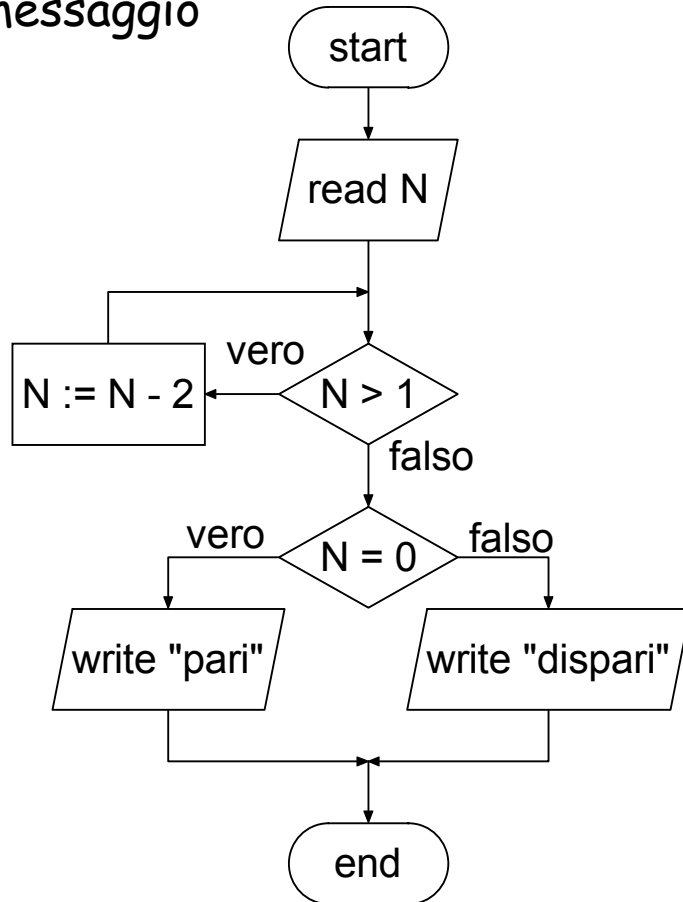


# Esercizio: dispari-pari

Dato un numero, verificare se è pari o dispari e stampare il relativo messaggio



**start**

read N

**while** (N > 1)

    N := N - 2

**end while**

**if** (N = 0) **then**

    write "pari"

**else**

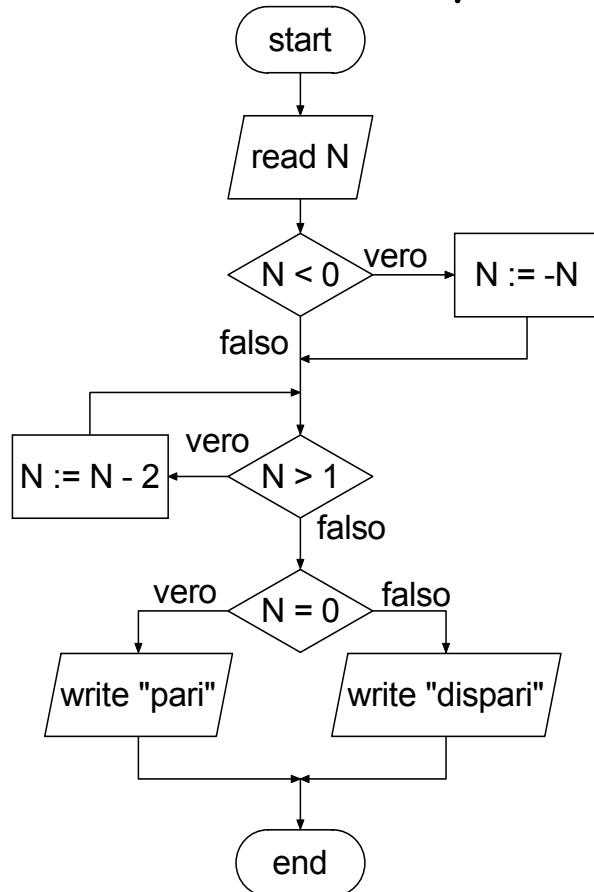
    write "dispari"

**end if**

**end**

# Esercizio: dispari-pari (controllo input)

Dato un numero stampare se è pari o dispari



**start**

read N

**if** (N < 0) **then**

    N := -N

**end if**

**while** (N > 1)

    N := N - 2

**end while**

**if** (N = 0) **then**

    write "pari"

**else**

    write "dispari"

**end if**

**end**

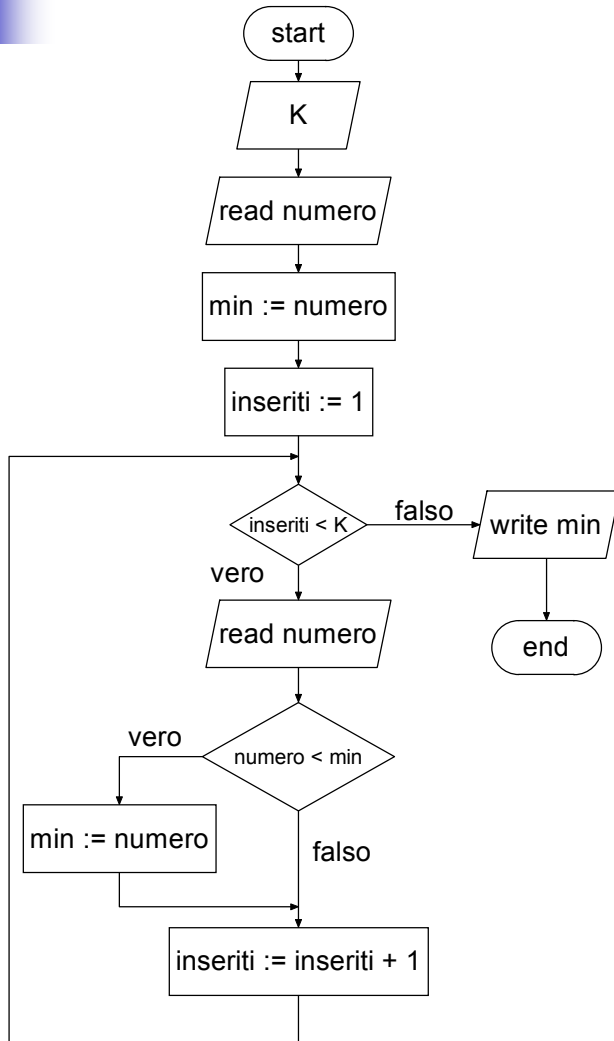


## Esercizio: minimo di una sequenza di $K$ numeri

---

- Si supponga di fornire in input ad un programma un numero  $K$  e  $K$  interi positivi. Il programma deve restituire il valore minimo tra quelli introdotti.

