Esercizio 1

Scrivere un programma che legga da **riga di comando** una sequenza di numeri decimali di lunghezza arbitraria.

Il programma deve stampare a video, come mostrato nell'**Esempio d'esecuzione**, una stringa di caratteri costituita come segue:

- per ciascun numero maggiore di quello precedente (di almeno un valore epsilon) stampare il carattere `">"`;
- per ciascun numero minore di quello precedente (di almeno un valore epsilon) stampare il carattere `"<"`;
- per ciascun numero uguale a quello precedente (a meno di un valore epsilon) stampare il carattere `"="`;

  Il valore epsilon è l'ultimo numero della sequenza inserita da **riga di comando** e non è da considerarsi ai fini della generazione della stringa di caratteri in output.

Si assuma che i valori letti da **riga di comando** siano specificati nel formato corretto.

Esempio d'esecuzione:

```
$ go run esercizio_1.go 5.4 5.3 5.6 7.0 6.999 0.01
<>>=

$ go run esercizio_1.go 3 4 5 6 2
===

$ go run esercizio_1.go 0.1 0 -0.1 0.01
<<

$ go run esercizio_1.go -0.1 0 0.1 0.01
>>
```

Esercizio 2

Scrivere un programma che:

- legga da **standard input** una stringa `s` costituita da cifre decimali;
- stampi a schermo, dalla più corta alla più lunga, tutte le sottosequenze della stringa `s` nelle quali le cifre sono in ordine decrescente (si considerino solamente sottosequenze di almeno 2 cifre).

Come mostrato nell'**Esempio d'esecuzione**, ciascuna sottosequenza deve essere stampata un'unica volta, riportando il numero di volte in cui la sottosequenza appare in `s`.

Se la stringa `s` letta da **standard input** non è costituita solamente da cifre decimali, il programma termina senza stampare nulla.

Oltre alla funzione `main()`, deve essere definita ed utilizzata almeno la funzione `Sottostringhe(s string) map[string]int`, che riceve in input un valore `string` nel parametro `s`, e restituisce un valore `map[string]int` in cui, per ogni sottosequenza di cifre ordinate (in senso decrescente) presente in `s` di almeno 2 cifre, è memorizzato il numero di volte in cui la sottosequenza appare in `s`.

Esempio d'esecuzione:

```
$ go run esercizio_2.go
123456
output:

$ go run esercizio_2.go
654321
output:
21 1
321 1
4321 1
54321 1
654321 1

$ go run esercizio_2.go
123121212
output:
21 2
31 1

$ go run esercizio_2.go
acc23

$ go run esercizio_2.go
01010101
output:
10 3
```

Esercizio 3

> Un interprete, in informatica e nella programmazione, è un programma in grado di eseguire altri programmi a partire direttamente dal relativo codice sorgente scritto in un linguaggio di alto livello. [da Wikipedia]

Scrivere un programma che simuli un interprete: dato in input il codice di un programma, l'interprete eseguirà le istruzioni in maniera sequenziale.

Parte 1

Scrivere un programma che:

- legge da **standard input** una sequenza di righe di testo;
- termina la lettura quando, premendo la combinazione di tasti `Ctrl+D`, viene inserito da **standard input** l'indicatore End-Of-File (EOF).

Ogni riga di testo letta è un'istruzione nel formato

```
ISTRUZIONE PARAMETRO_1 PARAMETRO_2
```

oppure nel formato

```
ISTRUZIONE PARAMETRO
```

dove `ISTRUZIONE` è il nome dell'operazione da eseguire (una stringa di testo senza spazi), mentre `PARAMETRO`, `PARAMETRO_1` e `PARAMETRO_2` sono dei valori (potenzialmente valori interi o stringhe alfanumeriche senza caratteri di spaziatura) il cui significato dipende dall'operazione specificata da `ISTRUZIONE`.

Definire la struttura `Istruzione` per memorizzare l'operazione specificata da un'istruzione ed i relativi parametri.

Implementare le funzioni:

- `LeggiIstruzioni() (istruzioni []Istruzione)` che:
  i. legge da **standard input** una sequenza di righe di testo, terminando la lettura quando viene letto l'indicatore End-Of-File (EOF);
  ii. restituisce un valore `[]Istruzione` nella variabile `istruzioni` in cui è memorizzata la sequenza di istanze del tipo `Istruzione` inizializzate con i valori letti da **standard input**.

Parte 2

Lo *stato* dell'interprete è definito:

- dalla sequanza di istruzioni che definiscono il programma da eseguire/in esecuzione;
- da un contatore che identifica la prossima istruzione (del programma in esecuzione) che l'interprete deve eseguire;
- dai valori correnti delle variabili del programma in esecuzione (allocate nella memoria dell'interprete).

Definire la struttura `Stato` per memorizzare lo *stato* dell'interprete. Questa struttura deve includere:

- un campo che contenga la sequenza di istanze di tipo `Istruzione` corrispondenti alle righe lette in input nella **Parte 1** dell'esercizio (la sequenza di istruzioni che definisce il programma da eseguire/in esecuzione);
- un campo che serva da contatore numerico per identificare la prossima istruzione che l'interprete deve

eseguire;
- un campo che funzioni da memoria associativa, associando il nome di una variabile del programma in esecuzione (ovvero una stringa alfanumerica senza spazi) ad un valore intero.

Implementare le funzioni:

- `InizializzaStato(programma []Istruzione) (s Stato)` che riceve in input una sequenza di istanze di tipo `Istruzione` nel parametro `programma` (la sequenza di istruzioni che definisce il programma da eseguire/in esecuzione) e restituisce una nuova istanza di tipo `Stato` nella variabile `s`. `s` deve essere inizializzata in modo tale da contenere il programma passato in input (ossia il valore della variabile `programma`), un contatore che identifichi la prima istruzione del programma passato in input, e una memoria associativa vuota.

