U.D. 1

0 Gli Algoritmi

Fonti: Antola Dispense del Corso

Indice

INTRODUZIONE

INFORMATICA

Scienza che studia la **rappresentazione** e **l'elaborazione** delle informazioni mediante **macchine**, dette calcolatori elettronici.

In inglese: ***Computer Science***

Macchine a **funzionalità intrinseca**, o «cablata»:

sono macchine (elettroniche, ma anche meccaniche, elettriche, ecc.) il cui comportamento è determinato dalla loro **struttura** costruttiva.

Approccio **efficiente** ma rigido.

Macchine a **funzionalità programmata**:

sono macchine il cui comportamento è determinato **esecuzione di programmi**, cioè descrizioni di soluzioni di problemi.

Approccio meno efficiente ma **molto flessibile**.

La massima flessibilità di impiego delle macchine programmabili si ha quando in esse è possibile inserire, a scelta, diversi programmi.
Questa flessibilità è la causa principale della notevole diffusione dei calcolatori e dello sviluppo dell'informatica.
In questo corso ci interessiamo quindi di macchine a funzionalità programmata.

Macchine tipiche dell'informatica

- **CALCOLATORI (o elaboratori)**
  capaci di eseguire non solo calcoli aritmetici, ma anche vari tipi di elaborazioni di informazioni.
- **ELETTRONICI**
  basati sull'uso delle tecnologie elettroniche che consentono velocità, compattezza ed economicità con bassi consumi.
- **DIGITALI BINARI**
  utilizzano, per rappresentare le informazioni, segnali con un numero discreto (in particolare due) di valori distinti.
- **PROGRAMMABILI**
  in grado di eseguire le operazioni descritte da un programma.

Termini inglesi, utilizzati perché intraducibili:

HARDWARE
= materiale solido. Designa i circuiti, i dispositivi e le macchine.

SOFTWARE
= materiale soffice. Designa i programmi in generale.

Risoluzione automatica di problemi

Classi di problemi che richiedono la risoluzione automatica

Diversi problemi riguardano la gestione e l'elaborazione di informazioni. Alcuni esempi:

- raccolta, ordinamento e ricerca di grandi quantità di dati,
- esecuzione di complessi calcoli matematici per la soluzione di problemi scientifici o tecnici,
- composizione e modifica di testi e disegni,
- smistamento di conversazioni telefoniche e trasmissione di dati,
- misure ripetute di grandezze fisiche di macchine per il loro controllo automatico.

La soluzione di questi problemi può richiedere la ripetizione di un gran numero di operazioni semplici, la capacità di trattare grandi quantità di dati senza errori, la rapidità e precisione di esecuzione di operazioni complesse, ecc.

Le attitudini umane sono più adatte ad individuare dei metodi per ottenere la soluzione di questi ed altri problemi, piuttosto che ad eseguire in modo veloce, ripetitivo, instancabile e preciso le operazioni richieste da tali metodi e necessarie per ottenere le soluzioni.

Da ciò l'interesse, tutt'altro che recente, di realizzare macchine opportune in grado di eseguire automaticamente operazioni di elaborazione secondo procedimenti dettati dall'uomo.

Dato un **problema**

Le **SPECIFICHE** sono la descrizione dei requisiti a cui dovrà soddisfare la soluzione per essere considerata **corretta**.

Il **PROGETTO** è il procedimento con cui si individua o si inventa una soluzione.

La **SOLUZIONE** è la tecnica che consente di risolvere il problema e – nell'*informatica* – è basata su un **algoritmo**

Il concetto di algoritmo costituisce un aspetto fondamentale dell'informatica.

Sistema elettronico digitale

Calcolatore:

sistema **elettronico** costituito da circuiti **digitali**

In un **sistema digitale** le informazioni vengono **rappresentate**, elaborate e trasmesse mediante grandezze fisiche (segnali) che si considerano assumere solo **valori discreti**. Ogni valore è associato a una cifra (**digit**) della rappresentazione.