# Esercizio: minimo di una sequenza di K numeri



**start**

read K

read numero

min := numero

inseriti := 1

**while** (inseriti < K)

read numero

**if** (numero < min) **then**

min := numero

**end if**

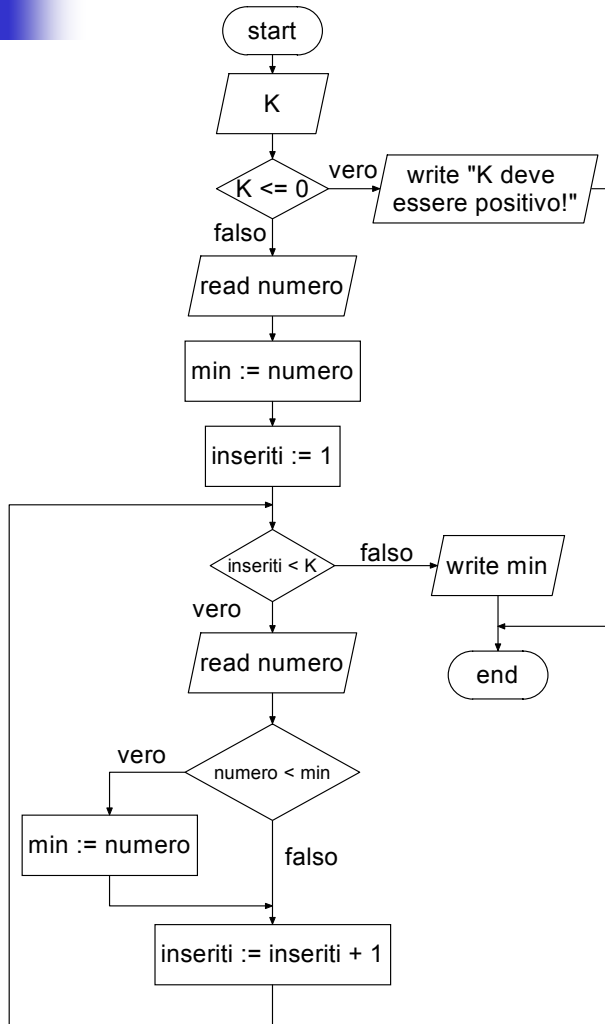
inseriti := inseriti + 1

**end while**

write min

**end**

# Esercizio: minimo di una sequenza di K numeri (controllo input)



**start**

read K

**if** ( K <= 0) **then**

write "K deve essere positivo!"

