POLITECNICO DI MILANO
Applicazione SchedaIncarico

Scheda Riassuntiva

Anno Accademico	2005/06
Facoltà	Facolta' di Ingegneria Industriale
Tipo Insegnamento	MONODISCIPLINARE
Codice Identificativo	060065
Denominazione Insegnamento	INFORMATICA C
Docente	MARTUCCI RENATO
CFU	5.0

Corsi di Studio cui l'insegnamento è offerto

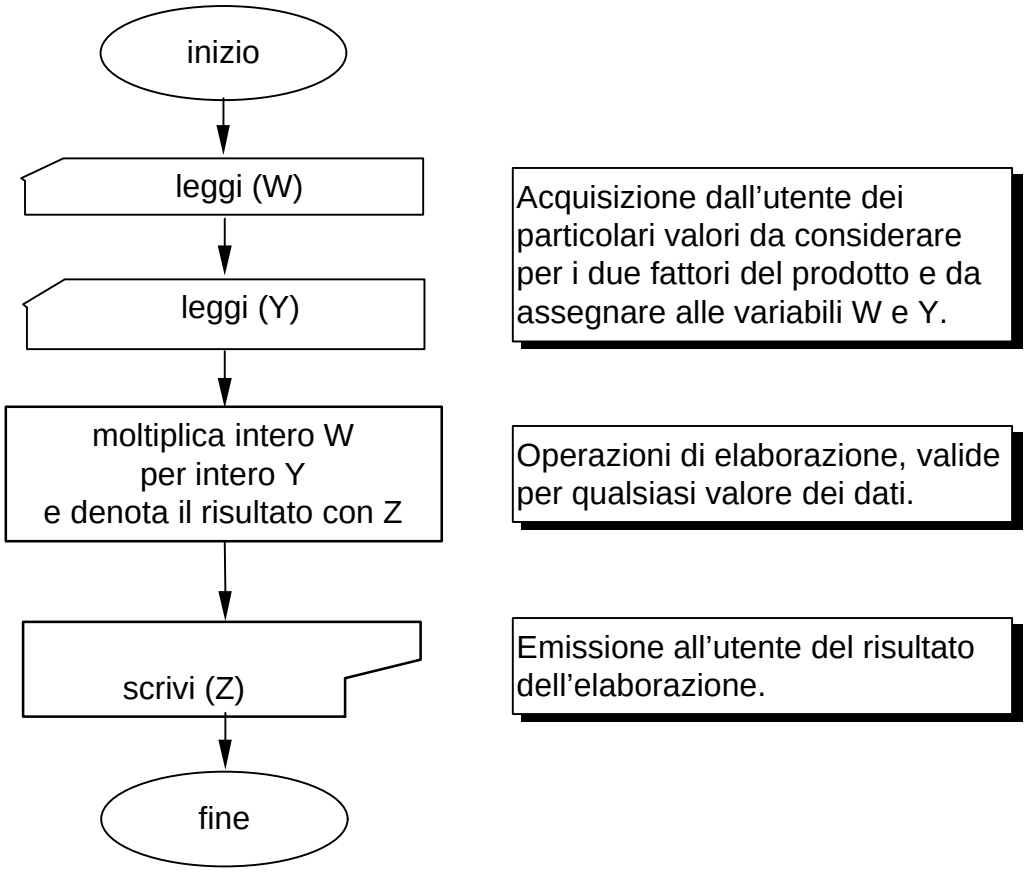
Nome Corso di Laurea	Corso Unione	Indirizzo	DA	A
Ing.IV(1 liv.) - BV (100) INGEGNERIA AEROSPAZIALE	-	*	N	ZZZZ

- 1^ parte
  - Programmare Gli Algoritmi
  - _____
  - _____

http://www.elet.polimi.it/upload/martucci/index.html

Esempio 1: prodotto di due numeri interi positivi

diagramma di flusso con direttive «in italiano»



variabili:

W, Y intere (rappresentano i fattori di ingresso)

variabile:

Z intera (rappresenta il risultato in uscita)

Operatori e espressioni

Le operazioni da eseguire possono essere descritte tramite simboli che costituiscono gli **operatori** del formalismo descrittivo, o con **nomi di funzioni**.

***Operatori aritmetici*:** +, -, ∗, / .....
Gli operatori aritmetici «agiscono» su valori, detti operandi, che possono essere rappresentati da variabili oppure essere valori costanti, e producono un valore numerico.

Operatori di confronto: >, =, <, ...
Gli operatori di confronto «agiscono» su valori, detti operandi, che possono essere rappresentati da variabili oppure essere valori costanti, e producono un valore logico **vero** o **falso**.

Funzioni: cos (X), ...
Le funzioni «agiscono» su valori, detti parametri, che possono essere rappresentati da variabili oppure essere valori costanti, e producono un valore.

Procedure: Leggi (X), Scrivi (N), ...
Le procedure «agiscono» su valori, detti parametri, ed effettuano delle operazioni.

Espressione: rappresenta una composizione di operatori, funzioni, variabili e valori costanti e ad essa è associato un valore

Operazione di assegnamento: consente di assegnare un nuovo valore alla variabile a sinistra dell'operatore di assegnamento (**:=**). Tale valore è determinato dalla valutazione dell'espressione a destra

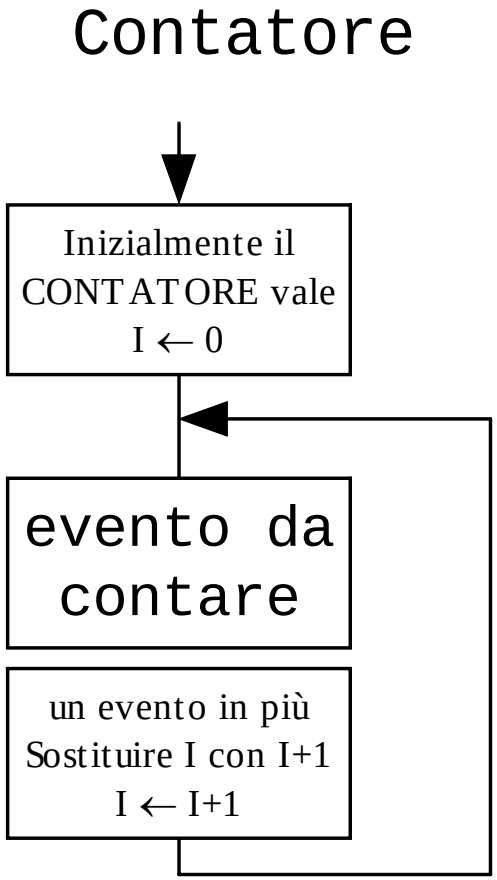
L'operazione di assegnamento costituisce il modo tipico per assegnare ad una variabile il valore risultante da elaborazioni. L'altro tipo di operazione che assegna un nuovo valore ad una variabile è

Programmare per Paradigmi

- Esistono alcuni schemi elementari e ricorrenti che possono essere usati per comporre casi più complessi
  - Contatore di eventi
  - Esecuzione di N elaborazioni elementari
  - “Accumulatore” – somma di N quantità
  - Accumulare sino al superamento di un MAX
  - Lettura di dati dall’ ”input”
  - …

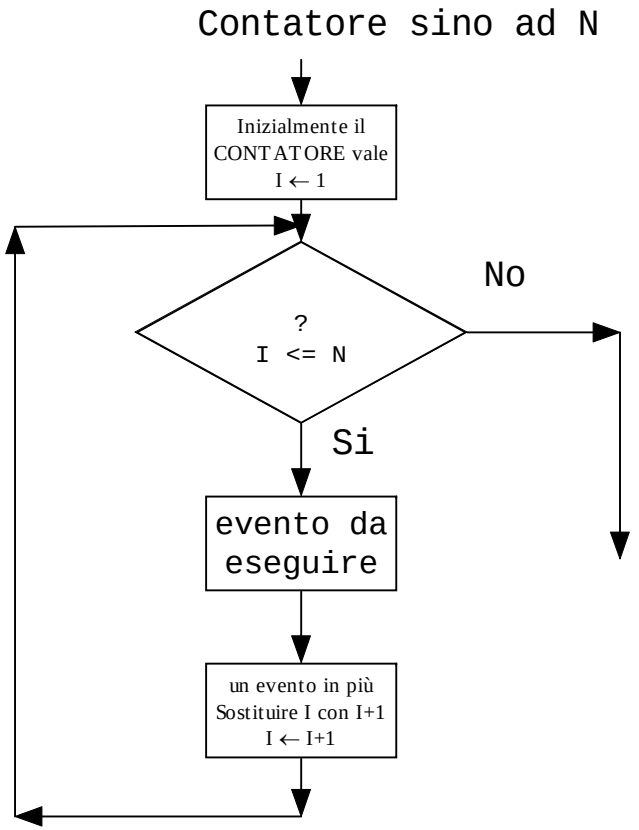
P1. Contatore di eventi

- Per contare qualcosa si deve
  - Stabilire un contatore (le mani, …. I)
  - Inizializzare a 0
  - Incrementare ogni volta che capita l'Evento



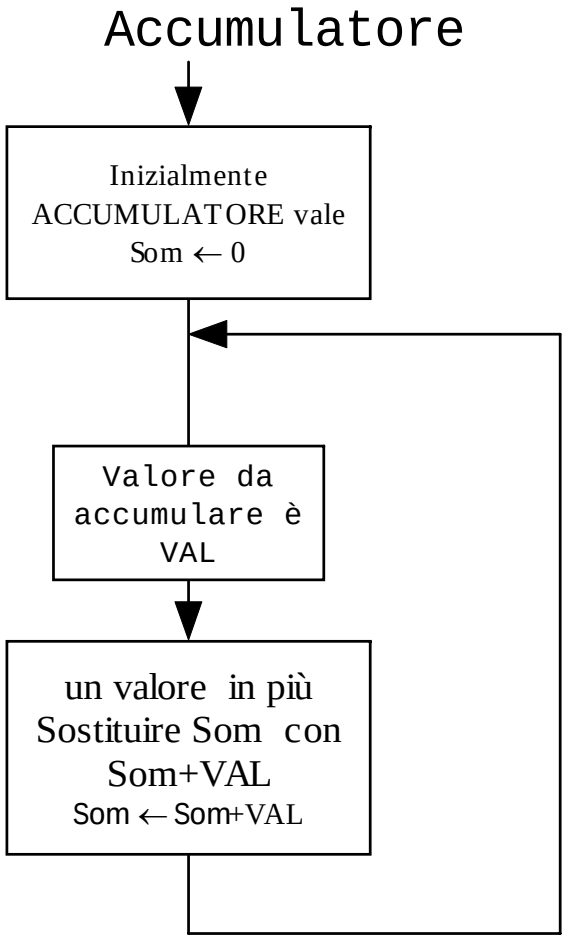
P2. Esecuzione di N elaborazioni elementari

