Laboratorio di Programmazione
(Corso di Laurea in Informatica)

Simulazione di prova d'esame - Dicembre 2018

Avvertenze

- VERRANNO CORRETTI SOLO E SOLTANTO I COMPITI IL CUI ESERCIZIO FILTRO FUNZIONA PERFETTAMENTE.
- ATTENZIONE ALLA CASE-SENSITIVENESS!!! I nomi dei file devono rispettare le specifiche (ad es. `Filtro.go` è DIVERSO da `filtro.go`).
- I programmi realizzati DEVONO **rispettare le specifiche fornite sull'input e sulla forma dell'output**, senza aggiungere messaggistica ulteriore o utilizzare termini e/o spaziatura differenti. Parte della valutazione è infatti su comprensione e rispetto delle specifiche.
- Si raccomanda di salvare, compilare e fare upload delle soluzioni con una certa regolarità. In ogni caso NON sarà riaperta la sessione di upload per chi non ha consegnato in tempo.
- Per la procedura di **consegna** si veda in fondo al documento.

1 ESERCIZIO FILTRO

1.1 Descrizione

Scrivere un programma `filtro.go` che, data su standard input una sequenza di voti (numeri interi), separati da caratteri di spaziatura (bianchi, tab, a-capo) e compresi tra 18 e 30, produca in uscita le percentuali (troncate al valore intero) dei voti tra 18 e 21 (D), tra 22 e 24 (C), tra 25 e 27 (B), tra 28 e 30 (A).
Il formato per l'output deve essere (vedi anche esempio sotto): per ogni fascia di voti, partendo da quella più alta, voto in lettere, spazio, ':', spazio, percentuale, spazio, carattere '%'.
Il programma deve ignorare eventuali valori fuori dall'intervallo [18-30] e terminare la lettura su input 0.

1.2 Vincoli

Non ci sono vincoli sul numero di dati in ingresso.

1.3 Esempio

Per il seguente input

```
28 22 25 18 21 26 14 22 26 30 23 27 24 28 25 0
```

l'output di `filtro.go` sarà:

```
A : 21 %
B : 35 %
C : 28 %
D : 14 %
```

2 Tic

2.1 Descrizione

Scrivere un programma `tic.go` dotato delle seguenti funzioni:

- una funzione `newOrario(s, m, h int) (bool, Orario)` che, dati come parametri tre interi (che rappresentano nell'ordine secondi, minuti e ore) restituisce:
  1. true o false, a seconda che i parametri rappresentino un orario valido (secondi e minuti tra 0 e 59, ore tra 0 e 23) o no;
  2. nel caso i parametri rappresentino un orario valido, una struttura `Orario` che ha come campi (int) secondi, minuti e ore (quest'ultime in formato h24), che contenga l'orario rappresentato dai parametri;
- una funzione `tic(orario *Orario)` che aggiorna un orario mandandolo avanti di un secondo e gestendo correttamente la fine di un minuto, di un'ora e di una giornata.
- una funzione `main()` che legge da linea di comando quattro numeri interi, di cui il primo rappresenta i secondi, il secondo i minuti e il terzo le ore di un orario, e il quarto un tempo in secondi. Se i primi tre argomenti sono valori validi per un orario, il programma crea l'orario corrispondente, lo visualizza su standard output nel formato s:m:h, lo manda avanti del tempo in secondi letto, e lo visualizza di nuovo. Altrimenti visualizza il messaggio "parametri non validi".

2.2 Vincoli

Non occorre fare controlli sui valori passati da linea di comando, si può assumere che siano sempre numeri interi.