**else**

read numero

min := numero

inseriti := 1

**while** (inseriti < K)

read numero

**if** (numero < min) **then**

min := numero

**end if**

inseriti := inseriti + 1

**end while**

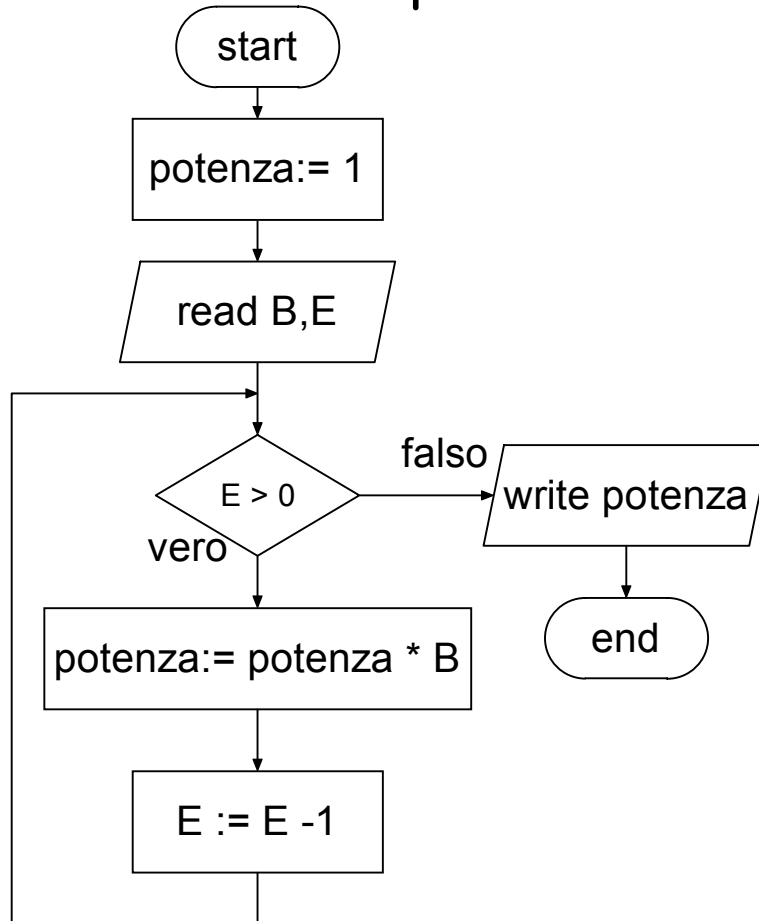
write min

**end if**

**end**

# Esercizio: elevamento a potenza

Data la base e l'esponente calcolare l'elevamento a potenza



**start**

potenza := 1

read B, E

**while** (E > 0)

potenza := potenza \* B

E := E - 1

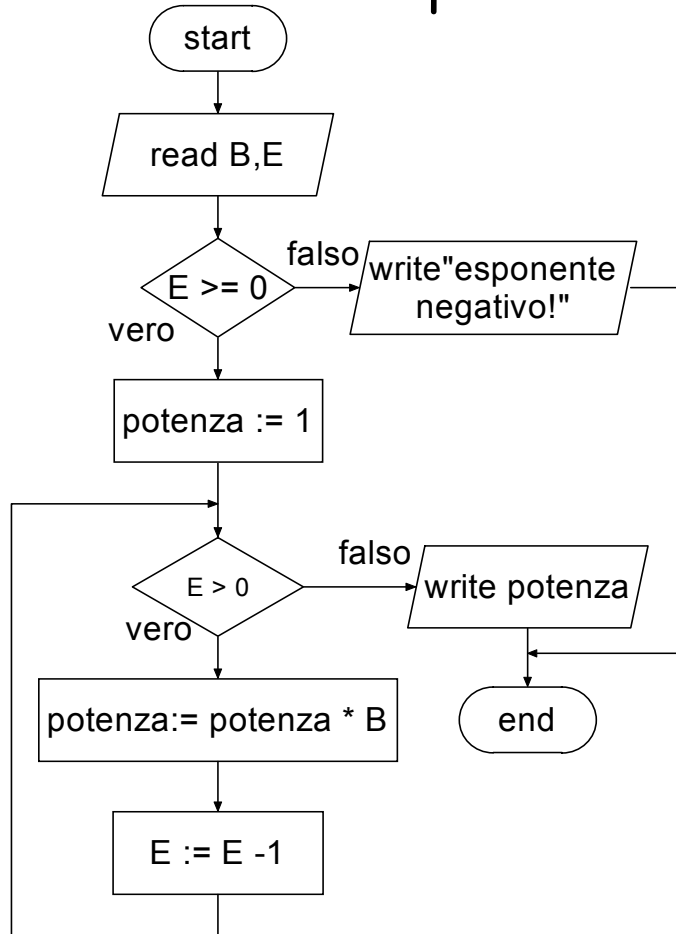
**end while**

write potenza

**end**

# Esercizio: elevamento a potenza (controllo input)

Data la base e l'esponente calcolare l'elevamento a potenza



**start**

read B,E

**if** (E >= 0) **then**

potenza := 1

**while** (E > 0)

potenza := potenza \* B

E := E - 1

**end while**

write potenza

**else**

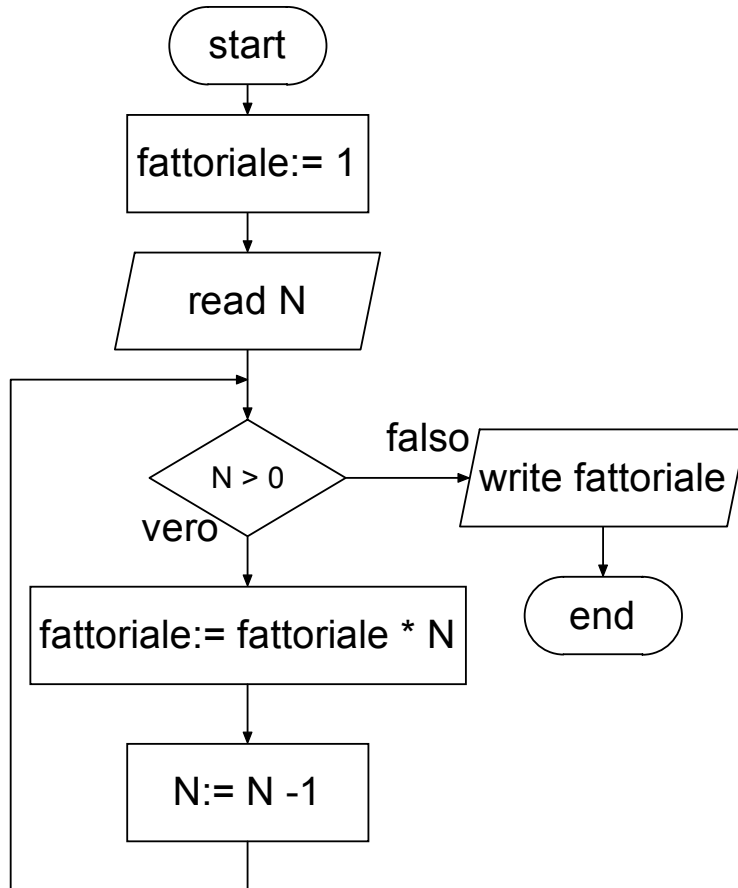
write "esponente negativo!"

**end if**

**end**

# Esercizio: fattoriale

Dato un numero calcolare il suo fattoriale



**start**

fattoriale := 1

read N

**while** (N > 0)

fattoriale := fattoriale \* N

N := N - 1

**end while**

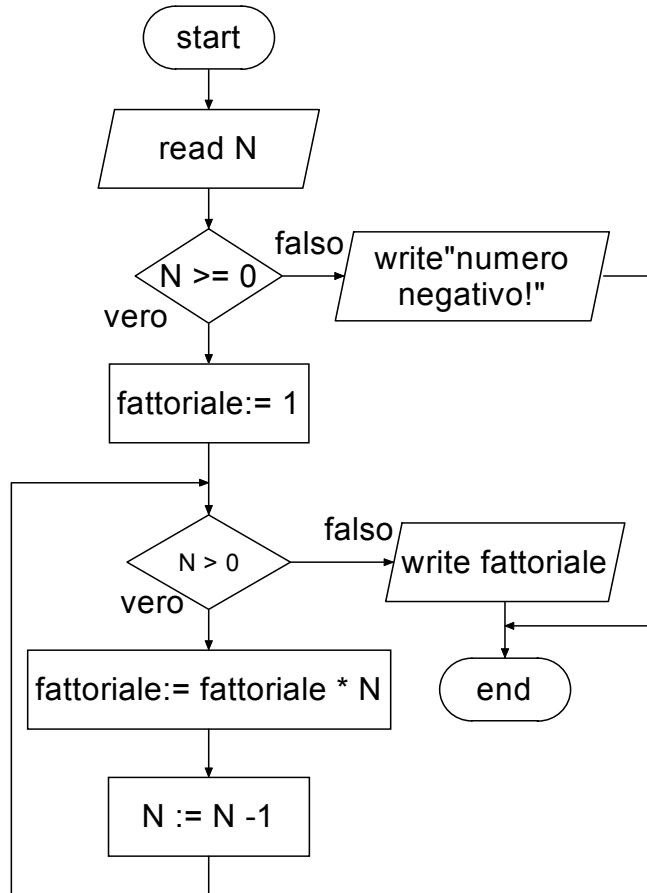
write fattoriale

**end**



# Esercizio: fattoriale (controllo input)

Dato un numero calcolare il suo fattoriale



**start**

read N

**if** (N >= 0) **then**

    fattoriale := 1

**while** (N > 0)

        fattoriale := fattoriale \* N

        N := N - 1

**end while**

    write fattoriale

**else**

    write "numero negativo!"

**end if**

**end**

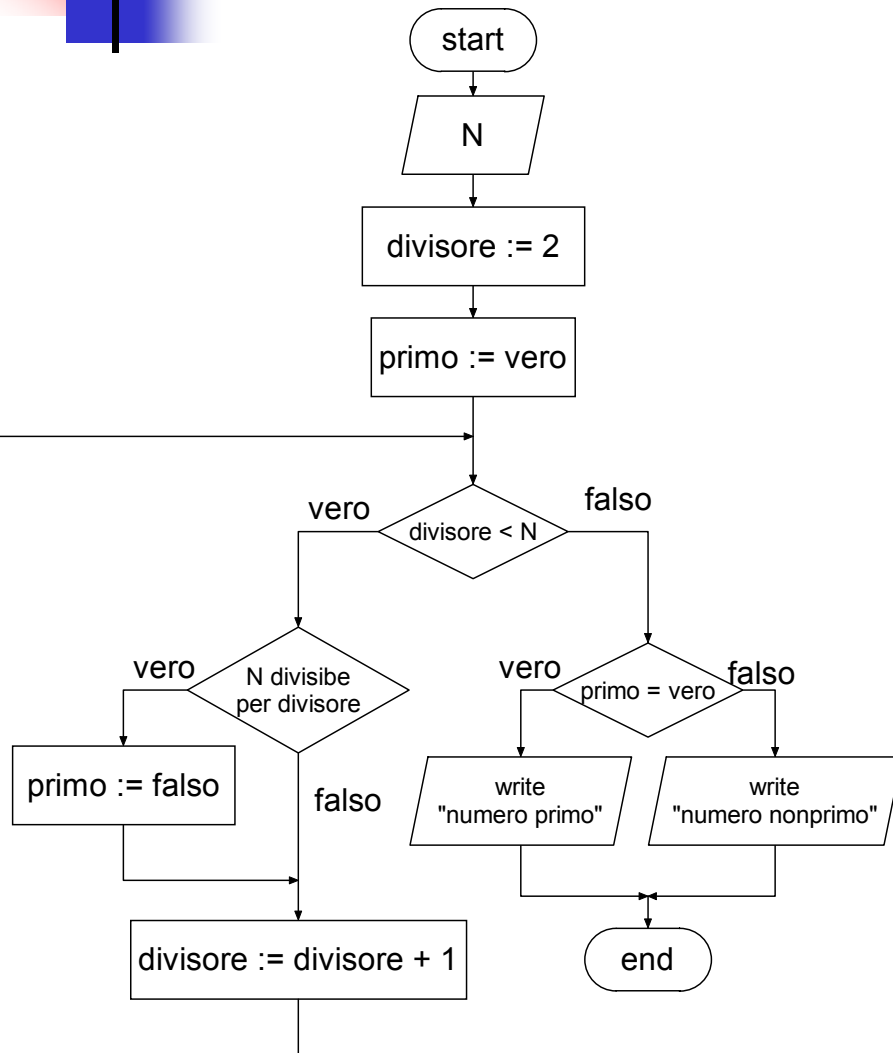


## Esercizio: numero primo

---

- Dato un numero  $N$  scrivere un algoritmo che verifichi se  $N$  è un numero primo e stampi un relativo messaggio
- Il numero  $N$  è un numero primo se è divisibile solo per 1 e per  $N$
- Quindi, per verificare se un numero  $N$  è primo è sufficiente provare a dividerlo per tutti gli interi minori di esso
  - Se almeno uno di questi interi è un divisore di  $n$  allora  $n$  non è primo
  - Altrimenti  $n$  è primo

