Programarea calculatoarelor (2018-2019)
Tema 1 - Telefon

Deadline: duminica 09.12.2017, ora 23:55

UPDATE 22.11: Clarificari + adaugare mai multe exemple cerinta 4 (mesaj perfect)

Responsabili temă: Paul Ungureanu, Raluca Radu

Gigel este un student proaspăt admis la Politehnică, foarte pasionat de tehnologie. El are de gând să facă un telefon, dar până la a face un telefon extrem de performant, acesta trebuie să stabilească bazele. Drept urmare, el intenționează să proiecteze un telefon cu butoane cu tastatură de tip ***T9***.

```
======================
|   1   |   2   |   3   |
|       |  abc  |  def  |
|======|=======|======|
|   4   |   5   |   6   |
|  ghi  |  jkl  |  mno  |
|======|=======|======|
|   7   |   8   |   9   |
| pqrs  |  tuv  | wxyz  |
|======|=======|======|
|       |   0   |       |
|       |(space)|       |
======================
```

Pentru a introduce un text, Gigel procedează astfel: dacă apasă o tasta de *n* ori, se va tasta a *n*-a literă de pe tasta respectivă. În cazul în care *n* e mai mare decât numărul de litere inscripționate pe tastă, se va relua numărătoarea de la capătul șirului.

Exemplu:

Apasand pe "2":

- o data, el va introduce litera "a"
- de 2 ori, el va introduce litera "b"
- de 3 ori, el va introduce litera "c"
- de 4 ori, el va introduce litera "a"
- de 5 ori, el va introduce litera "b"
- de 6 ori, el va introduce litera "c"
- ...
- de 100 ori, el va introduce litera "a"

Mai mult, pentru a putea introduce rapid majuscule, Gigel a decis ca cifra *1* înaintea unei alte cifre să semnifice că următoarea literă reprezintă o majusculă.
Exemplu:

```
18 -> T
812 -> tA.
```

Gigel a observat însă ca există o problemă și că nu poate scrie cuvântul ***tu*** deoarece telefonul ar interpreta codul ca fiind litera ***v*** (cod 888). Drept urmare, a decis ca:

- În cazul în care se tastează doua litere de pe aceeași tastă (precum ***t*** si ***u***), se vor despărți prin ***#***.
- În cazul în care se tastează două litere de pe aceeași tastă, dar a doua e majusculă, caracterul ***#*** e inutil (deoarece va fi inlocuit de ***1***).
- În cazul în care se tastează doua litere de pe taste diferite (precum ***t*** și ***o***), nu se vor desparți prin ***#***.

Un mesaj necodificat va conține DOAR litere mici, mari și spații, fără niciun alt tip de caracter special. Pentru că Gigel nu vrea să facă febră musculară la degete, acesta nu va introduce mesaje mai mari de **100** de caractere, iar mesajele codificate nu vor depăși **300** de caractere.

Exemplu:

```
888    -> v
8#88   -> tu
8188   -> tU
88#8   -> ut
86668   -> tot. Atentie "8#666#8" nu îi convine lui Gigel deoarece ar consuma mai multă memorie.
```

Cerința 1

Ajutați-l pe Gigel să scrie un text. El vă va spune ce vrea vrea să scrie, iar voi va trebui să ii spuneți ce taste să apese. Vom numi aceasta operație: codificarea unui text.

Exemplu:

```
Tu ai mere  ->    18#8802444063377733
```

Cerința 2

Ajutați-l pe Gigel să citească un text. El vă va spune ce taste apasă, iar voi va trebui să îi spuneți ce a scris. Vom numi aceasta operație: decodificarea unui text.

Exemplu

```
17#777666477726277733  ->  Programare
```

Cerința 3

Pentru că greșește foarte des, Gigel a decis să implementeze funcția de *auto-correct*. Astfel, dându-se un *dicționar* de cuvinte, Gigel va trebui să înlocuiască cuvintele din text cu cele din dicționar.

Exemplu:

Pentru propoziția

```
Eu am mere
```

și dicționarul:

```
3
Eu Tu
am ai
mere Pere
```

Se va obține:

```
Tu ai Pere
```

Dicționarul se va da astfel:

```
<Număr cuvinte>
<Cuvânt gresit> <Cuvânt corect>
...
<Cuvânt gresit> <Cuvânt corect>
<Cuvânt gresit> <Cuvânt corect>
```

Se va da un nou text **necodificat**. Afișați **codificarea** lui după ce l-ați corectat.
Implementați dicționarul sub forma unui vector de structuri pe care să îl alocați **dinamic**. Alocați **exact** cât spațiu aveți nevoie, nu mai mult.

Cerința 4

Un mesaj ***codificat*** este considerat ***prim*** dacă toate componentele sale sunt prime între ele două câte două. O componentă este o succesiune de cifre, care face abstracție de caracterul ***#***, a unui mesaj codificat. Două componente sunt separate prin cifra ***0***. O componentă va putea fi stocată într-o variabilă de tip **long long** și tratată ca un număr. Două componente sunt prime intre ele dacă cel mai mare divizor comun este 1.