- Si deve Contare sino ad N
- Si deve quindi
  - Contare (vedi precedente)
  - Chiedersi se si è superato N



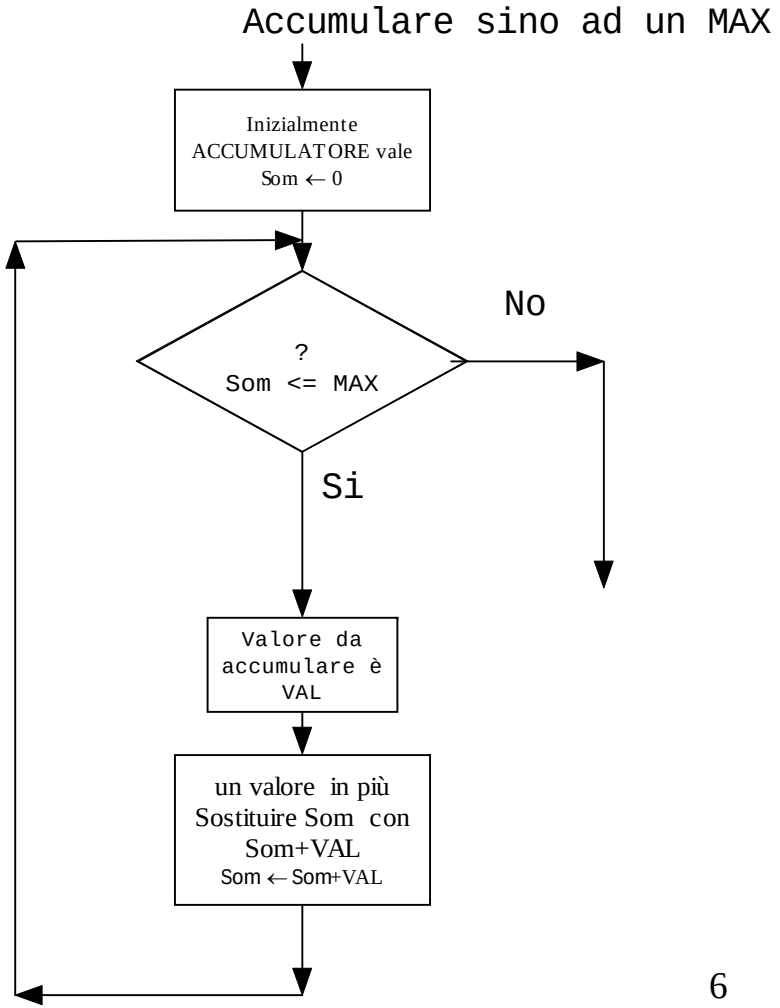
P3. “Accumulatore” – somma di N quantità

- Si deve sommare ogni numero ai precedenti
- Si deve quindi
  - Definire un Accumulatore (Som)
  - Inizialmente si deve assegnare il “valore neutro” (nel caso di somma è 0)
  - Per ogni numero nuovo si deve eseguire l’accumulo



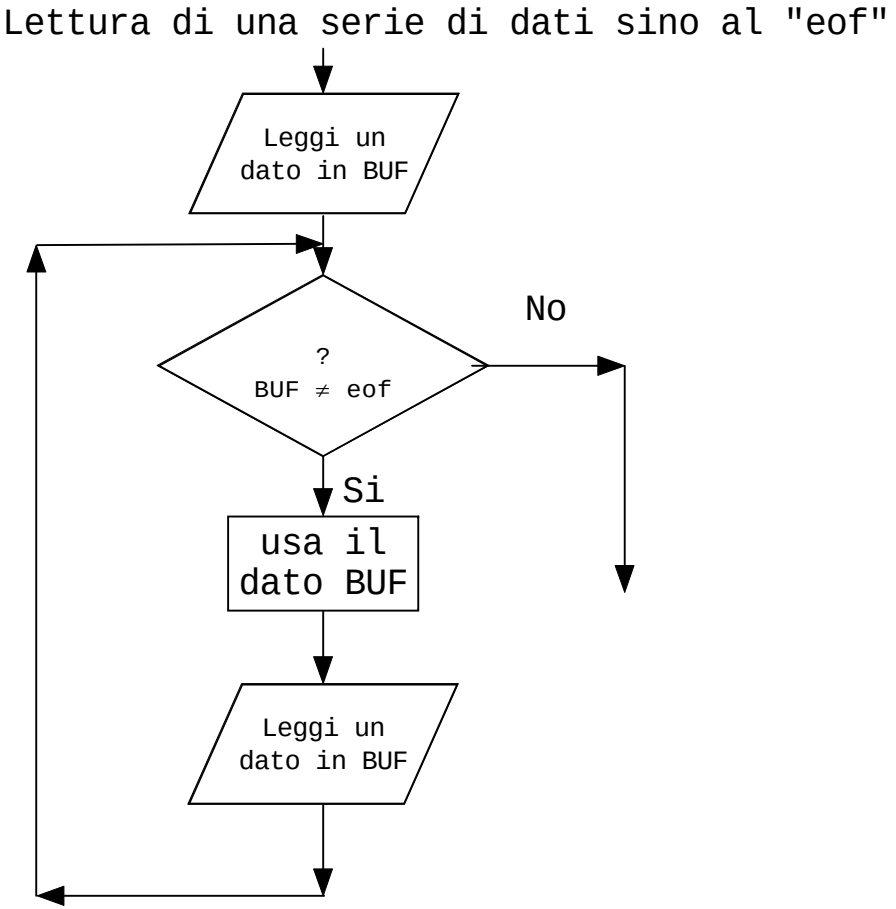
P4. Accumulare sino al superamento di un MAX

- Si deve sommare ogni numero ai precedenti
- Si deve quindi
  - Accumulare (P3)
  - Verificare il supero del valore MAX (P2)



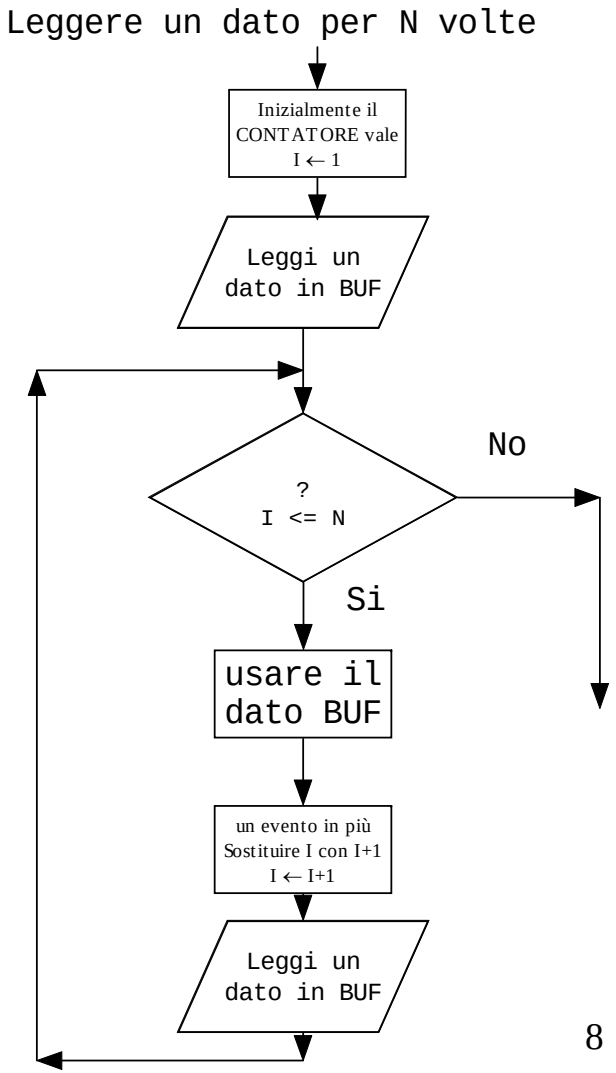
P5. Lettura di valori da un Input sino ad un “tappo” (eof)

- Si deve predisporre una variabile per accettare il valore
- Si deve quindi
  - Ciclare per leggere
  - Verificare il raggiungimento del valore “tappo”
- *Le letture di N numeri vengono fatte in N+1 volte*



P6. Lettura di valori da un Input per N volte

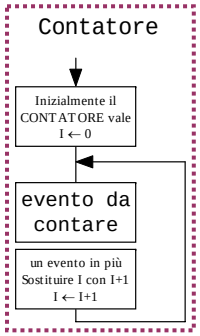
- Si deve predisporre una variabile per accettare il valore
- Si deve predisporre un contatore per il numero delle letture
- Si deve quindi
  - Ciclare per leggere
  - Verificare il superamento del numero di valori prefissati
- **ATTENZIONE – si da per certo che il dato in lettura ci sia!!**