# Esercizio: numero primo



**start**

read N

divisore := 2

primo := vero

**while** (divisore < N)

**if** (N divisibile per divisore) **then**  
        primo := falso

**end if**

    divisore := divisore + 1

**end while**

**if** (primo = vero) **then**

    write "numero primo"

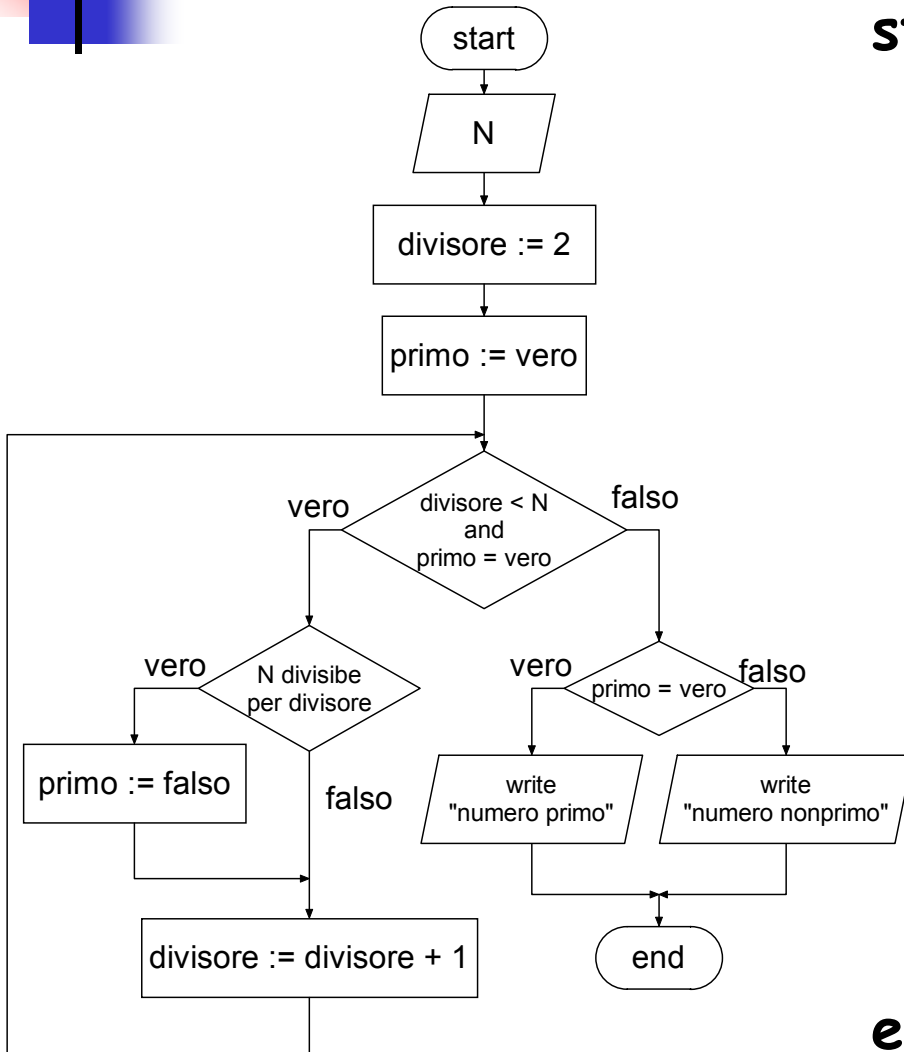
**else**

    write "numero non primo"

**end if**

**end**

# Esercizio: numero primo - ottimizzazione I



**start**

read N

divisore := 2

primo := vero

**while** (divisore < N and primo = vero)

**if** (N è divisibile per divisore) **then**  
        primo := falso

**end if**

    divisore := divisore + 1

**end while**

**if** (primo = vero) **then**

    write "numero primo"

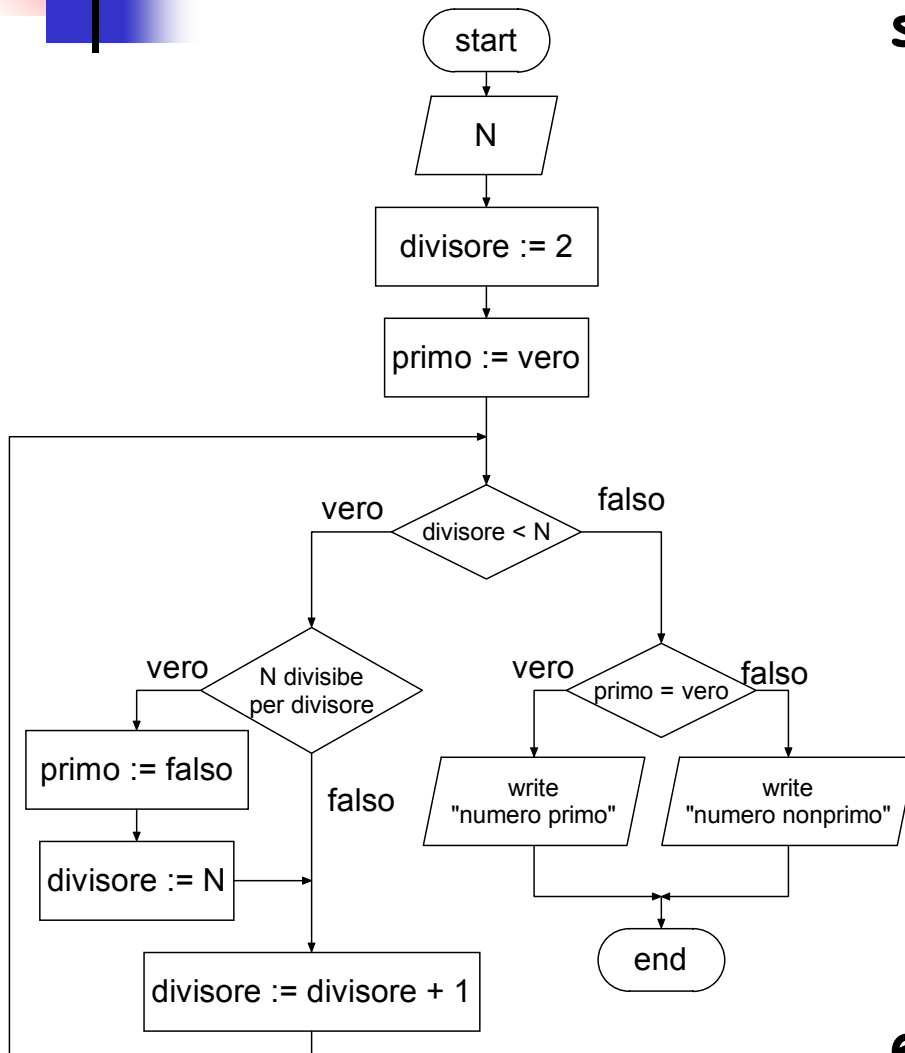
**else**

    write "numero non primo"