Al contrario, in un *sistema analogico* le informazioni vengono rappresentate (elaborate e trasmesse) mediante grandezze che possono assumere con continuità tutti gli infiniti valori in un dato intervallo.

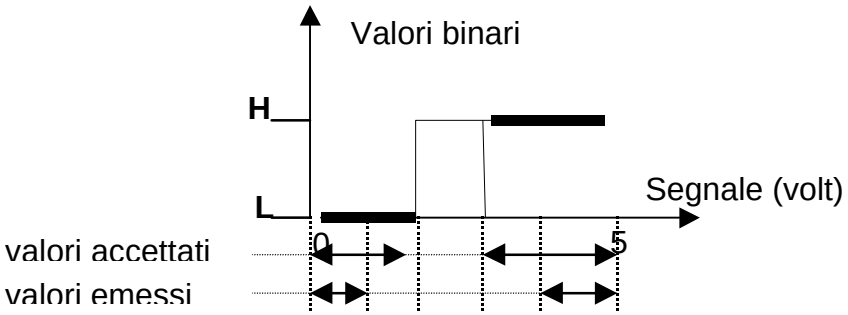
Grandezze fisiche utilizzate in un sistema digitale per la rappresentazione dell'informazione:

- ⇒ segnali **elettrici** (**tensione**, corrente)
- ⇒ grandezze di tipo magnetico (stato di magnetizzazione)
- ⇒ segnali ottici

Il comportamento fisico dei sistemi di elaborazione riproduce le operazioni di elaborazione e trasmissione delle informazioni rappresentate con le grandezze fisiche (segnali). Si ha quindi una corrispondenza biunivoca tra le operazioni sulle informazioni e le trasformazioni fisiche (operate dai circuiti) sui segnali che le rappresentano.

Rappresentazione delle informazioni con segnali digitali binari

La grandezza fisica che si utilizza (segnale elettrico di tensione) assume solo **due valori discreti** (binaria)



L'elemento tecnologico base per la realizzazione di circuiti digitali è il **transistore** il cui funzionamento può essere modellato (in modo molto semplice) come il funzionamento di un interruttore (*aperto* o *chiuso*), quindi con due stati fisici, cui corrispondono 2 opportune tensioni (in genere 0V e 5V)..

BIT (*binary digit*) = cifra binaria. (unità di informazione elementare)
Un bit può assumere due valori che possono essere associati ai simboli:

L(ow)	**H(igh)**	*aspetto fisico del segnale*
0	**1**	*aspetto aritmetico*
false	**true**	*aspetto logico*

Terminologia e «unita' di misura»

1 cifra	= bit	1 Kilo	= 2^{10} = 1024	$\cong 10^3$
8 bit	= byte	1 Mega	= 2^{20} = 1048576	$\cong 10^6$
16 bit	= word (parola)	1 Giga	= 2^{30}	$\cong 10^9$
32 bit	= double word			
64 bit	= quad word			

Sistemi elettronici digitali

Caratteristiche principali della tecnologia elettronica che utilizza segnali digitali binari:

- gli *elementi base* (realizzati utilizzando uno o più transistori opportunamente collegati) sono di pochi tipi e relativamente semplici, e sono dotati di ingressi e uscite:

  ⇒ **porte logiche** (*gate*) che realizzano gli operatori e consentono le elaborazioni
  ⇒ **elemento di memoria** (*flip-flop* o *bistabile*) che consente il mantenimento di una singola informazione binaria

Gli elementi complessi si ottengono con una «costruzione» incrementale e ripetitiva degli elementi base, cioè aggregando anche numerosi elementi base con opportune interconnessioni.
Le interconnessioni consentono la propagazione dei segnali, e quindi delle informazioni associate, dall'uscita di un elemento all'ingresso di uno o più altri elementi.