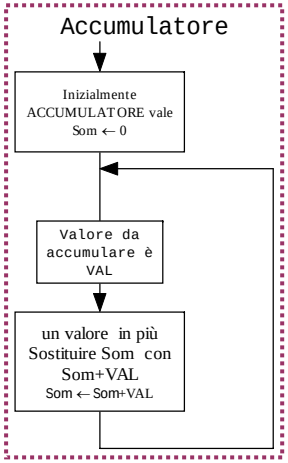


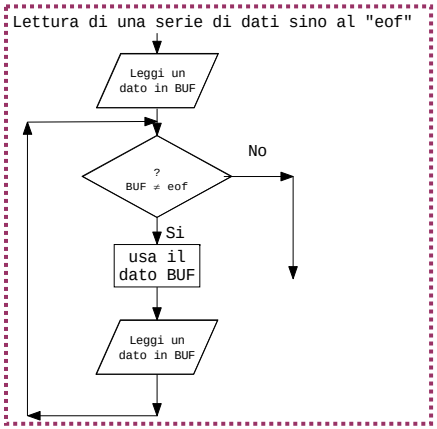
Composizione di vari Paradigmi

- Fare la media di N numeri letti da tastiera
- I numeri sono interi positivi
- Il valore di “tappo” è 0

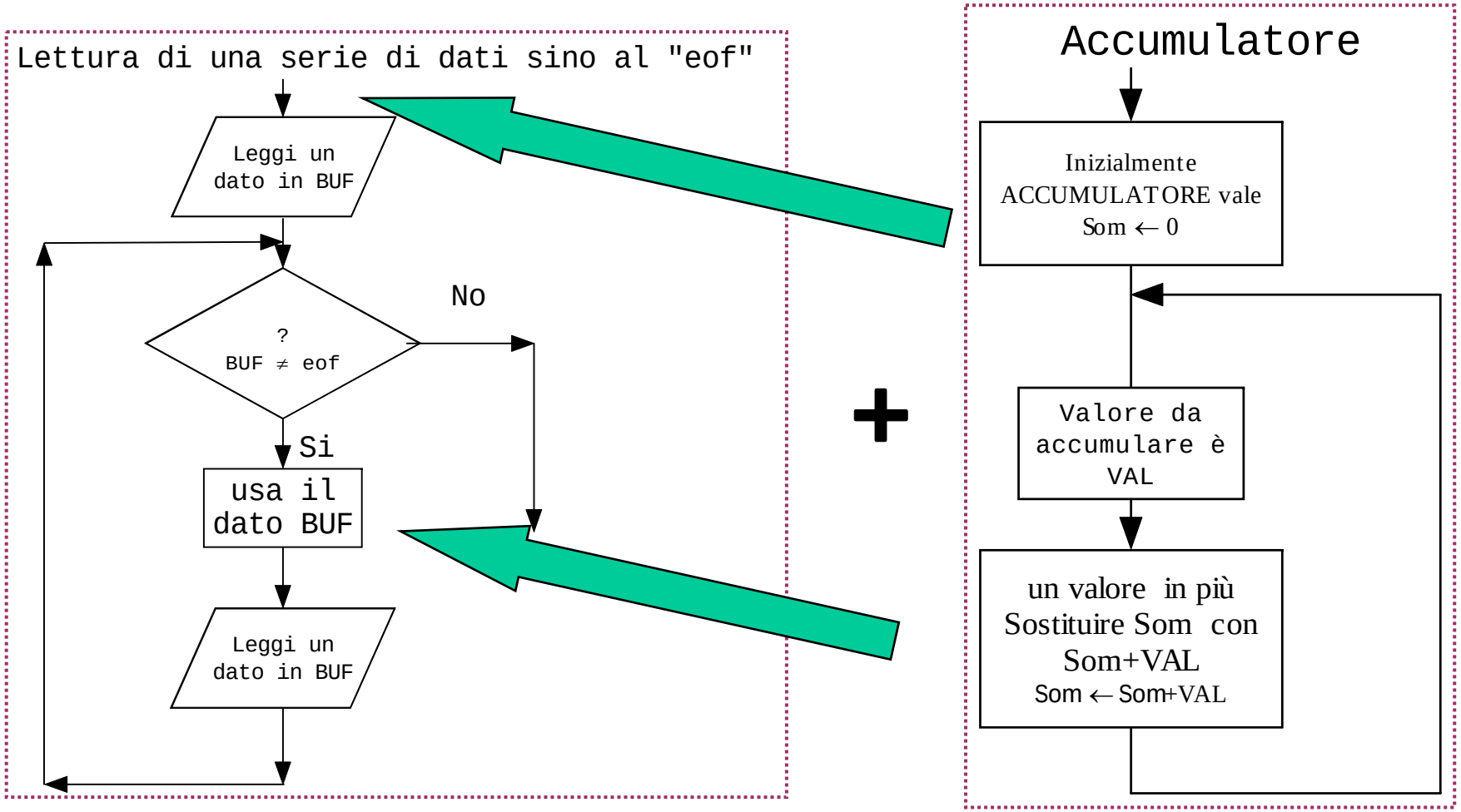
- La soluzione è la composizione di più Paradigmi
  - P1 – per fare la media si devono contare gli eventi
  - P3 – per fare la media si devono sommare gli eventi
  - P5 – si deve eseguire l’operazione (di conta e somma) sino alla lettura di eof (ovvero 0)