**end if**

**end**

# Esercizio: numero primo - ottimizzazione II



**start**

read N

divisore := 2

primo := vero

**while** (divisore < N)

**if** (N è divisibile per divisore) **then**

        primo := falso

        divisore := N

**end if**

    divisore := divisore + 1

**end while**

**if** (primo = vero) **then**

    write "numero primo"

**else**

    write "numero non primo"

**end if**

**end**



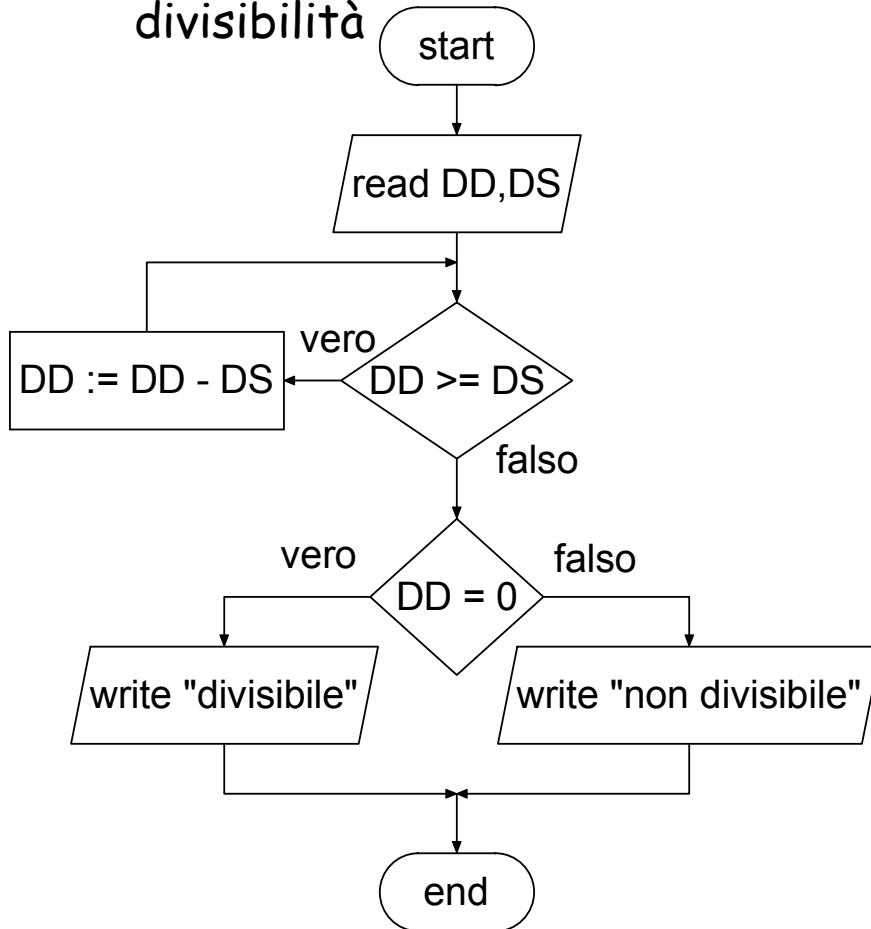
# Esercizi

---

- Produrre un algoritmo che controlla la correttezza dell'input
- Produrre un algoritmo più efficiente di quello di base (più efficiente vuol dire che compie meno operazioni)

# Esercizio: divisibilità

Dati un dividendo ed un divisore scrivere un algoritmo che verifichi la divisibilità



**start**

read DD,DS

**while** (DD >= DS)

DD := DD - DS

**end while**

**if** (DD = 0) **then**

write "divisibile"

**else**

write "non divisibile"

**end if**

**end**



# Esercizi

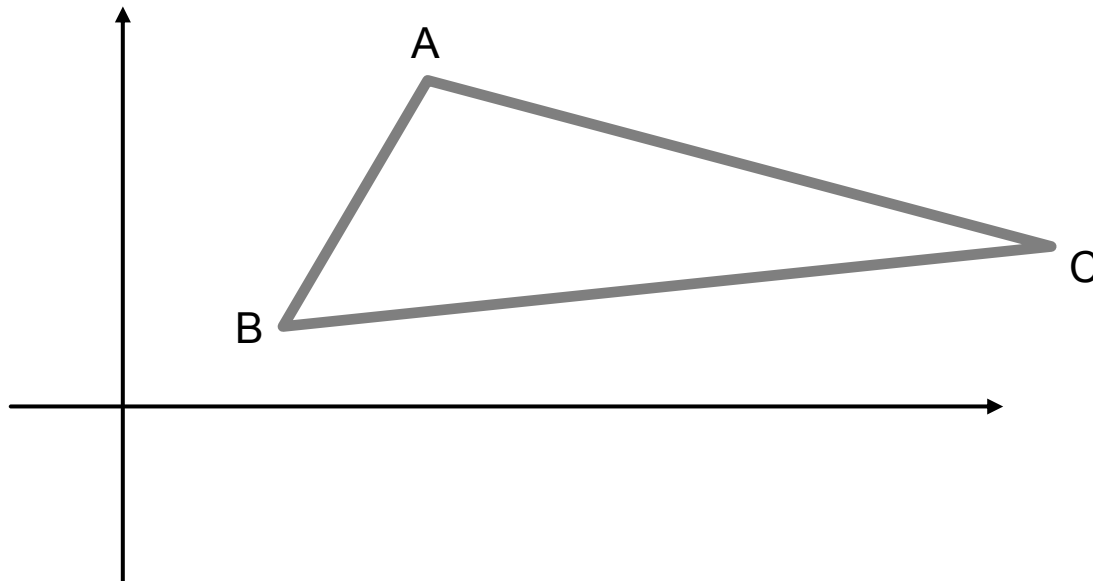
---

- Produrre un algoritmo che sia corretto per ogni tipologia di dati in ingresso
- Come risolvereste il problema del pari o dispari adesso?

