și cifre și semne de punctuație.
  Exemple: `hieAyger` `45d/4"` `-3OS>,p=` `bn7Bzva` `5654322112522598R`
  Observație: Cuvintele citite vor avea maxim 50 de caractere.
- **Caractere.** Orice caracter care se găsește singur pe un rând nou citit și care nu reprezintă o cifră.
- **Numere.** Orice numere între -2147483648 și 2147483647.
  Observație: Numerele nu vor conține cifra 0, iar numerele pozitive nu vor avea caracterul + în fața lor.

Numărați tipurile de termeni și afișați în consolă, pe o linie, următorii 3 întregi:

- numărul de termeni de tip cuvânt;

- numărul de termeni de tip caracter;
- numărul de termeni de tip număr.

1.2 Task 2 - Codificare (40p)

Prelucrați corect toți termenii citiți la taskul anterior și adăugați-i la mesajul codificat.

- Pentru **cuvinte** (25p), se determină cel mai mare divizor d al numărului de caractere l al cuvântului, divizor care trebuie să fie diferit de numărul însuși $d \neq l$ (**exemple:** 7 pentru 21, 14 pentru 28, 1 pentru 3). Primele d caractere ale cuvântului se mută de la început la finalul acestuia. În cazul în care cuvântul nu conține cifre, restul caracterelor vor fi păstrate în ordinea inițială, altfel vor fi în ordine inversă (dacă există cifre în cuvânt).

  Exemple:

  - `ce4ntauri` va fi transformat în `iruatnce4` (cuvântul are 9 caractere, iar 3 este cel mai mare divizor al lui 9. `ce4` se mută la sfârșit, iar `ntauri` devine `iruatn`, deoarece cuvântul conține o cifră).
  - `carte` va fi transformat în `artec` (cuvântul are 5 caractere, divizorul determinat este 1, iar cuvântul nu conține nicio cifră)

  Cuvântul astfel prelucrat se adaugă la mesajul final.

- Pentru **caractere** (5p), se determină din mesajul obținut până în acel moment caracterul cu cel mai mic număr de apariții în mesaj, respectiv caracterul cu cel mai mare număr de apariții (în numărarea aparițiilor se va face distincție între litere mari și mici). Dacă există mai multe caractere cu număr de apariții maxim sau minim, se va lua în considerare cel care apare primul în mesaj. Înaintea caracterului citit de la tastatură se adaugă caracterul determinat ca având cel mai mare număr de apariții în mesaj, iar după caracterul citit se adaugă caracterul cu cel mai mic număr de apariții. Aceste 3 caractere determinate astfel se concatenează la mesajul final.

  Observație: Dacă șirul inițial este gol, caracterele cu un număr minim sau maxim de apariții nu există și se va insera doar caracterul citit.

  Exemplu: Avem până acum mesajul: `t49h'=v5y;ef-ffv3vkrj4b/` și se citește caracterul `>`. Primul caracter care apare de cele mai puține ori este `t` (o apariție), iar primul caracter care apare de cele mai multe ori este `v` (3 apariții). Astfel, la mesaj se adaugă `v>t` și mesajul devine: `t49h'=pv5y;ef-ffv3vkrj4b/v>t`

- Pentru **numere** (10p): dacă termenul citit este un număr pozitiv, atunci se determină cel mai mare număr ce se poate obține din acesta prin rotație ciclică a cifrelor sale. Dacă termenul este un număr negativ, se ia în valoare absolută și se determină cel mai mic număr ce poate fi obținut prin rotația cifrelor. Numărul astfel determinat se adaugă la mesajul final.

  Exemple:

  - pentru numărul -4723275 se obține numărul 2327547
  - pentru numărul 349528973 se obține numărul 973349528

După construirea șirului se afișează șirul codificat, urmat de newline.

1.3 Task 3 - Rearanjare (25p)

La sfârșit se ia mesajul final obținut și se împarte într-un număr de bucăți egal cu numărul de componente n citit imediat înaintea întâlnirii lui `END`. Toate bucățile vor avea număr egal de caractere. În cazul în

care împărțirea caracterelor mesajului în n componente nu se face exact (împărțire cu rest), ultimele caractere vor face parte din ultima componentă de mesaj.

Fiecare bucată de mesaj se reține într-o structură ce conține și un câmp pentru complexitatea componentei. Structura va conține, deci, câmpurile:

- șir de caractere (string) – de tip `char*`
- complexitate – de tip `double`

Veți utiliza astfel un vector de structuri ce va avea n structuri pentru stocarea întregului șir.

Observații:

- n va fi cuprins între 1 și 30. Pentru câmpul de tip `char*` va trebui să alocați spațiu exact cât este necesar, nu mai mult!
- Complexitatea unui șir se calculează ca fiind media codurilor ASCII ale caracterelor ce compun bucata de mesaj din structura respectivă.
- Daca doua siruri au aceeasi complexitate, se sorteaza lexicografic.

Odată creat vectorul de structuri, acesta va fi ordonat descrescător după complexitățile componentelor de mesaj din structuri. După ordonare, mesajul se va recompune prin concatenarea bucăților ce ocupă următorii indecși în vector, în această ordine: 0, $n-1$, 1, $n-2$, 2, $n-3$, 3, etc. Mesajul astfel recompus se afișează în terminal, urmat de newline.

Observație: Mesajul final va conține maxim 5000 de caractere.

2 Date de intrare

Se vor citi, de la tastatură, mai mulți termeni, câte unul pe linie. La final, se va citi un număr n și cuvântul `END`.