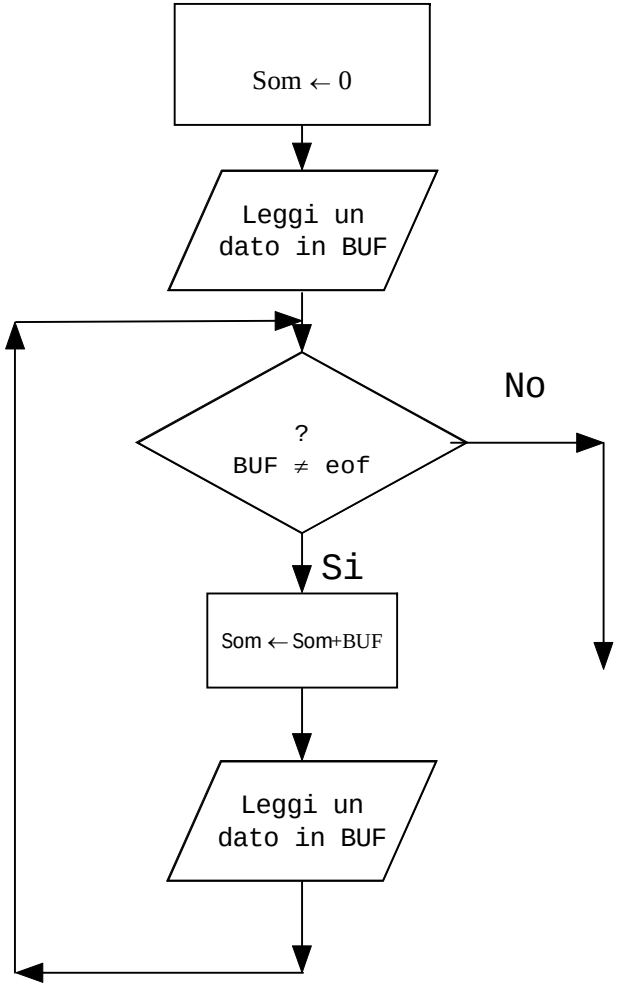




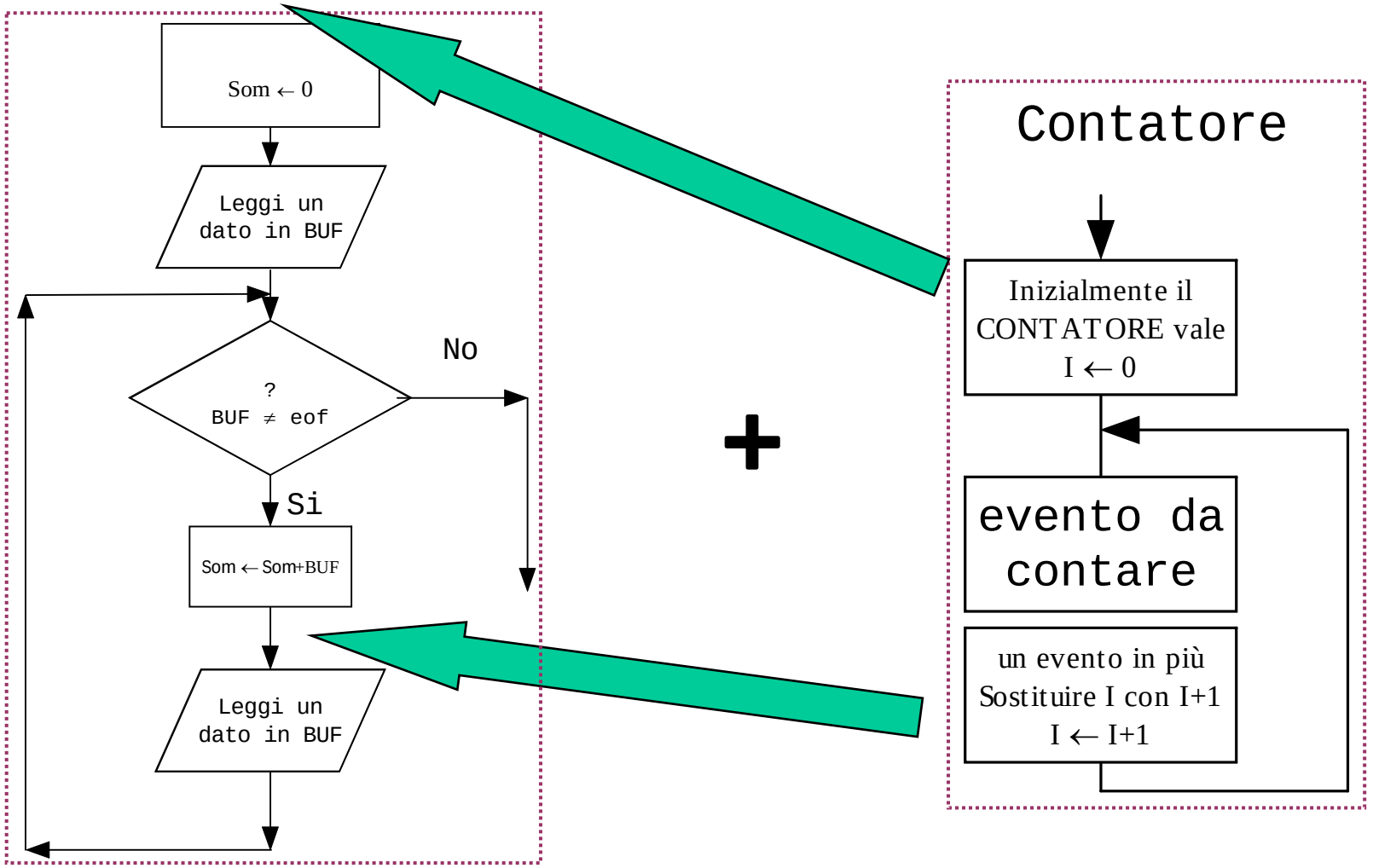
La Media dei Numeri Positivi



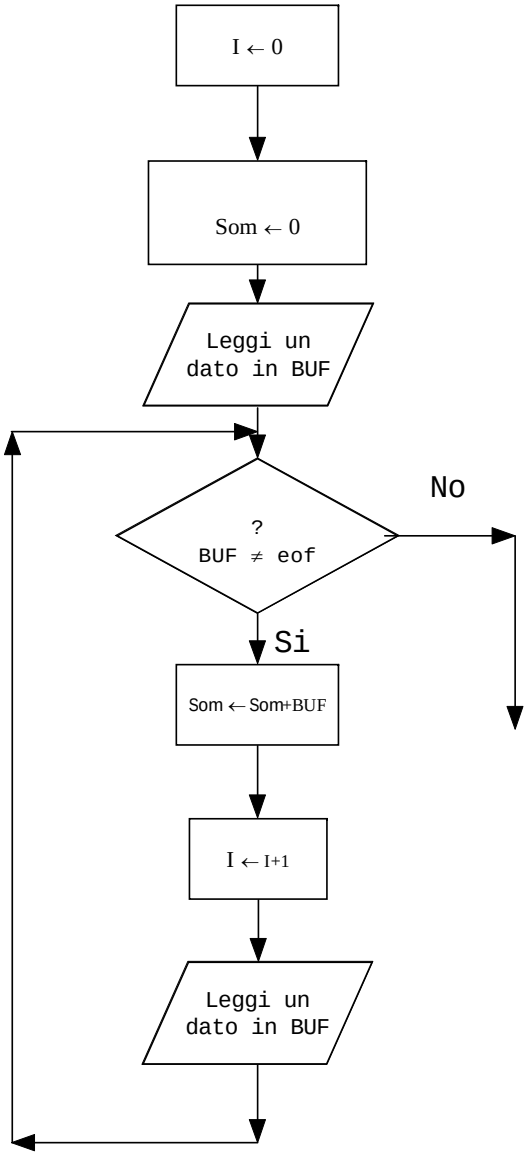
La Media dei Numeri Positivi



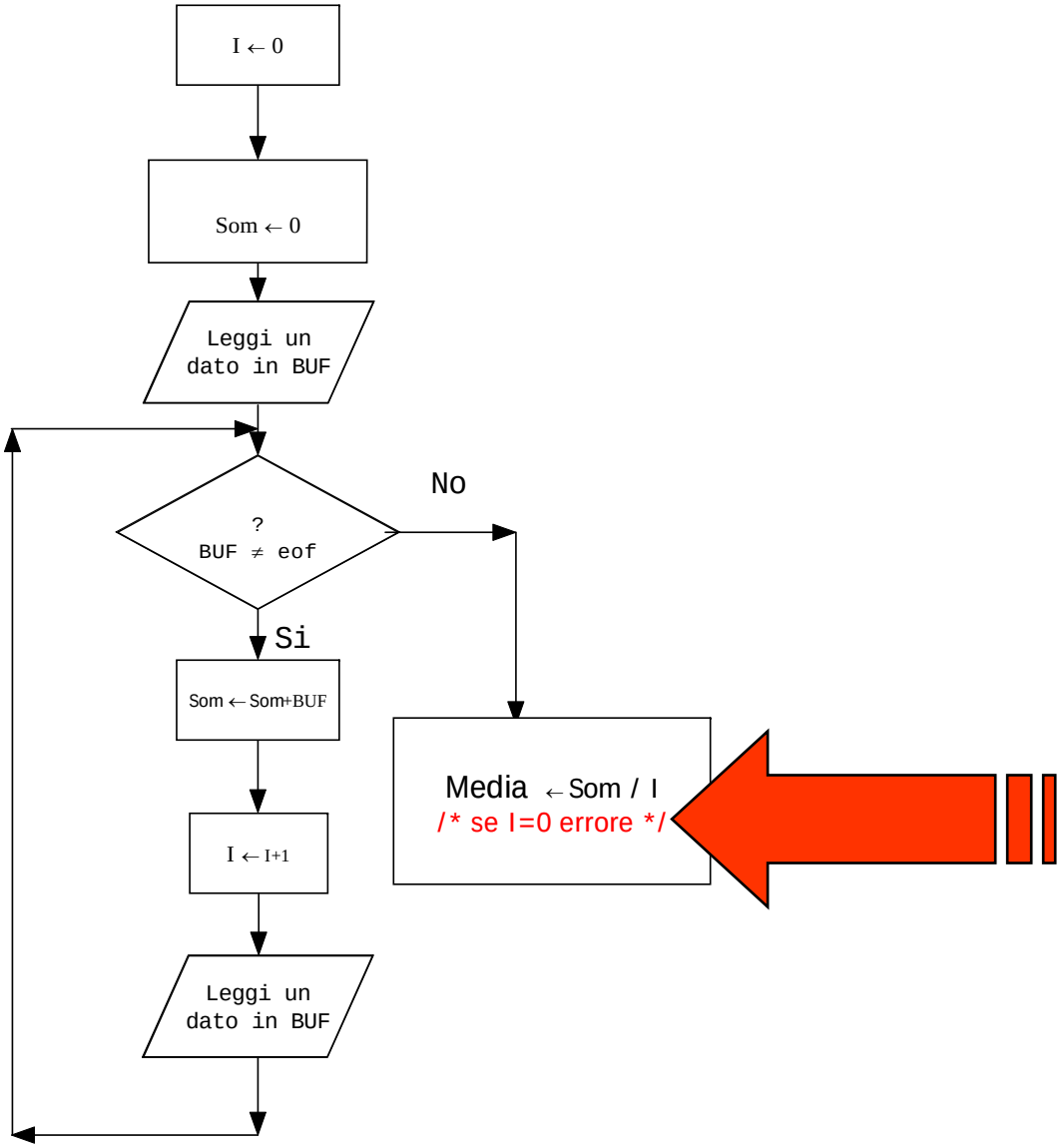
La Media dei Numeri Positivi



La Media dei Numeri Positivi



La Media dei Numeri Positivi



Conclusione

- Ogni programma è una storia a sé
- Esistono pochi schemi di base (paradigmi) che vengono usati in condizioni diverse
- Bisogna saper riconscere gli schemi ed applicarli
    - Contare i numeri in input la cui somma non fa superare il numero MAX
        - P1, P5, P4

Esempio 2: prodotto di due numeri interi tramite somme ripetute

Questo algoritmo mostra la possibilità di risolvere il problema «calcolo di un prodotto» in modo eseguibile da un esecutore in grado di eseguire somme e non prodotti.

algoritmo:

somma W a se stesso tante volte quante vale Y

variabili:

W, Y intere (rappresentano i fattori di ingresso)

variabile:

Z intera (rappresenta il risultato in uscita)

variabili ausiliarie:

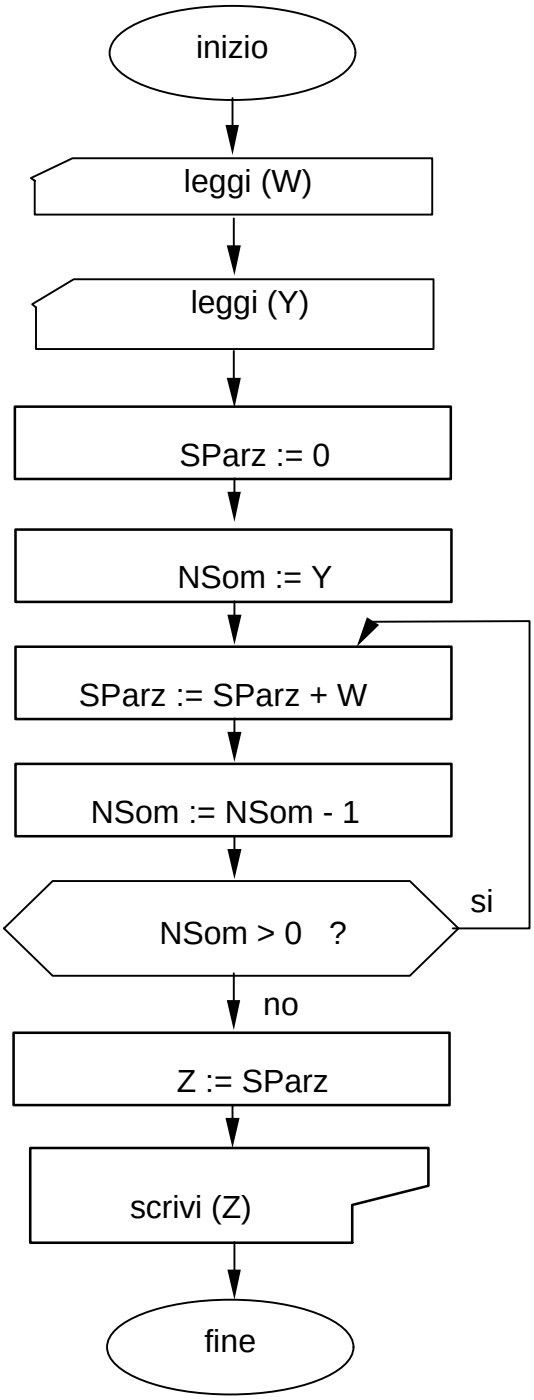
NSom intera (utilizzata per tener conto del numero di somme ancora da eseguire)

SParz intera (utilizzata per contenere il valore della somma parziale)

Esempio 2a: prodotto di due numeri interi tramite somme ripetute

*Schema a blocchi con **ciclo a condizione finale**:*

l'algoritmo è corretto se **Y > 0**



Acquisizione dei dati in ingresso e attribuzione dei loro valori alle variabili W e Y

Inizializzazione delle variabili ausiliarie prima di effettuare le vere e proprie elaborazioni

Corpo del ciclo.
Sono le operazioni da eseguire ripetutamente

Valutazione della condizione di uscita dal ciclo.