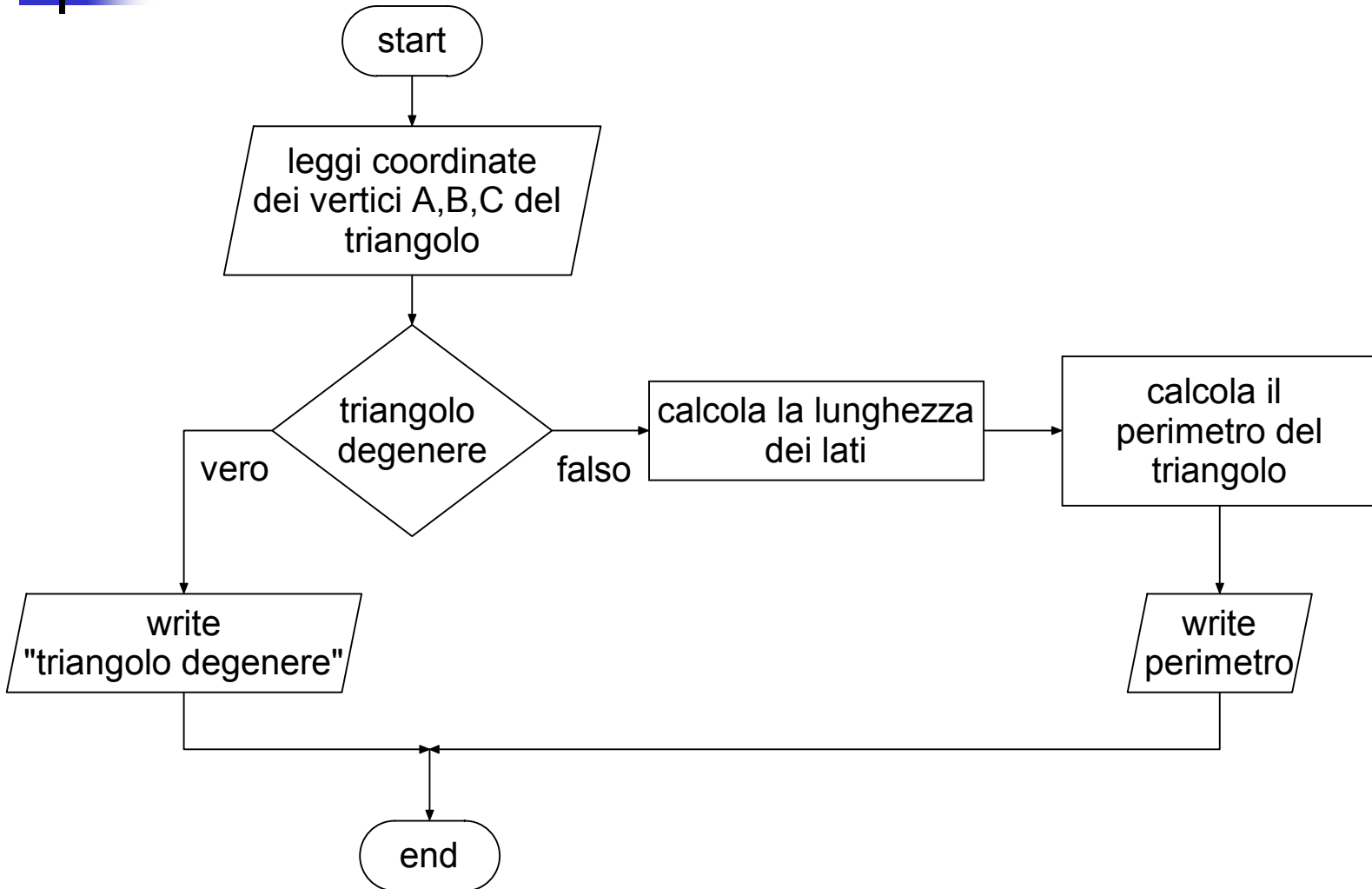


## Esercizio: triangoli

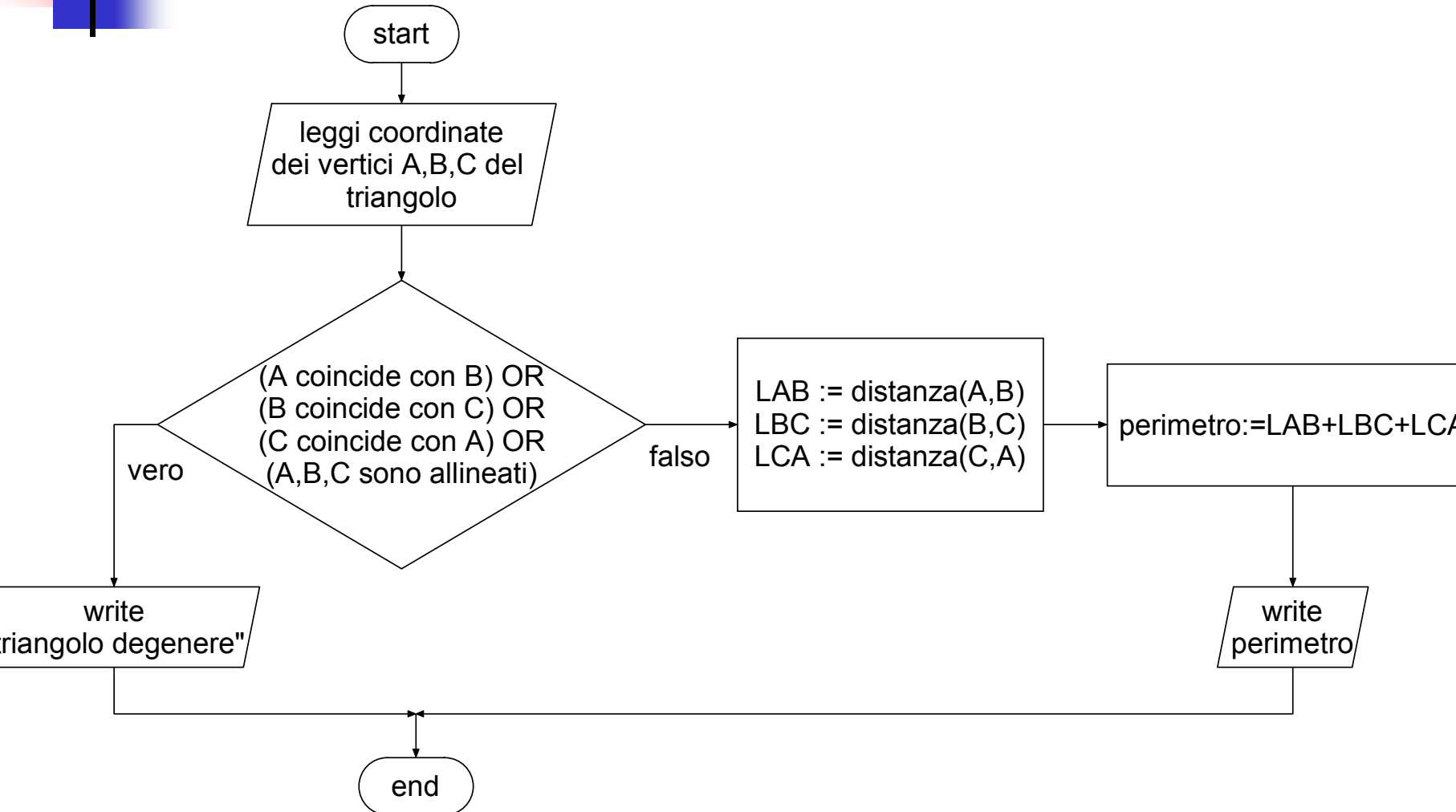
- Scrivere un algoritmo che, date le coordinate di tre punti corrispondenti ai vertici di un triangolo, riconosca se si tratta di un triangolo degenere o no, e nel caso di triangolo non degenere calcoli il suo perimetro