2.3 Esempi

```
$ go run tic.go 65 3 2 56789
```

darà come output:

```
parametri non validi


$ go run tic.go 5 3 2 56789
```

darà come output:

```
5:3:2
34:49:17
```

3 Sostituzioni

3.1 Descrizione

Scrivere un programma `sostituzioni.go` dotato delle seguenti funzioni:

- una funzione `func sostituisci(s, old, new []byte, n int) []byte` che, date tre slice di byte, *s*, *old*, *new*, e un intero *n*, restituisce una slice corrispondente a *s* in cui la *n*-esima occorrenza senza sovrapposizioni di *old* viene sostituita con *new*; si noti ad esempio che c'è una sola occorrenza senza sovrapposizioni di `aa` in `aaa`. Nel caso tale occorrenza non ci sia, la funzione restituisce *s* senza modificarla;
- una funzione `func main()` che legge 4 argomenti della linea di comando, rispettivamente:
  1. una frase (stringa) da elaborare (racchiusa tra `""` se è composta da più parole)
  2. una stringa da cercare
  3. una stringa con cui effettuare la sostituzione
  4. il numero dell'occorrenza da sostituire

  e visualizza su standard output la frase prima della sostituzione e la frase dopo la sostituzione.

3.2 Esempio

Per la seguente invocazione:

```
$ go run sostituzioni.go "nel mio anello c'è dell'oro. nel    tuo no" nel Nl 3
```

l'output risulterà:

```
nel mio anello c'è dell'oro. nel   tuo no
nel mio anello c'è dell'oro. Nl   tuo no
```

Nota. "nel", la sequenza da sostituire, compare una prima volta all'inizio della frase, una seconda nella parola "anello" e una terza volta dopo il punto. Solo la terza occorrenza deve essere sostituita in questo caso, con "Nl".

Per la seguente invocazione:

```
$ go run sostituzioni.go aaa aa bb 2
```

l'output risulterà:

```
aaa
aaa
```

Nota. In questo caso c'è una sola occorrenza senza sovrapposizioni di "aa" in "aaa", quindi non è possibile sostituire la seconda. Se considerassimo invece occorrenze con sovrapposizioni, ne avremmo due: una data dalla prima e seconda 'a', e una data dalla seconda e terza 'a'.

4 Occorrenze

4.1 Descrizione

Scrivere un programma `occorrenze.go` dotato di

- una funzione ricorsiva `occorrenze` che, data una stringa e una runa, determina (in modo ricorsivo) il numero di occorrenze di quella runa nella stringa;
- una funzione `main()` che legge da linea di comando una runa r e una stringa s (in quest'ordine) e visualizza su standard output il numero di occorrenze di r in s, precedute da un messaggio, come nell'esempio sotto.

4.2 Vincoli

Si assuma che gli argomenti sulla linea di comando siano nell'ordine e numero e del tipo richiesti, non occorre fare controlli.

4.3 Esempio

```
$ go run occorrenze.go o laboratorio
```

darà come output:

```
occorrenze di o in laboratorio
3
```

5 Frequenze

5.1 Descrizione

Scrivere un programma `frequenze.go` dotato delle seguenti funzioni:

- `func frequenze(s []string) map[string]int` che data una slice s di stringhe, restituisce una mappa che associa a ogni stringa di s il numero di volte che essa compare nella slice.
- `func main()` che legge da standard input una sequenza di stringhe (separate da caratteri di spaziatura) e produce su standard output la mappa delle frequenze delle stringhe della sequenza.

Nota: per terminare l'input da tastiera, premere invio seguito dalla combinazione di tasti Ctrl D. In caso di dubbi su come gestire la fine dell'input nel programma, consultare la documentazione della funzione `Scan`, funzione che, oltre a salvare i valori letti, restituisce dei valori.

5.2 Esempio

Per il seguente input

```
aa bb cc d aa cc aa
```

l'output di `frequenze.go` sarà:

```
map[d:1 aa:3 bb:1 cc:2]
```

Consegna

Per la consegna, eseguite l'upload dei SINGOLI file sorgente (NON un file archivio!) dalla pagina web: http://upload.di.unimi.it nella sessione che vi è stata indicata.

*** ATTENZIONE!!! ***

NON VERRANNO VALUTATI GLI ESERCIZI CON ERRORI DI COMPILAZIONE O LE CONSEGNE CHE NON RISPETTANO LE SPECIFICHE (ad esempio consegnare un archivio zippato è sbagliato)

Per ritirarsi fare l'upload di un file vuoto di nome `ritirato.txt`.