Emissione del risultato per visualizzarlo all'utente.

Flusso di controllo ciclico

Il flusso di controllo ciclico consente di esprimere nella descrizione di un algoritmo il concetto di *iterazione*, cioè di ripetizione di un insieme di operazioni (corpo del ciclo). Il corpo del ciclo viene ripetuto un numero finito di volte e la ripetizione è controllata dalla valutazione della *condizione di permanenza* nel ciclo.

La gestione di un flusso di controllo ciclico è caratterizzata da 3 elementi

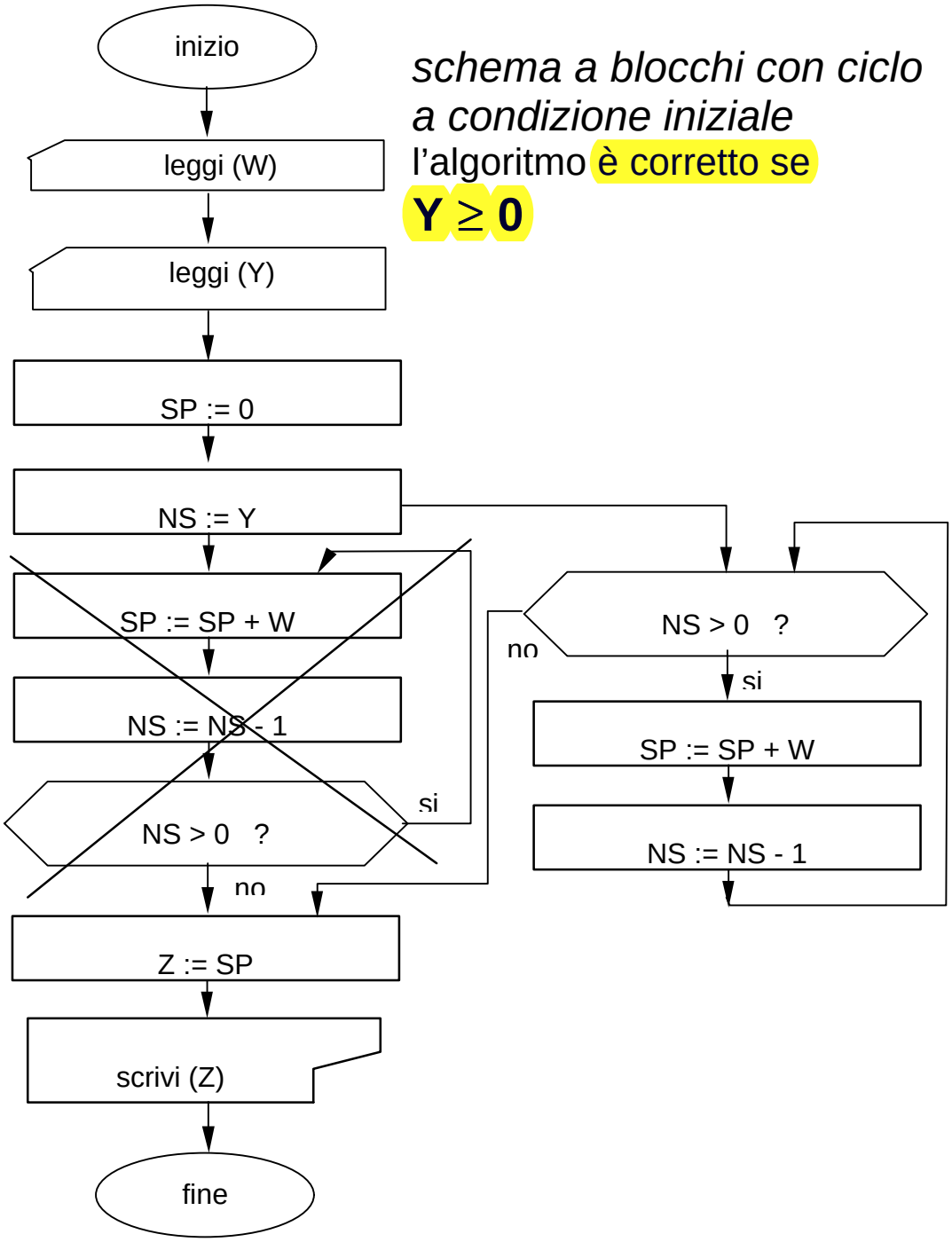
- variabile di controllo del ciclo (nell'esempio NSom), inizializzata prima di entrare nel ciclo stesso.
- condizione di permanenza nel ciclo, funzione della variabile di controllo
- corpo del ciclo che deve contenere la modifica della variabile di controllo del ciclo

Ciclo a condizione finale (usato nell'esempio precedente)

Ciclo a condizione iniziale (più usato) (usato nell'es. seguente)

Negli schemi a blocchi un flusso di controllo ciclico viene espresso utilizzando un blocco condizionale

Esempio 2b: prodotto di due numeri interi tramite somme ripetute



schema a blocchi con ciclo a condizione iniziale

l'algoritmo è corretto se

$Y \geq 0$

Algoritmo con esecutore calcolatore

Prodotto di due numeri per somme ripetute: codifica in linguaggio C (ciclo a condizione finale)

```
main ()
{
int  w, y, z, sparz, nsom;
/* dichiarazione delle variabili */

  leggi (w);
  leggi (y);

  sparz=0;       /* inizializzazione  */
  nsom=y;       /* inizializzazione  */

 /* ciclo a condizione finale: l'algoritmo
  e' corretto solo nell'ipotesi di y>0 */

  do
  {
     sparz=sparz+w;
     nsom=nsom-1;
  } while (nsom > 0);
  z=sparz;
  scrivi (z);
}
```

Prodotto di due numeri per somme ripetute: codifica in linguaggio C (ciclo a condizione iniziale)

```
main ()
{
int  w, y, z, sparz, nsom;
/* dichiarazione delle variabili */

  leggi (w);
  leggi (y);

  sparz=0;       /* inizializzazione  */
  nsom=y;       /* inizializzazione  */

 /*ciclo a condizione iniziale: l'algoritmo
 e' corretto solo nell'ipotesi di y>=0) */

  while (nsom > 0)
  {
    sparz=sparz+w;
    nsom=nsom-1;
  }
  z=sparz;
  scrivi (z);
}
```

Lo sviluppo degli algoritmi

Raffinamenti successivi

E' usuale che nel corso del progetto di un algoritmo, questo subisca raffinamenti successivi. Ciò può avvenire perché il progettista dell'algoritmo nelle prime fasi di progetto trascura volontariamente dei dettagli che si propone di sviluppare nelle fasi successive, oppure perché man mano che il progetto evolve, e quindi si conosce e si domina meglio il problema, si individuano miglioramenti, soluzioni più generali o più eleganti e arricchimenti che portano ad un algoritmo migliore.

Verifica dei dati in ingresso

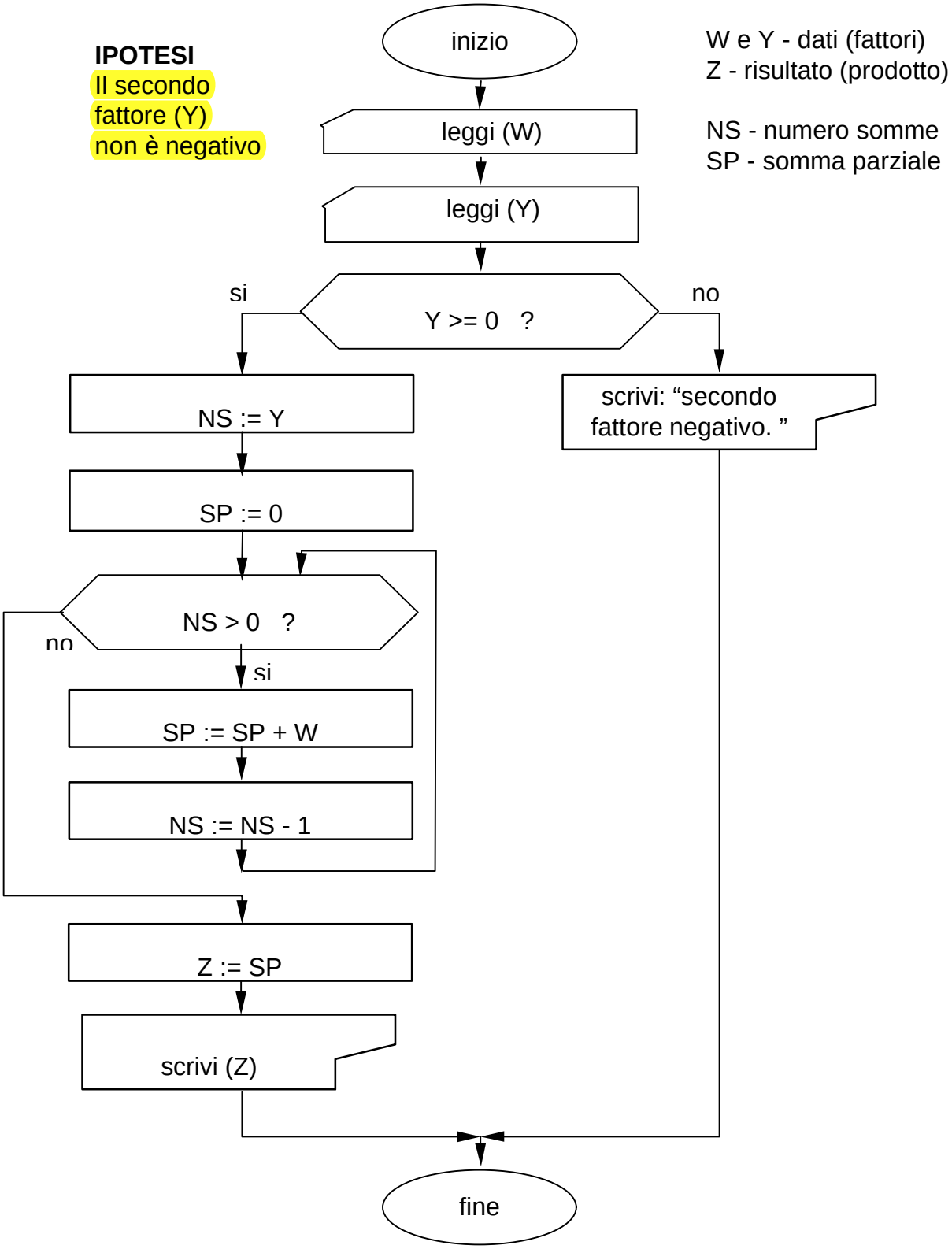
Uno dei raffinamenti tipici e molto importanti per un buon algoritmo consiste nell'introdurre operazioni che verificano la validità dei dati in ingresso. Data la possibilità che l'utente commetta errori nel fornire i dati in ingresso è opportuno verificare che i loro valori siano accettabili rispetto alle ipotesi su cui si basa la correttezza dell'algoritmo, prima di passare alle elaborazioni vere e proprie.