Parte 3

Le istruzioni che l'interprete deve eseguire possono essere le seguenti:

1. `VAR nomeVariabile valore`: l'interprete alloca nella propria memoria una variabile intera di nome `nomeVariabile` assegnandole il valore `valore`. Se la variabile `nomeVariabile` è già stata definita in istruzioni precedenti, l'interprete termina l'esecuzione del programma stampando a video il messaggio d'errore: `ERRORE - Istruzione 'N': Ridefinizione della variabile 'nomeVariabile'`, dove `N` è il numero della riga corrispondente all'istruzione.

2. `ASS nomeVariabile1 nomeVariabile2`: l'interprete assegna il valore della variabile `nomeVariabile2` alla variabile `nomeVariabile1`.

3. `MUL nomeVariabile1 nomeVariabile2`: l'interprete assegna alla variabile `nomeVariabile1` il valore `nomeVariabile1 * nomeVariabile2`.

4. `SUB nomeVariabile1 nomeVariabile2`: l'interprete assegna alla variabile `nomeVariabile1` il valore `nomeVariabile1 - nomeVariabile2`.

5. `TRI nomeVariabile`: l'interprete assegna alla variabile `nomeVariabile` il valore ottenuto triplicando il valore della variabile `nomeVariabile`.

6. `PRT nomeVariabile`: l'interprete stampa a video il valore della variabile `nomeVariabile`.

7. `JMP nomeVariabile numeroIstruzione`: se il valore della variabile `nomeVariabile` è `0`, l'interprete deve riprendere l'esecuzione dall'istruzione `numeroIstruzione` (la numerazione delle istruzioni parte da 1), altrimenti l'interprete deve proseguire l'esecuzione del programma a partire dall'istruzione successiva a quella corrente. Se il valore della variabile `nomeVariabile` è `0`, l'istruzione ha quindi l'effetto di impostare il valore del contatore numerico (parte della definizione dello *stato* dell'interprete) in modo tale che identifichi l'istruzione di riga pari al valore `numeroIstruzione`, altrimenti il contatore numerico viene incrementato di 1. Se `numeroIstruzione` è superiore al numero delle istruzioni del programma in esecuzione, o inferiore a 1, l'interprete termina l'esecuzione del programma stampando a video il messaggio d'errore: `ERRORE - Istruzione 'N': Riferimento ad un'istruzione inesistente`, dove `N` è il valore del contatore numerico associato all'istruzione corrente.

Al termine della corretta esecuzione di una tra le istruzioni 1-6, il valore del contatore numerico (parte della definizione dello *stato* dell'interprete) viene incrementato di 1.

Se una tra le istruzioni 2-7 fa riferimento ad una variabile che non esiste, ossia una variabile che non è stata allocata nella memoria dell'interprete da istruzioni precedenti, l'interprete termina l'esecuzione del programma stampando a video il messaggio d'errore: `ERRORE - Istruzione 'N': Utilizzo di variabili non allocate`, dove `N` è il valore del contatore numerico associato all'istruzione.

Implementare le funzioni:

- `EseguiIstruzione(s Stato) (Stato, string)` che:
    - riceve in input un'istanza di tipo `Stato` nel parametro `s`, in cui è memorizzato lo *stato* corrente

dell'interprete;
    - esegue l'istruzione identificata dal valore del contatore numerico memorizzato in `s`:
        - se l'esecuzione dell'istruzione non genera errori, la funzione restituisce un'istanza di tipo `Stato` in cui è memorizzato il nuovo *stato* dell'interprete ed il valore `""`;
        - se l'esecuzione dell'istruzione genera errori, la funzione restituisce il valore inalterato di `s` ed un valore di tipo `string` che descrive l'errore che si è verificato (come descritto all'inizio **Parte 3** dell'esercizio);

- `EseguiInterprete(programma []Istruzione)` che riceve in input una sequenza di istanze di tipo `Istruzione` nel parametro `programma` (la sequenza di istruzioni che definisce il programma che l'interprete deve eseguire), inizializza un'istanza di tipo `Stato` ed esegue il corrispondente interprete (utilizzando le funzioni `ÈTerminato` ed `EseguiIstruzione`).

Esempio d'esecuzione:

```
$ cat programma1.txt
VAR base 2
VAR esponente 4
VAR potenza 1
VAR decremento 1
VAR true 0
JMP esponente 10
MUL potenza base
SUB esponente decremento
JMP true 6
PRT potenza

$ go run esercizio_3.go < programma1.txt
Valore variabile potenza: 16

$ cat programma2.txt
VAR a 5
TRI a
PRT a

$ go run esercizio_3.go < programma2.txt
Valore variabile a: 15

$ cat programma3.txt
VAR a 5
VAR b 10
VAR c 15
PRT a
PRT b
PRT c
VAR a

$ go run esercizio_3.go < programma3.txt
Valore variabile a: 5
Valore variabile b: 10
Valore variabile c: 15
ERRORE - Istruzione '7': Ridefinizione della variabile a

$ cat programma4.txt
VAR a 5
VAR b 6
PRT a
PRT b
SUB a b
MUL a a
PRT a
ASS b a
PRT a
PRT b
ASS c b

$ go run esercizio_3.go < programma4.txt
Valore variabile a: 5
```

```
Valore variabile b: 6
Valore variabile a: 1
Valore variabile a: 1
Valore variabile b: 1
ERRORE - Istruzione '11': Utilizzo di variabili non allocate
```