3 Date de ieșire

Se vor afișa, în consolă, trei linii:

- trei întregi, separați de un spațiu care reprezintă:
  - numărul de termeni de tip cuvânt;
  - numărul de termeni de tip caracter;
  - numărul de termeni de tip număr;
- un șir de caractere reprezentând mesajul codificat (înainte de rearanjare);
- un șir de caractere reprezentând mesajul codificat (după rearanjare).

4 Restricții și precizări

- Temele sunt strict individuale! Copierea temelor va fi sancționată. Persoanele cu porțiuni de cod identice nu vor primi niciun punctaj pe temă.
- Temele trimise dupa deadline nu vor fi luate în considerare (nu vor fi punctate).

- **Separați logica programului în mai multe funcții.**
- **Este interzisă folosirea variabilelor globale.**
- Soluția temei va fi scrisă în C. Nu folosiți sintaxă sau instrucțiuni specifice limbajului C++.
- Rezolvarea temei va fi scrisă într-un fișier numit `encoder.c`, iar executabilul generat se va numi `encoder`. Pot fi folosite oricâte fișiere sursă sau alte fișiere auxiliare.
- Fișierul `Makefile` va conține cel putin 3 reguli: `build` (implicit), `run` și `clean`. Regula `build` va genera executabilul, regula `run` va executa tema (dacă nu există executabilul, acesta va fi creat), iar `clean` va șterge toate fișierele create de compilare și execuție.
- În `README` precizați cât timp v-a luat implementarea programului vostru și explicați, pe scurt, implementarea temei (comentariile din cod vor documenta mai amănunțit rezolvarea).
- Este recomandat ca liniile de cod și cele din fișierul `README` să nu depășească 80 de caractere.
- Pentru a introduce mai ușor datele în program, puteti redirecționa input-ul (ex. `./encoder < input.txt`). Alternativ, puteti folosi functia `freopen`. Înainte de efectua orice citire sau scriere, apelați:

  ```
  freopen("input.txt", "r", stdin);
  freopen("output.txt", "w", stdout);
  ```

  Aveti grijă să ștergeți aceste instrucțiuni înainte de a trimite tema!
- Folosiți un coding style astfel încât codul să fie ușor de citit și înțeles. De exemplu:
  - Dați nume corespunzătoare variabilelor și funcțiilor.
  - Nu adăugați prea multe linii libere sau alte spații goale unde nu este necesar (exemplu: nu terminați liniile în spații libere, trailing whitespaces; nu adăugați prea multe linii libere între instrucțiuni sau la sfârșitul fișierului). Principalul scop al spațiilor este identarea.
  - Fiți consecvenți. Coding style-ul are o natură subiectivă. Chiar dacă pentru unele persoane nu pare bun, faptul că îl folosiți consecvent este un lucru bun.
  - Există programe sau extensii pentru editoare text care vă pot formata codul. Deși vă pot ajuta destul de mult, ar fi ideal să încercați să respectați coding style-ul pe măsură ce scrieți codul.
  - Pentru sugestii de coding style, puteți intra *aici* și *aici*.
- Veţi trimite o arhivă ZIP cu numele de tipul `GRUPA_Nume_Prenume.zip` (exemplu: `311CC_Popescu_Maria_Ioana.zip`), care va conţine fișierele necesare, `Makefile` și `README`. Fșierele trebuie să fie în rădăcina arhivei, nu în alte subdirectoare.
- Compilarea nu ar trebui să producă avertizări (verificați prin adăugarea flagului `-Wall` la `gcc`).
- **Eliberați memoria alocată dinamic.** Folosiți `valgrind --tool=memcheck --leak-check=full ./encoder` pentru a verifica dacă memoria este eliberată corect.

5 Exemplu

INPUT:

```
5=frdg43
-48926
p
!
1
rgd';esfthjgfjyfzAjiN
485
5
END
```

OUTPUT:

```
2 2 3
34gd5=fr264894p34!g1fthjgfjyfzAjiNrgd';es854
gfjyfzAj264894p34!g1fthj34gd5=friNrgd';es854
```

6 Indicații de rezolvare

- Orice coding style sugereaza structurarea programelor în mai multe funcții. Creați câte o funcție pentru:
  - citirea inputului;
  - a determina tipul termenului;
  - a prelucra fiecare tip de termen;
  - sortare;
  - rearanjarea șirului.
- Sortarea poate fi făcută folosind `qsort` din `stdlib.h`.

7 Notarea temei

- **25p** – Task 1 - Citire
- **40p** – Task 2 - Codificare
  - **25p** – Codificarea corectă a cuvintelor.
  - **5p** – Codificarea corectă a caracterelor.
  - **10p** – Codificarea corectă a numerelor.
- **25p** – Task 3 - Rearanjare
  - **20p** Rearanjarea corectă a subșirurilor.
  - **5p** Eliberarea memoriei după rearanjare.
- **10p** – Coding style, `Makefile` și `README`
  - **4p** – Coding style
    - comentarii
    - folosirea a mai multor funcții
    - logică clară
    - spațiere corectă
    - stil consecvent
    - variabile și funcții denumite adecvat
  - **4p** – Crearea fișierului `Makefile`
  - **3p** – Completarea fișierului `README`

8 Editări

- **23.11.2015, 00:00** – Corecturi gramaticale.
- **07.12.2015, 22:00** – Incarcare checker. Modificare observatii task 3.
- **08.12.2015, 00:00** – n poate avea valori in intervalul $[1, 30]$, nu $[2, 30]$.

Responsabili temă: Alin Mîndroc <mindroc.alin@gmail.com>, Dan Ungureanu <dan@ungureanu.me>
Autor original: Victor Doca