Un mesaj ***codificat*** este considerat ***perfect*** dacă mesajul codificat (**in care se ignora #**) se împarte perfect în numere de *N* cifre, unde numărul ***magic*** *N* reprezintă suma **cifrelor** mesajului codificat, modulo ***9***. Dacă ***N*** e ***0***, atunci mesajul nu este magic. Aceste grupuri de **N** cifre pot fi stocate în variabile de tip **long long** și vor fi tratate ca numere (dacă numărul începe cu 0, se va ignora cifra 0 de la început, dar va fi considerat în continuare un grup valid, chiar dacă mai puțin de ***N*** cifre). Exemple:

- Mesajul codificat 020202020202 (**N = 3**), va fi considerat valid, împărțindu-se în 20, 202, 20, 202
- Mesajul codificat 122#2#23 (**N = 3**) va fi considerat valid, împărțindu-se în 122, 223
- Mesajul codificat 6016 (**N = 4**) va fi considerat valid, împărțindu-se în 6016
- Mesajul codificat 020513 (**N = 2**) va fi considerat valid, împărțindu-se în 2, 5, 13.
- Mesajul codificat 2#12 (**N = 5)** va fi considerat invalid (212 nu se imparte in grupuri de 5 cifre)

Se va da un nou text **necodificat**. După ce l-ați **codificat** (fără dicționar):

- Afișați numărul de componente.
- Afișați componentele.
- Afișați cea mai mare componentă și poziție ei în șirul de componente.
- Afișați ***1*** dacă mesajul este *prim*, respectiv ***0*** dacă mesajul nu este *prim*. Dacă alegeți să folosiți un vector pentru a stoca componentele, alocați dinamic memoria.
- Verificați dacă mesajul este *perfect*. Dacă mesajul este *perfect*, afișați grupurile de *N* caractere în ordine **descrescătoare**, în caz contrar, afișați ***0***. Având în vedere că știți numărul de grupuri, alocați **dinamic** memoria vectorului pe care îl veți folosi. **Nu alocați mai multă memorie decât veți folosi**.

Exemplu:

```
tU ai mere -> 818802444063377733
3                       // nr componente
8188 2444 63377733      // componentelele
63377733 3              // cea mai mare componenta si pozitia ei
0                       // mesajul nu e prim
818802 444063 377733    // mesaj perfect
```

Date de intrare

Se vor citi de la tastatură datele pentru fiecare cerință. Găsiți un exemplu mai jos. Dacă nu puteți rezolva o anumită cerință, dar puteți rezolva o cerința următoare, inputul pentru cerința nerezolvată tot va trebui citit.

Date de ieșire

Se vor afișa în consolă răspunsurile pentru fiecare cerință. Găsiți un exemplu mai jos. Dacă nu puteți rezolva o cerință (sau subcerință) lăsați un număr de rânduri libere câte ar fi ocupat cerința dacă ar fi fost rezolvată (pentru cerințele 1, 2, 3 și 5, un rând, iar pentru cerința 4, 5 rânduri).

Exemplu 1

Intrare:

```
Salut               // Cerinta 1
17777255588#8       // Cerinta 2
Sal                 // Cerinta 3
1
Sal Salut
SA LU TA RE         // Cerinta 4
```

Ieșire:

```
17777255588#8  // Cerinta 1
Salut          // Cerinta 2
17777255588#8  // Cerinta 3
4              // Cerinta 4 - numar componente
1777712 1555188 1812 1777133  // componentele
1777712 1                     // componenta maxima + pozitia
0                             // nu este prim
8018120 1777712 1777133 155518     // afisarea grupurilor
```

Exemplu 2

Intrare:

```
Politehnica                   // Cerinta 1
176665554448334466444222#2      // Cerinta 2
Ana are mere                  // Cerinta 3
5
Ana Poli
AnA ION
are tehni
mere ca
ca nu
test tesw                     // Cerinta 4
```

Ieșire:

```
176665554448334466444222#2         // Cerinta 1
Politehnica                   // Cerinta 2
1766655544408334466444O222#2  // Cerinta 3
```

```
2                               // Cerinta 4 - nr componente
83377778 83377779               // componente
83377779 2                      // componenta maxima + pozitia
1                               // este prim
0                               // nu este perfect
```

Indicații de rezolvare

- Pentru dicționar, puteți folosi o structură de date formată din două elemente care să memoreze cuvântul greșit și cuvântul corect.
- Puteți folosi funcția ***qsort*** pentru sortare.

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 după deadline nu vor fi luate în considerare.
- 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 ***telefon.c***, iar executabilul generat se va numi ***telefon***. Pot fi folosite oricâte fișiere sursă sau alte fișiere auxiliare.
- Fișierul ***Makefile*** va conține trei 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, puteți redirecționa input-ul (ex. ./telefon < input.txt). Alternativ, puteți folosi funcția *freopen*. Înainte de efectua orice citire sau scriere, apelați:

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

  Aveți 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 comanda de mai jos pentru a verifica dacă memoria este eliberată corect.

```
valgrind --tool=memcheck --leak-check=full ./telefon
```

- **Temele trebuie să fie încărcate pe vmchecker. NU se acceptă teme trimise pe e-mail sau altfel decât prin intermediul vmchecker-ului.**

Notare

- **25p** - Cerința 1
- **25p** - Cerința 2
- **30p** - Cerința 3
- **50p** - Cerința 4
  - **5p** - Afișare număr componente
  - **5p** - Afișare componente
  - **10p** - Afișare cea mai mare componentă și poziția ei în șir.
  - **15p** - Verificare și afișare mesaj *prim*
  - **15p** - Verificare și afișare mesaj *perfect*
- **20p** - *Coding style*, *Makefile*, *README*
  - **10p** - **Coding Style**
    - Comentarii
    - Folosirea a mai multor funcții
    - Logică clară
    - Spațiere corectă
    - Stil **consecvent**
    - Variabile și funcții denumite adecvat
  - **5p** - Crearea fișierului **Makefile**
  - **5p** - Completarea fișierului **README**