La tecnologia e il processo costruttivo dei **circuiti integrati** consentono di realizzare circuiti molto complessi in poco spazio e con un buon rapporto costi/prestazioni

Sistemi elettronici digitali

- la tecnologia elettronica digitale presenta buone proprietà per manipolare l'informazione:

  ⇒ **Miniaturizzabile**: «ci stanno tanti bit» (porte e bistabili) in poco spazio. In un circuito integrato a larga scala si arriva a 10^6 porte logiche.

  ⇒ **Veloce**: trasferimenti ed elaborazioni delle informazioni richiedono tempi brevi (nsec.=10^{-9} secondi, μsec.=10^{-6} secondi)

  ⇒ **Buona immunità ai disturbi**: due soli livelli di segnale sono facilmente riconoscibili anche se generati da dispositivi «vecchi» e in un ambiente che introduce disturbi elettrici

  ⇒ **Senza parti in movimento**: non si consuma e quindi è affidabile e dura a lungo

  ⇒ **Bassi livelli energetici**: consuma poca energia e dissipa poco calore

  ⇒ **Economia di produzione in larga scala**: il costo maggiore è il progetto, la produzione di ogni singolo dispositivo costa poche lire

Algoritmo

Obbiettivo: Dato un problema

definire un *procedimento* che possa essere eseguito *automaticamente* da un *esecutore* per risolvere il problema.

DEFINIZIONE di ALGORITMO

Dato un **problema** e un **esecutore**, l'algoritmo è:

- una *successione* finita di passi elementari (*direttive*)
- eseguibili senza ambiguità dall'**esecutore**,
- che risolve il **problema** dato.

l'algoritmo costituisce un *procedimento sequenziale* di soluzione automatica

Caratteristiche salienti degli algoritmi

- sono **procedimenti sequenziali:** viene eseguito un passo dopo l'altro secondo un ordine specificato (**flusso di esecuzione**)

- i passi elementari devono poter essere eseguiti in **modo univoco** dall'esecutore, e quindi devono essere **descritti** in una forma eseguibile per l'esecutore

Un *buon algoritmo* deve prevedere tutti i casi significativi che derivano dall'esecuzione di ogni passo elementare

La descrizione di un algoritmo per un esecutore automatico deve avere una **formulazione generale**: la soluzione individuata non deve dipendere solo da valori predefiniti dei dati.

Questa caratteristica è importante per rendere l'algoritmo utilizzabile nel maggior numero possibile di casi. Per ottenere ciò gli algoritmi generalmente prevedono appositi passi destinati ad acquisire i valori dei dati da utilizzare ed elaborare in ogni particolare esecuzione.

Elementi degli Algoritmi

In generale negli algoritmi si possono evidenziare i seguenti aspetti:

- **oggetti**
- **operazioni**
- **flusso di controllo**

Oggetti. Costituiscono le entità su cui opera l'algoritmo.

In genere si tratta di dati iniziali del problema, informazioni ausiliarie e risultati parziali e finali. Si noti che in informatica le informazioni sono generalmente dette ***dati***, anche se costituiscono risultati parziali o finali di elaborazioni. Possono essere **variabili** o **costanti** di vari tipi.

Operazioni. Sono gli interventi da effettuare sugli oggetti, cioè sui dati.

In genere si tratta di calcoli, confronti, ricopiature, acquisizioni emissioni, ecc.

Flusso di controllo. Costituisce l'indicazione delle possibili evoluzioni dell'esecuzione delle operazioni, cioè le possibili successioni dei passi

E' importante notare che la correttezza dei risultati dipende non solo dalla corretta esecuzione delle singole operazioni, ma anche dalla corretta sequenza con cui sono eseguite.

NOTA: è importante chiarire la differenza tra

Flusso di controllo e flusso di esecuzione.

- Il **flusso di controllo** è la descrizione a priori di tutte le possibili sequenze nell'esecuzione dei passi dell'algoritmo, in particolare di operazioni in alternativa e di operazioni da ripetere più volte ciclicamente.