Ad esempio nell'algoritmo presentato precedentemente l'ipotesi è che il secondo fattore del prodotto (assegnato in ingresso alla variabile Y) sia positivo, dato che l'algoritmo così realizzato non fornisce risultati corretti per valori negativi.

Nel caso che i valori introdotti non risultino accettabili, un buon algoritmo segnala all'utente il motivo dell'inaccettabilità e consente all'utente stesso di introdurre nuovi valori o di terminare la seduta di esecuzione.

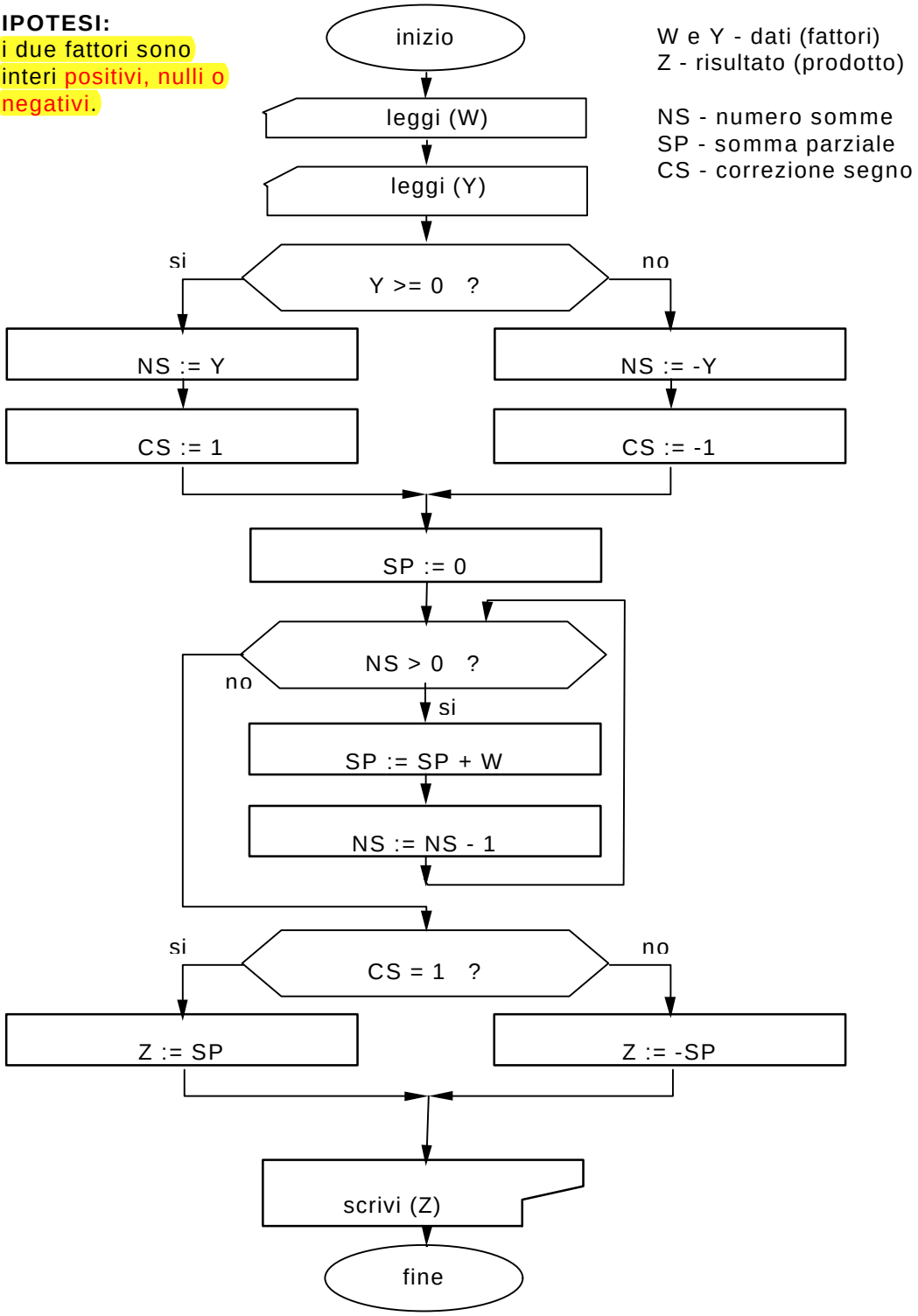
E' significativo notare che, in molti algoritmi, la parte di gestione dell'interazione con l'utente può essere molto articolata e talvolta addirittura più complicata della parte che esegue le vere e proprie elaborazioni.

Esempio 3: prodotto di due numeri interi tramite somme ripetute e verifica dei dati di ingresso



Esempio 4: prodotto di due numeri interi tramite somme ripetute

IPOTESI:
i due fattori sono interi positivi, nulli o negativi.



Codifica in C di esempio 4

```
main ()
{
 int w, y, z, sp, ns, cs;
/* dichiarazione delle variabili */

   leggi (w);
   leggi (y);

   if (y >= 0)
      {
           ns=y;          /* inizializzazione  */
           cs=1;          /* iniz. correzione segno */
      }

   else /* y è negativo */
      {
           ns=-y;         /* inizializzazione  */
           cs=-1;         /* iniz. correzione segno */
      }

   sp=0;        /* inizializzazione  */

     /* ciclo a condizione iniziale */

   while (ns >0)
      {
        sp=sp+w;
        ns=ns-1;
      }

   if (cs == 1)
      {
        z=sp;
      }
   else
      {
        z= -sp;
      }
 scrivi (z);
}
```

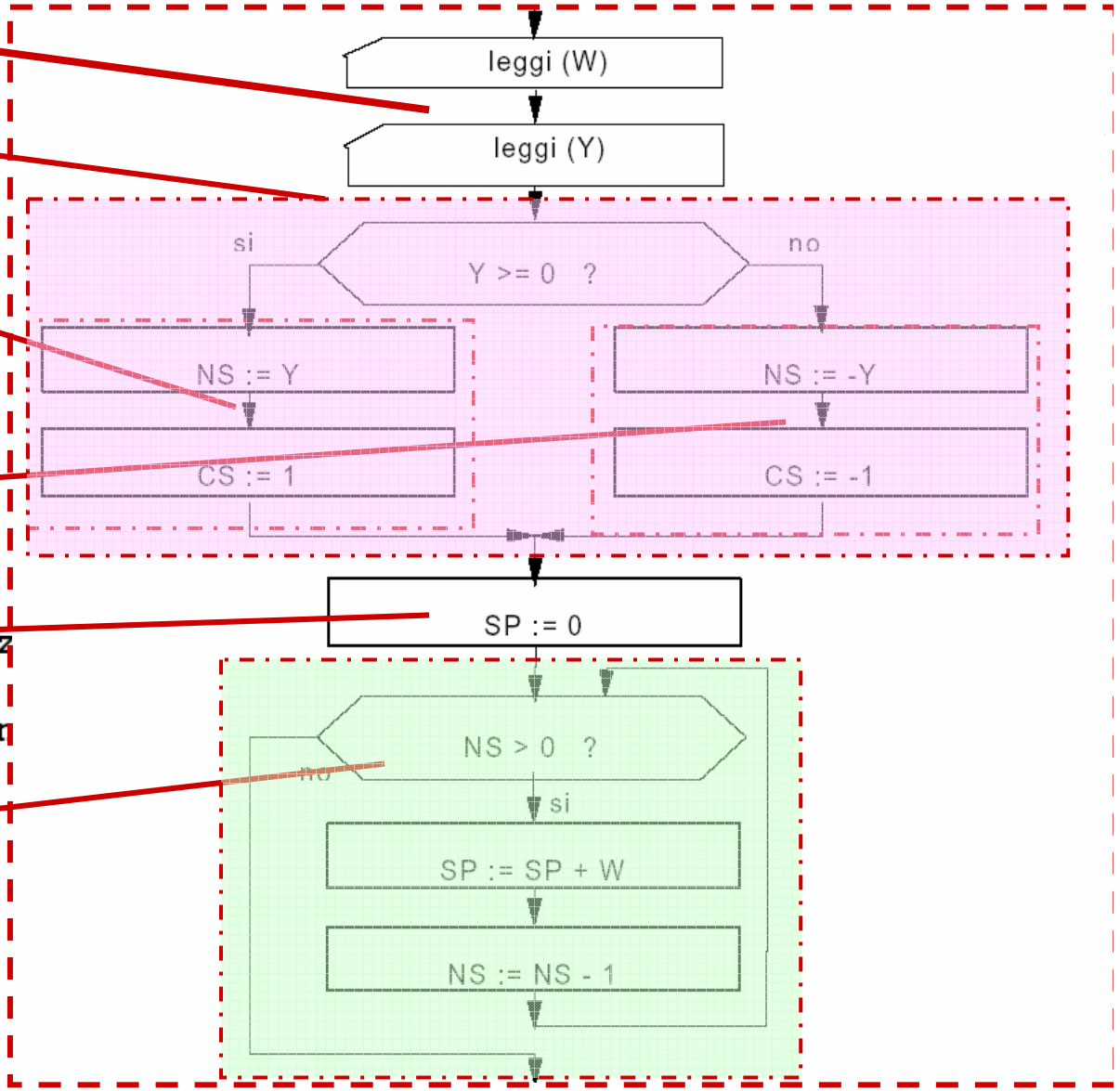
Linearizzare uno schema a blocchi

```
leggi (w);
leggi (y);

if (y >= 0)
   {
      ns=y;
      cs=1;
   }
else /* y è negativo */
   {
      ns=-y;
      cs=-1;
   }
sp=0;          /* inizializz

  /* ciclo a condizione in

while (ns >0)
   {
     sp=sp+w;
     ns=ns-1;
   }
```



