Esercizio 1

Definizione: Un numero naturale è primo se è divisibile solo per se stesso e per 1.

Scrivere un programma che legga da **riga di comando** un numero intero `numero` e stampi tutti i numeri *primi* ottenibili rimuovendo al più `3` cifre consecutive tra quelle che definiscono `numero`.

Ad esempio, se il numero intero letto da **riga di comando** fosse:

`5899`

i numeri ottenibili rimuovendo al più `3` cifre consecutive tra quelle che definiscono `5899` sarebbero:

```
5899
899
599
589
589
99
59
58
9
5
```

Si assuma che il valore specificato a riga di comando sia nel formato corretto e, in particolare, sia un intero maggiore o uguale a 1000.

Oltre alle funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni:

- una funzione `ÈPrimo(n int) bool` che riceve in input un valore (di tipo) `int` nel parametro `n` e restituisce `true` se `n` è primo (i.e., se il valore di `n` rappresenta un numero primo) e `false` altrimenti.

Esempio d'esecuzione:

```
$ go run esercizio_1.go 5899
599
59
5

$ go run esercizio_1.go 12345

$ go run esercizio_1.go 10113
113
1013
1013
113
113
103
101
13
13

$ go run esercizio_1.go 2468
2
```

Esercizio 2

Scrivere un programma che legga da **standard input** tre numeri interi `n`, `L` e `H`, e, come mostrato nell'**Esempio d'esecuzione**, stampi a video una serpetina utilizzando in sequenza i caratteri corrispondenti alle cifre decimali:

- La serpentina è formata da `n` tratti verticali ed `n+1` tratti orizzontali.
- La serpentina inizia e termina con un tratto orizzontale.
- Nella serpentina si alternano un tratto orizzontale e uno verticale.
- Il primo tratto verticale è sul lato destro della serpentina, il secondo tratto verticale è sul lato sinistro della serpentina, il terzo tratto verticale è sul lato destro della serpentina, etc.
- Ciascun tratto orizzontale è largo `L` caratteri.
- Ciascun tratto verticale è alto `H` caratteri.
- La sequenza di caratteri utilizzata per rappresentare la serpentina parte dal carattere `0`. Il carattere successivo al carattere `0` è il carattere `1`. Il carattere successivo al carattere `1` è il carattere `2`, e così via. Il carattere successivo al carattere `9` è il carattere `0`.
- Nel primo tratto orizzontale, ed in tutti quelli al di sotto di un tratto verticale che appare sul lato sinistro della serpentina, i caratteri si susseguono da sinistra verso destra (la sequenza di caratteri cresce da sinistra verso destra).
- In tutti i tratti orizzontali al di sotto di un tratto verticale che appare sul lato destra della serpentina, i caratteri si susseguono da destra verso sinistra (la sequenza di caratteri cresce da destra verso sinistra).

Si assuma che vengano sempre inseriti correttamente da **standard input** tre valori interi positivi (maggiori di 1).

Esempio d'esecuzione:

```
$ go run esercizio_2.go
2 3 4
012
  3
  4
765
8
9
012

$ go run esercizio_2.go
2 4 3
0123
   4
8765
9
0123

$ go run esercizio_2.go
4 3 3
012
  3
654
7
890
  1
432
5
678

$ go run esercizio_2.go
4 2 2
01
32
45
76
```

```
89
```

Esercizio 3

Scrivere un programma che legga da **riga di comando** una stringa di caratteri `operazioni.txt`.

`operazioni.txt` è il nome di un file di testo memorizzato nella stessa directory in cui è memorizzato il programma.

Ogni riga del file `operazioni.txt` è una stringa nel formato

`DATA;TIPO;IMPORTO`

e specifica un'operazione bancaria effettuata su un libretto di risparmio:

1. `DATA`: Una stringa senza spazi che codifica la data in cui è avvenuta l'operazione bancaria, specificata in uno dei seguenti possibili formati:
   i. aaaa_m_g
   ii. aaaa_m_gg
   iii. aaaa_mm_g
   iv. aaaa_mm_gg
2. `TIPO`: Un carattere che specifica la natura dell'operazione bancaria: `'V'` per versamento e `'P'` per prelievo.
3. `IMPORTO`: Un valore intero maggiore di 0 che specifica l'ammontare dell'importo versato o prelevato.

Per ogni giorno in cui è stata effettuata almeno un'operazione bancaria, il programma deve calcolare il saldo giornaliero *limitatamente alle operazioni effettuate nel giorno considerato*, il programma deve cioè calcolare la differenza tra l'ammontare degli importi versati e l'ammontare degli importi prelevati con operazioni effettuate nel giorno considerato.

Infine, il programma deve stampare a video le date associate ai giorni in cui è stata effettuata almeno un'operazione bancaria, riportando per ognuna di esse il saldo giornaliero corrispondente (cfr. **Esempio d'esecuzione**).

Si assuma che:

- ogni riga del file `operazioni.txt` sia nel formato corretto;
- i valori presenti in ogni riga del file `operazioni.txt` specifichino correttamente un'operazione bancaria;
- il file `operazioni.txt` non sia vuoto.