- Il **flusso di esecuzione** è la sequenza di operazioni effettivamente seguita durante una particolare esecuzione dell'algoritmo e che dipende dai particolari valori che i dati assumono in quella esecuzione.

Algoritmo adatto per esecutore umano

Esempio: istruzioni di montaggio

MONTAGGIO DI UN MODELLO DI AUTO

1. Inserire le ruote sugli assali
2. Montare gli assali sul telaio
3. Verificare la scorrevolezza
4. Montare la carrozzeria sul telaio
5. Inserire volante e leva cambio negli appositi fori
6. Applicare il tettuccio.

- **Oggetti**: ruote, assali, telaio, carrozzeria, ecc.
- **Operazioni**: inserire, montare, applicare, ecc.
- **Flusso di controllo**: è la sequenza implicita nella successione delle frasi.

E' una descrizione **non rigorosa** del procedimento sequenziale, infatti:

- usa il linguaggio naturale (non rigoroso)
- il significato delle direttive non e' univoco
- alcune operazioni sono descritte in modo approssimativo
- alcune operazioni sono implicite
- richiede intuito e buon senso

però è adatta per un esecutore umano.

Rappresentazione di un algoritmo destinato all'esecuzione automatica

La rappresentazione di un algoritmo è la descrizione, univoca per l'esecutore, di tutte le possibili sequenze di operazioni da eseguire per risolvere il problema dato.
E' quindi la ***descrizione*** del ***flusso di controllo,*** delle ***operazioni eseguibili***, e degli ***oggetti*** su cui agiscono le singole operazioni

E' necessario un supporto ***formale*** alla descrizione di un algoritmo, cioè un formalismo (linguaggio) costituito da:

1. un insieme di **elementi** per la descrizione di oggetti, operazioni e flusso di controllo: ***vocabolario***

2. un insieme di **regole** per la **composizione** degli elementi in frasi eseguibili e costrutti di controllo (istruzioni): ***sintassi***

Il rispetto delle *regole sintattiche* deve poter essere verificato in modo automatico.

3. un insieme di **regole** per **l'interpretazione** degli elementi e delle istruzioni sintatticamente corrette: ***semantica***

Linguaggi di descrizione di algoritmi

La rappresentazione di un algoritmo destinato all'esecuzione automatica deve essere basata su
formalismi descrittivi costituiti dai
LINGUAGGI (artificiali) DI PROGRAMMAZIONE

I linguaggi di descrizione degli algoritmi rappresentano gli **oggetti** tramite dei **nomi simbolici** (identificatori).
In particolare si introduce il concetto di **variabile**: una variabile costituisce la **rappresentazione simbolica di dati** cui si possono attribuire diversi valori di un certo tipo.

I linguaggi rappresentano le **operazioni** mediante **operatori** (es. +) o **funzioni** (es. cos(x)).

I linguaggi rappresentano la sequenza di operazioni (**flusso di controllo**) con appositi **costrutti** di controllo.

Linguaggi di descrizione
Corrispondenza tra concetti e loro rappresentazione

Concetto	*Rappresentazione*
oggetto	identificatore
operazione	operatore, funzione
direttiva	istruzione
flusso di controllo	costrutti di controllo
algoritmo	programma

Linguaggi di descrizione

I linguaggi per descrivere algoritmi devono essere noti all'uomo che progetta gli algoritmi e al calcolatore che deve eseguire tali algoritmi.

Soprattutto nelle fasi iniziali di progetto sono molto utili all'uomo dei linguaggi che preservano la descrizione **rigorosa del flusso di controllo**, ma consentono la descrizione **semplificata delle direttive**, anche se non sono ancora eseguibili dal calcolatore.
Infatti una descrizione informale delle direttive, ad esempio mediante l'uso di un linguaggio naturale, consente al progettista di concentrarsi sulla struttura generale dell'algoritmo (cioè del suo flusso di controllo), senza dover subito risolvere tutti i dettagli che verranno precisati nei raffinamenti successivi e che sono peraltro necessari per rendere l'algoritmo effettivamente eseguibile.