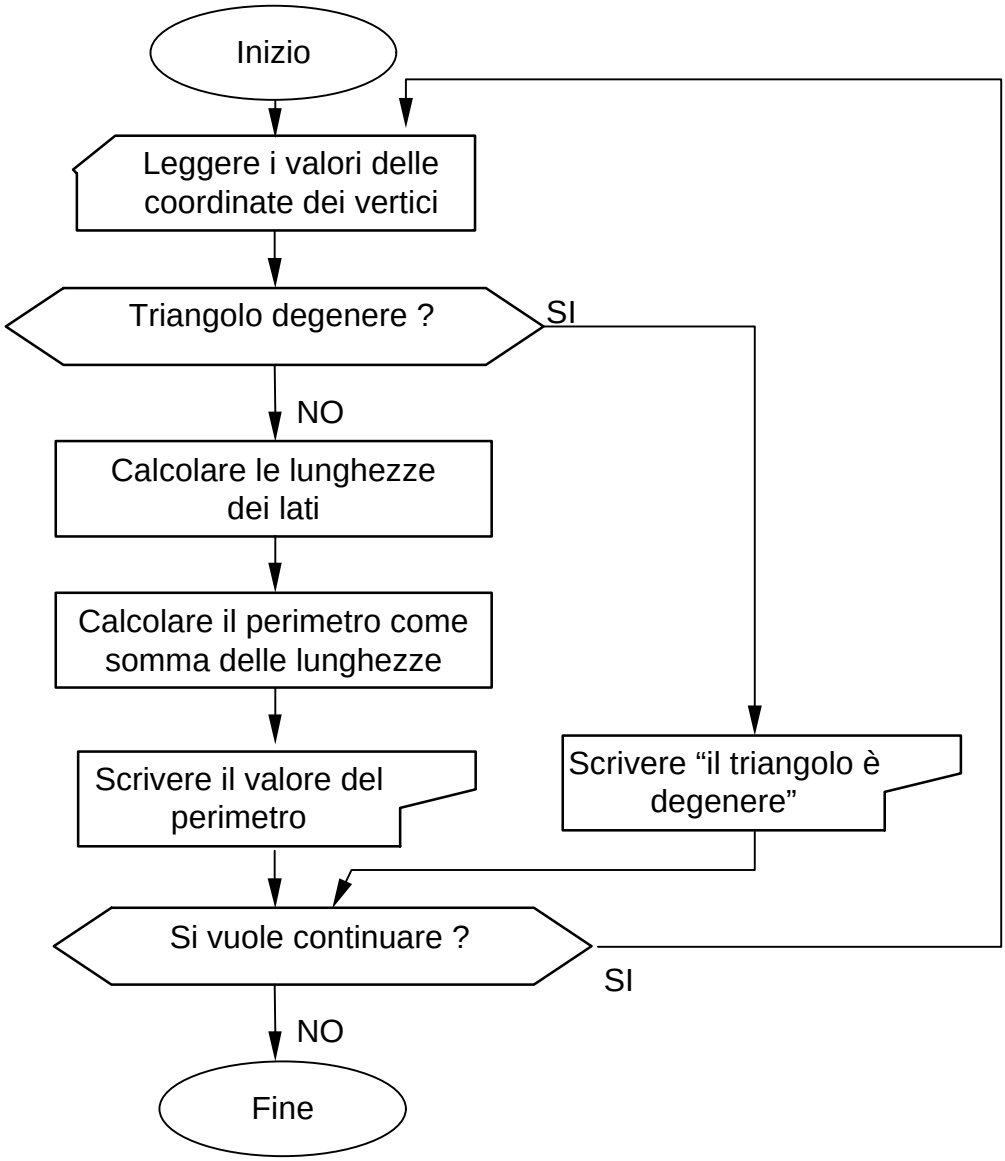
Blocchi Semplici

- Tutti i Blocchi hanno un ingresso ed una uscita
- Questo semplifica la linearizzazione
- I Blocchi sono innestati uno nell’altro

Esempio 5

Problema
Date le coordinate cartesiane di tre punti corrispondenti ai vertici di un triangolo
riconoscere se si tratta di un triangolo degenere o no,
e nel caso di triangolo non degenere calcolare il suo perimetro.

Stesura iniziale dell'algoritmo:



Stesura iniziale in pseudo-codice

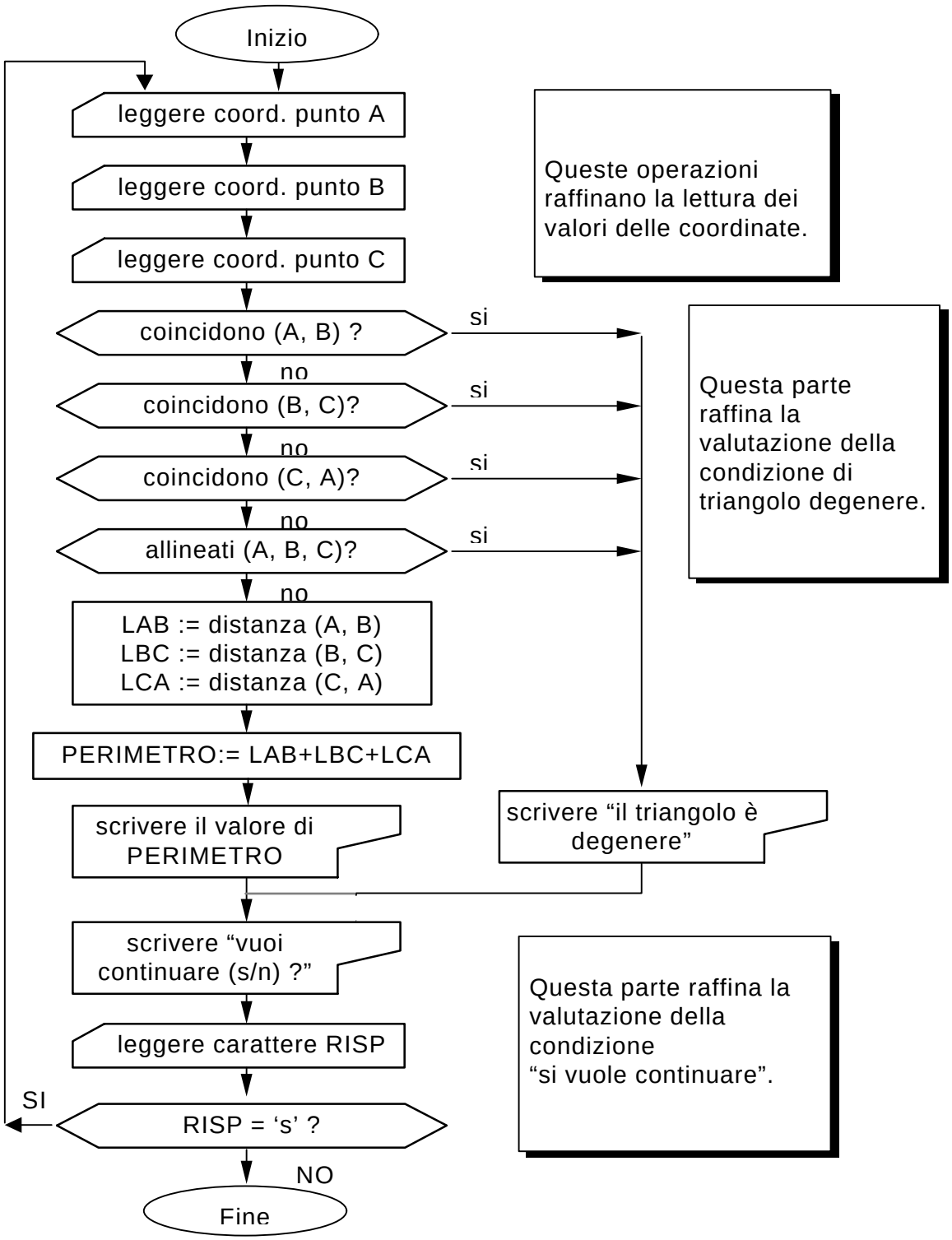
Uso dei costrutti di controllo del linguaggio C con le parole-chiave

- `main () { ...}`
- `do { ..... } while ( );`
- `if ( ) { … } else { …}`

Operazioni descritte in linguaggio naturale (italiano)

```
main ()  /* inizio dell'algoritmo */
{
   do
   {
      leggere le coordinate dei vertici
      if (triangolo degenere)
      {
         scrivi(«il triangolo e' degenere»)
      }
      else
      {
         calcolare le lunghezze dei lati
         perimetro = somma dei lati
         scrivi (perimetro)
      }
   } while (si vuole continuare);
} /* fine dell'algoritmo */
```

Raffinamento 1



Direttive «complesse» *astrazioni*: introduzione al concetto di sottoprogramma

Consideriamo «complesse» le direttive che non sono considerabili come operazioni elementari, cioè direttamente eseguibili dall'esecutore, e che quindi richiedono un ulteriore raffinamento.

Il raffinamento di direttive complesse può essere a fasi.