Esempio d'esecuzione:

```
$ cat operazioni_1.txt
2019_06_04;P;34
2020_01_05;V;45
2019_6_04;P;10
2019_6_24;P;23
2019_06_04;P;456
2019_06_4;V;26
2019_06_8;P;74
2020_1_05;V;178
2020_01_7;V;194
2020_02_05;V;56

$ go run esercizio_3.go operazioni_1.txt
2019/6/8 -> -74
2020/1/7 -> 194
2020/2/5 -> 56
2019/6/4 -> -474
2020/1/5 -> 223
2019/6/24 -> -23

$ cat operazioni_2.txt
2019_10_2;V;130
2019_10_3;P;10
```

```
2019_11_01;V;560
2019_12_01;P;126
2019_10_02;P;30
2019_10_03;V;110
2019_11_1;P;560
2019_12_1;V;120

$ go run esercizio_3.go operazioni_2.txt
2019/10/2 -> 100
2019/10/3 -> 100
2019/11/1 -> 0
2019/12/1 -> -6
```

Esercizio 4

Sul piano cartesiano, ad ogni punto individuato da una coppia di numeri reali, chiamati rispettivamente ascissa e ordinata, può essere associata un'etichetta simbolica, generalmente una lettera maiuscola.

Scrivere un programma che:

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

Ogni riga del testo è una stringa nel formato:

```
etichetta;x;y
```

La tripla di valori separati dal carattere `;` specifica un punto sul piano cartesiano:

1. *etichetta*: una stringa che specifica l'etichetta simbolica associata al punto (ad es.: "A", "B", ...)
2. *x*: un valore reale che specifica l'ascissa del punto;
3. *y*: un valore reale che specifica l'ordinata del punto.

Ogni tripla di punti definisce un triangolo che ha per vertici i punti stessi.

Si ipotizzi che vengano inserite da **standard input** le seguenti di righe di testo:

```
A;10.0;2.0
B;11.5;3.0
C;8.0;1.0
D;14.0;-1.0
```

La terna di punti:

```
A;10.0;2.0
B;11.5;3.0
C;8.0;1.0
```

definisce il triangolo `ABC`.

La terna di punti:

```
A;10.0;2.0
B;11.5;3.0
D;14.0;-1.0
```

definisce il triangolo `ABD`.

La terna di punti:

```
A;10.0;2.0
C;8.0;1.0
D;14.0;-1.0
```

definisce il triangolo `ACD`.

La terna di punti:

```
B;11.5;3.0
C;8.0;1.0
```

```
D;14.0;-1.0
```

definisce il triangolo `BCD`.

La lunghezza di ciascun lato di un triangolo è pari alla distanza euclidea tra gli estremi del lato.

Per esempio, la lunghezza del lato `AB` del triangolo `ABD` è pari alla distanza euclidea tra i punti `A` e `B`: $((x_A-x_B)^2 + (y_A-y_B)^2)^{1/2}$.

Una volta terminata la fase di lettura, il programma deve stampare a video (come mostrato nell'**Esempio di esecuzione**) la descrizione del triangolo *rettangolo* con *area maggiore* tra tutti quelli definibili a partire dai punti specificati nelle righe di testo lette da **standard input**, e tale che:

1. uno dei due cateti del triangolo rettangolo sia parallelo all'asse delle ascisse (l'altro cateto deve essere quindi parallelo all'asse delle ordinate).

Se non esistono triangoli rettangoli che soddisfano la condizione 1, il programma non deve stampare nulla.

Si assuma che:

- le righe di testo lette da **standard input** siano nel formato corretto;
- la tripla di valori presente in ogni riga specifichi correttamente un punto sul piano cartesiano;
- vengano lette da **standard input** almeno 3 righe di testo che specificano 3 punti distinti sul piano cartesiano.

Oltre alla funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni:

- una funzione `Distanza(p1, p2 Punto) float64` che riceve in input due instanze del tipo `Punto` nei parametri `p1` e `p2` e restituisce un valore `float64` pari alla distanza euclidea tra i punti rappresentati da `p1` e `p2`;
- una funzione `StringPunto(p Punto) string` che riceve in input un'instanza del tipo `Punto` nel parametro `p` e restituisce un valore `string` che corrisponde alla rappresentazione `string` di `p` nel formato `ETICHETTA = (X, Y)`, dove `ETICHETTA` è il valore `string` che specifica l'etichetta simbolica di `p`, mentre `X` ed `Y` sono i valori `float64` che specificano rispettivamente l'ascissa e l'ordinata di `p`;
- una funzione `StringTriangoloRettangolo(t TriangoloRettangolo) string` che riceve in input un'instanza del tipo `TriangoloRettangolo` nel parametro `t` e restituisce un valore `string` che corrisponde alla rappresentazione `string` di `t` nel formato `Triangolo rettangolo con vertici VERTICE_1, VERTICE_2 e VERTICE_3, ed area AREA.`, dove `VERTICE_1`, `VERTICE_2` e `VERTICE_3` sono le rappresentazioni `string` delle istanze del tipo `Punto` che rappresentano i vertici di `t`, mentre `AREA` è il valore `float64` che specifica l'area di `t`.

Esempio d'esecuzione:

```
$ go run esercizio_4.go
A;4;8
B;4;2
C;7;2
D;5;2
E;4;9
F;5;9
Triangolo rettangolo con vertici B = (4, 2), C = (7, 2) e E = (4, 9), ed area 10.5.

$ go run esercizio_4.go
A;4;8
B;3;9
C;8;7
D;2;4
E;1;0

$ go run esercizio_4.go
A;4;5
B;4;1
C;4;3
```

```
D;7;4
```