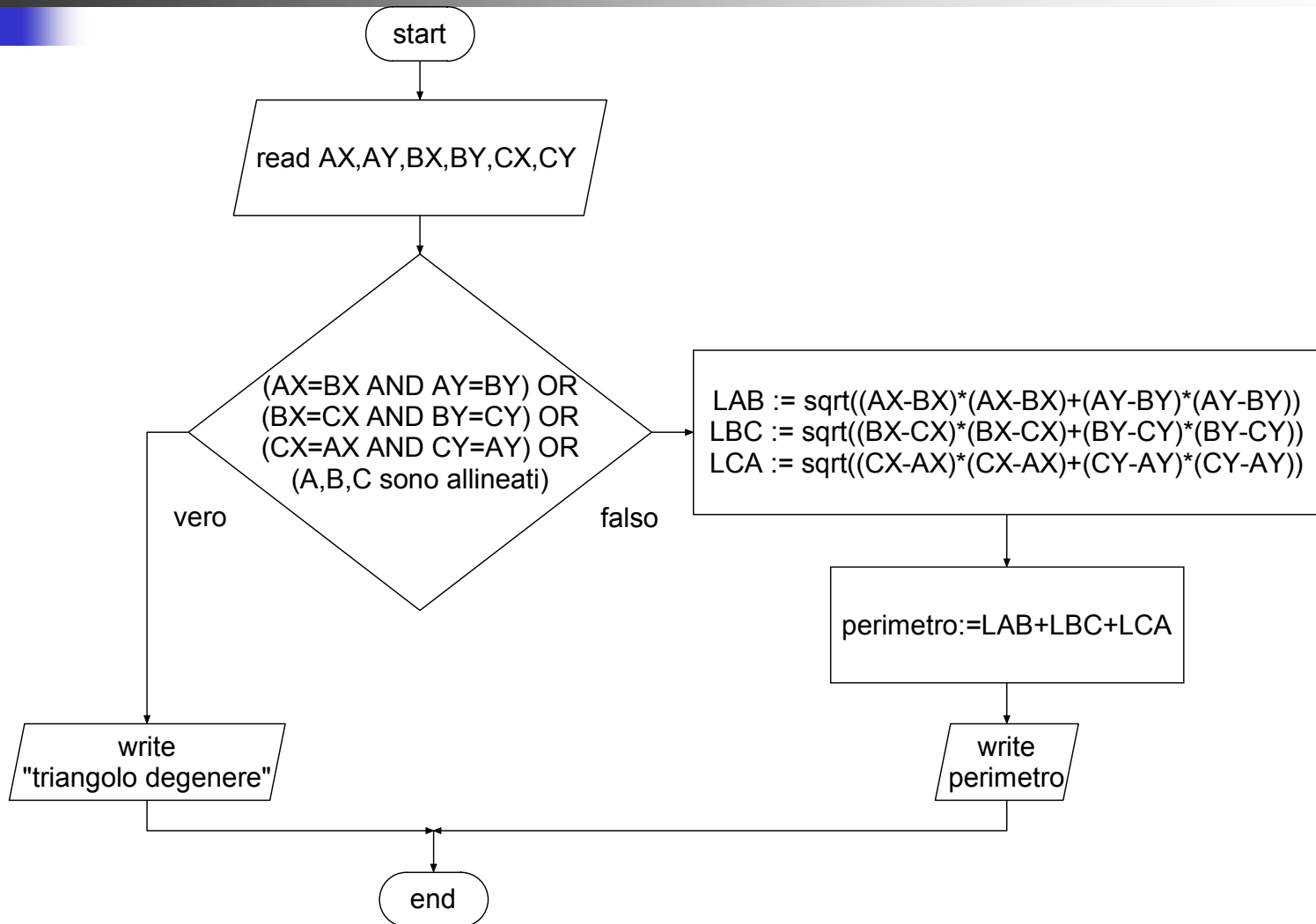
# Esercizio: triangoli - soluzione preliminare



# Esercizio: triangoli - raffinamento

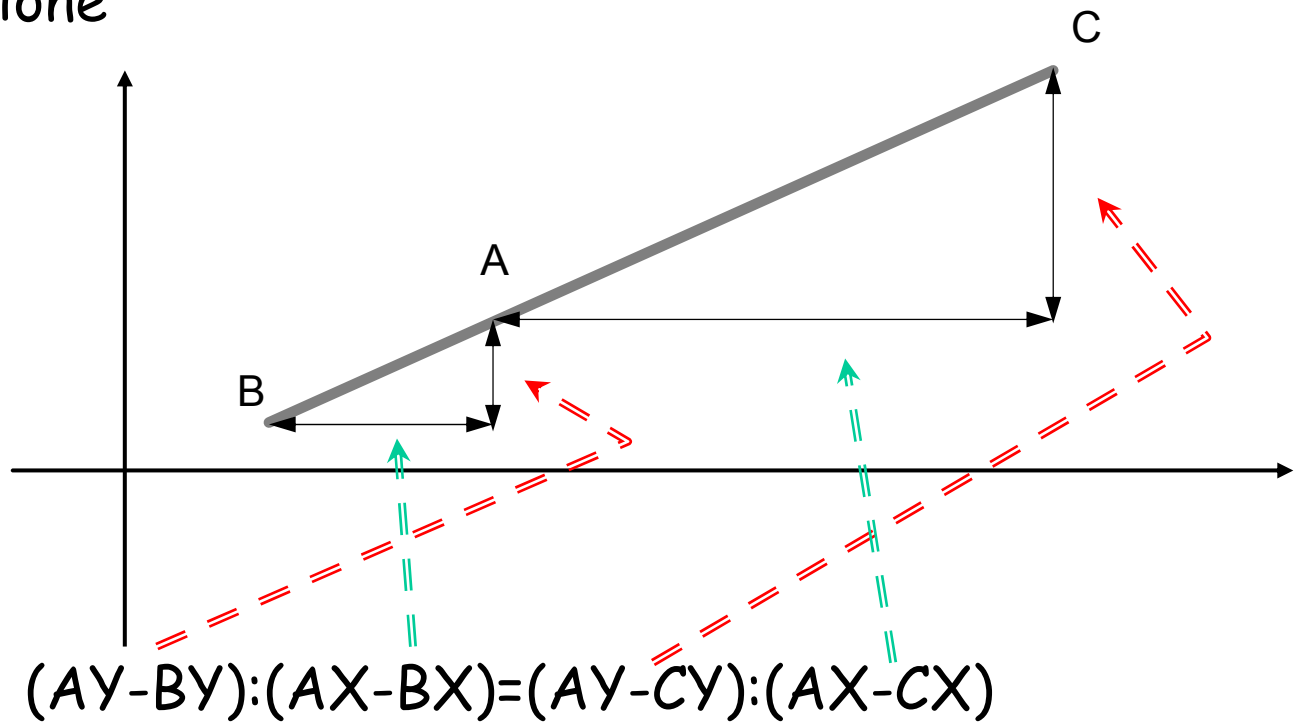


# Esercizio: triangoli - raffinamento ulteriore



# Esercizio: triangoli

- Se i tre vertici sono allineati allora otteniamo due triangoli rettangoli i cui cateti sono nella stessa proporzione