Infatti possiamo considerare le direttive complesse come dei problemi (o meglio sottoproblemi) da risolvere con un algoritmo a loro dedicato.

Le descrizioni di questi algoritmi «accessori» costituiscono i sottoprogrammi.

Le direttive «complesse» possono quindi essere interpretate come chiamate a sottoprogrammi.

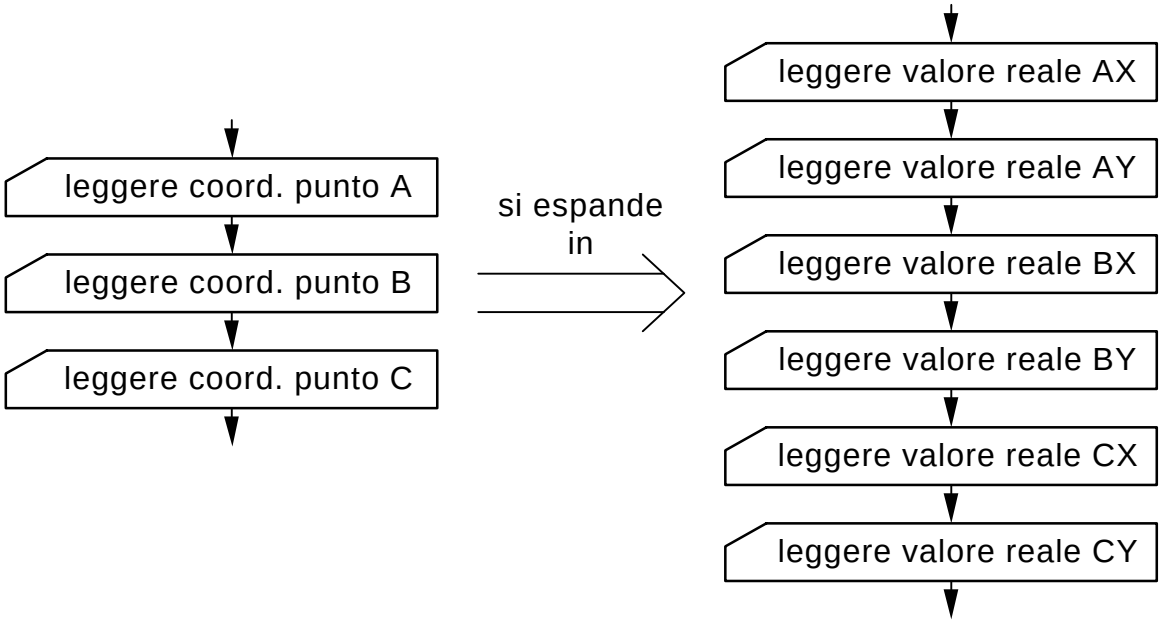
I sottoprogrammi sono molto utilizzati nella soluzione di problemi reali, dato che il loro impiego presenta diversi e importantissimi vantaggi.

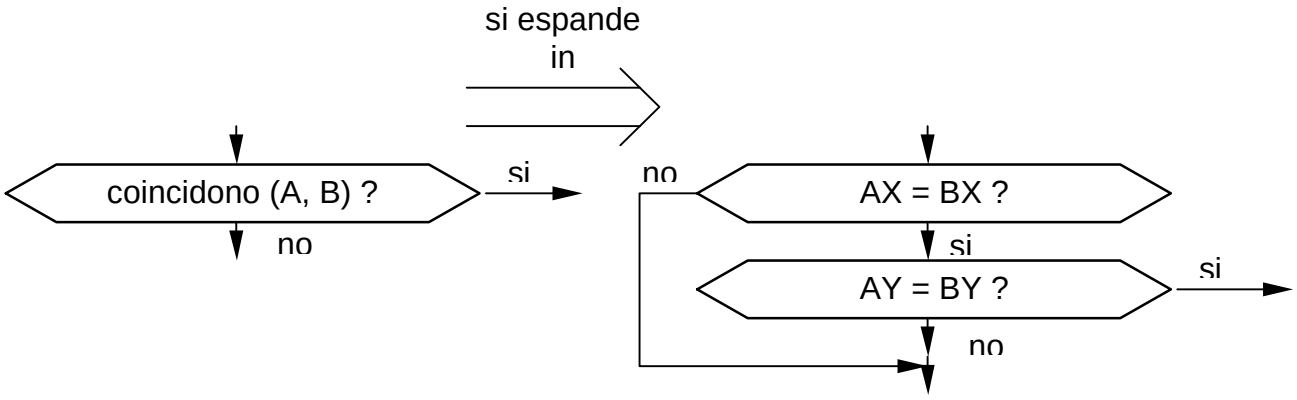
Chiarezza del programma principale. Tutti i dettagli di basso livello sono descritti a parte nei sottoprogrammi, senza affollare il programma principale che quindi descriverà in modo più comprensibile la struttura di controllo generale.

Si evitano ripetizioni. Spesso alcuni sottoproblemi devono essere affrontati più volte nell'ambito della soluzione di un problema principale. Affidando ad un sottoprogramma il compito di risolvere il sottoproblema si potrà richiamare il sottoprogramma tutte le volte che sia necessario, evitando di inserire nel programma principale tutte le volte le stesse sequenze di operazioni.

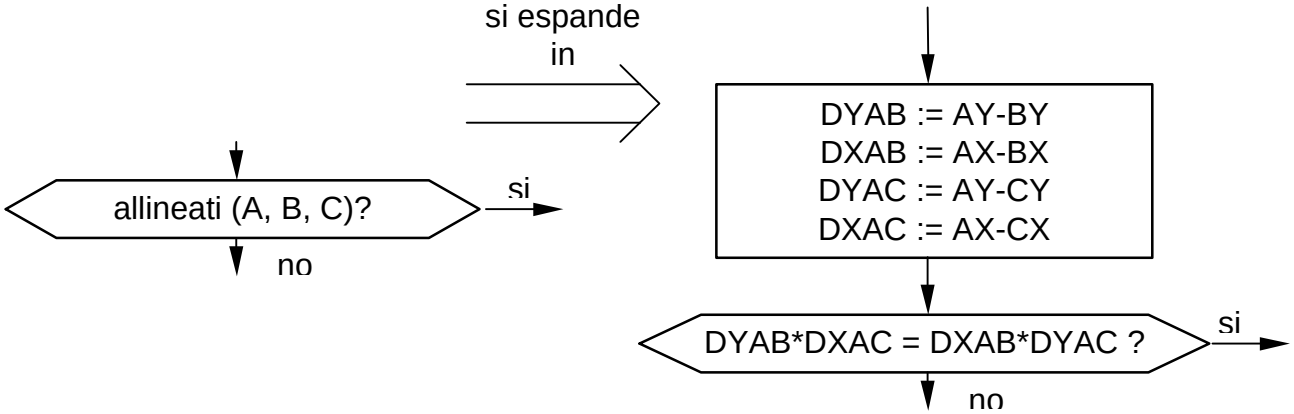
Sottoprogrammi «prefabbricati». Alcuni sottoproblemi di uso comune sono disponibili, già risolti da esperti programmatori, raccolti nelle cosiddette librerie di sottoprogrammi, evitando al programmatore di reinventare algoritmi già risolti.

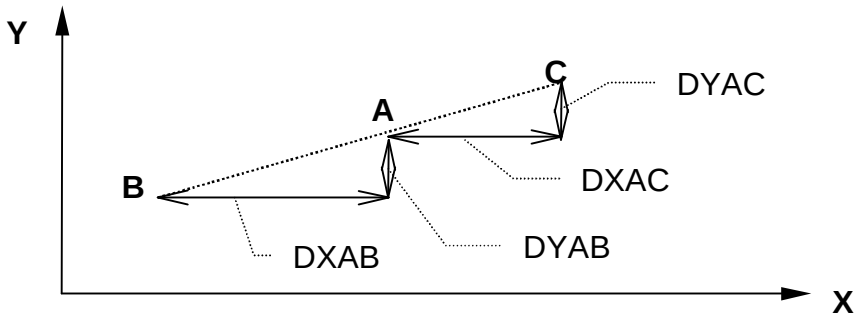
Raffinamento 2: espansione delle direttive complesse





Raffinamento 2 - seguito



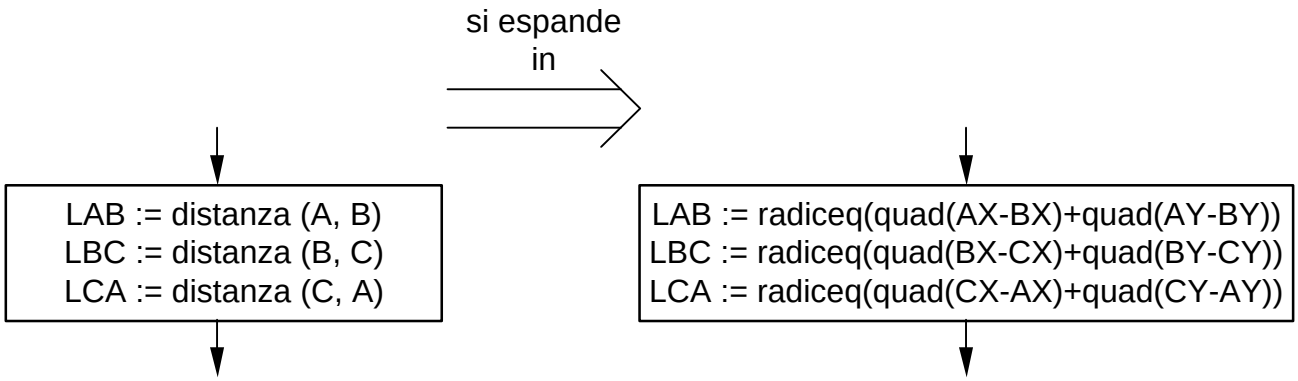


NOTA:

Se i punti A, B e C sono allineati vale la proporzione

DYAB : DXAB = DYAC : DXAC

in cui il prodotto dei medi è eguale al prodotto degli estremi



dove quad (N) sta per N*N

radiceq (N) sta per $\sqrt{N}$