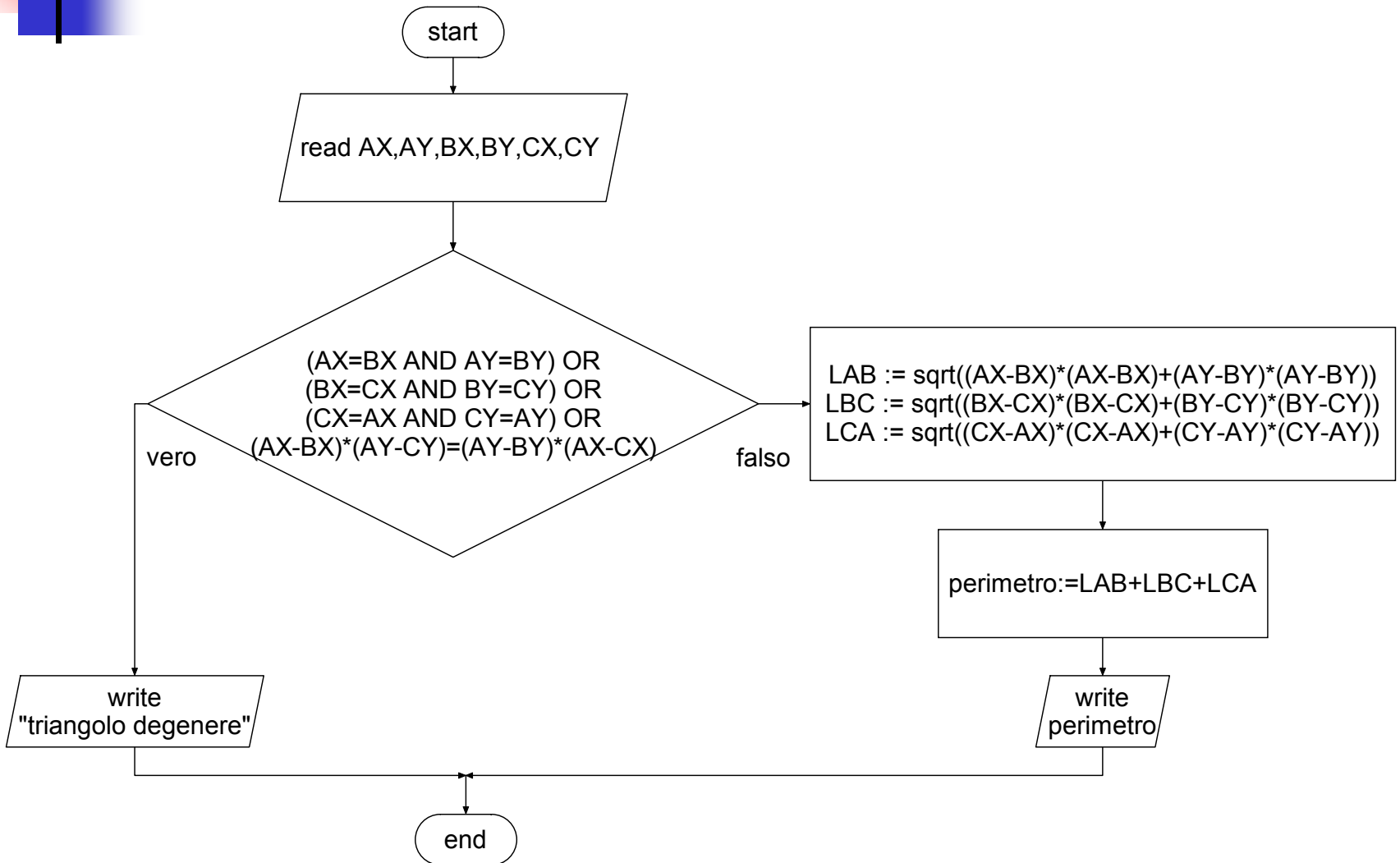
## Esercizi: triangoli

---

- In una proporzione il prodotto dei medi è uguale al prodotto degli estremi per cui i tre vertici sono allineati se è vera la condizione logica

$$(AX-BX)*(AY-CY)=(AY-BY)*(AX-CX)$$

# Esercizi: triangoli - soluzione





# Individuazione di sottoproblemi

---

- Quando il problema è complesso conviene partire con una individuazione di sottoproblemi
- Scriviamo un algoritmo contenente azioni o condizioni complesse per l'esecutore che dettaglieremo e raffineremo in passaggi successivi per ottenere un algoritmo direttamente eseguibile
- Ognuno dei sottoproblemi potrà essere risolto da un algoritmo a parte che potremo riutilizzare, quando sarà necessario, nella soluzione di ulteriori problemi complessi.





## Individuazione di sottoproblemi: vantaggi

---

- I dettagli delle diverse soluzioni sono descritti negli algoritmi dei sottoproblemi
- In generale, uno stesso sottoproblema deve essere risolto più volte nella soluzione di un problema principale
- Dagli algoritmi derivano programmi, quindi si possono raccogliere librerie di software da riutilizzare in nuovi programmi



## Esercizio: frazioni

---

- Scrivere un algoritmo che verifichi se una frazione è apparente o propria
- Sapreste risolverlo senza un'analisi del problema?
- Vi ricordate la "FASE 1: Dare un nome al problema partendo dall'analisi del problema" e la "FASE 2: Scrivere la specifica funzionale"?
  - apparenti: numeratore multiplo di denominatore
  - proprie: numeratore minore di denominatore



## Esercizio: MCD

---

- Scrivere un algoritmo che calcoli il massimo comune divisore di due numeri
- Sapreste risolverlo senza un'analisi del problema?
- Vi ricordate la "FASE 1: Dare un nome al problema partendo dall'analisi del problema" e la "FASE 2: Scrivere la specifica funzionale"?
- Il massimo comune divisore di due numeri è il più grande numero, minore o uguale del più piccolo dei due, che divide entrambi



## Esercizio: anno bisestile

---

- Scrivere un algoritmo che verifichi se un anno è bisestile producendo un messaggio
- Sapreste risolverlo senza un'analisi del problema?
- Vi ricordate la "FASE 1: Dare un nome al problema partendo dall'analisi del problema" e la "FASE 2: Scrivere la specifica funzionale"?
- Un anno è bisestile (ha 366 giorni) se è divisibile per quattro (come il 1980) e non è divisibile per 100 (ad es. il 1900 non è bisestile). Fanno eccezione gli anni divisibili per 400, che sono bisestili (ad es. il 2000 è bisestile).
- Questa regola non si applica prima del 1582, anno di introduzione del calendario gregoriano.



## Esercizio: busta paga

---

- Scrivete un algoritmo che calcoli l'importo della busta paga settimanale di un lavoratore sapendo il numero di ore che ha lavorato durante una settimana e la retribuzione oraria
- L'algoritmo deve segnalare l'opportunità di far recuperare al lavoratore delle ore di lavoro se non è stato rispettato l'accordo sindacale che prevede un minimo di 35 ore settimanali
- L'algoritmo deve altresì tenere in conto le ore di straordinario che sono, come da contratto, retribuite il doppio di quelle normali



## Esercizio: poligoni

---

- Scrivere un algoritmo che, date le coordinate di quattro punti corrispondenti ai vertici di un poligono irregolare, riconosca se si tratta di un quadrato o di un rettangolo e nel caso calcoli la sua area



# Problema: calcolare il massimo tra K numeri

---

- Scrivere un algoritmo che fornisca in input ad un programma un numero K e K interi positivi. L'algoritmo deve restituire il valore massimo tra quelli introdotti e stampare in ordine inverso i numeri inseriti.
- Sembra un problema già visto:
  - Si supponga di fornire in input ad un programma un numero K e K interi positivi. Il programma deve restituire il valore minimo tra quelli introdotti.
- ma non lo è!! Infatti dobbiamo memorizzare tutti i valori inseriti per stampare il primo inserito per ultimo
- Ma quante variabili usiamo?