Due tipici esempi di linguaggi semiformali sono:

- schemi a blocchi
- «pseudo-codice»

Schemi a blocchi

Vengono anche detti **diagrammi di flusso** (*flow chart*). Sono un formalismo descrittivo che fa uso di elementi grafici per indicare il flusso di controllo e per indicare i tipi di operazioni.
Fa invece uso di elementi testuali per descrivere le operazioni e gli oggetti su cui esse operano.
Questa caratteristica li rende semplici da usare e facilmente leggibili, in particolare per algoritmi di limitata complessità e soprattutto nelle fasi di bozza iniziale della descrizione.

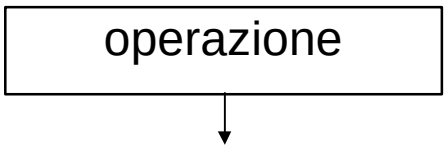
Pseudo-codice

La rappresentazione con pseudo-codice è completamente testuale.
Un aspetto molto interessante è che i costrutti di controllo sono descritti con la forma e le parole chiave corrispondenti a quelle dei linguaggi di programmazione, mentre le operazioni possono essere descritte in modo informale e sintetico.
La somiglianza strutturale con i programmi definitivi rende le descrizioni degli algoritmi con pseudo-codice molto adatte alle fasi di sviluppo dei programmi stessi.

Schemi a blocchi

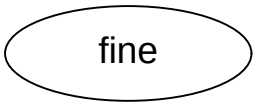
Gli **elementi base,** di tipo grafico, che vengono adottati per la descrizione di algoritmi sono:

- **blocco esecutivo**



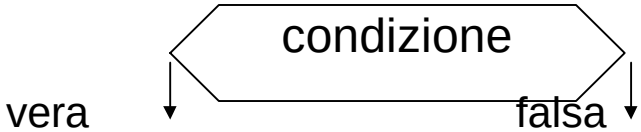
- **blocco di inizio dell'esecuzione**

- **blocco di terminazione**

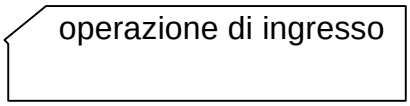


- **flusso di controllo delle operazioni**

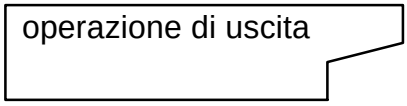
- **blocco condizionale**



- **blocco di ingresso dati**



- **blocco di uscita dati**



Schemi a blocchi

Come in quasi tutti i linguaggi, negli schemi a blocchi le variabili vengono rappresentate tramite nomi simbolici

L'uso di nomi simbolici per le variabili (operandi) consente di descrivere operazioni da eseguire di volta in volta sui diversi valori assegnabili a tali variabili. In tal modo è possibile scrivere algoritmi di

- una **variabile** può essere definita come un «contenitore» di valori

  proprietà

  una variabile non può essere «vuota», cioè ad una variabile è sempre associato un valore che può essere significativo o non significativo

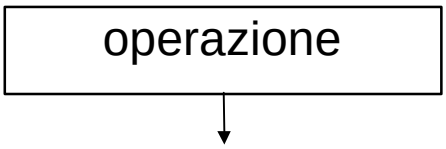
Ovviamente al momento dell'esecuzione di operazioni che «usano» la variabile, essa deve avere un valore significativo. Una variabile ha un valore significativo dopo che è stata effettuata un'operazione di ingresso dati che le attribuisce un valore acquisito dall'esterno, oppure un'operazione di calcolo che le assegna il risultato di un'espressione.

Negli schemi a blocchi le **operazioni** e le **condizioni** vengono espresse in modo testuale ed eventualmente tramite simboli che rappresentano operatori aritmetici, di confronto, ecc.

Schemi a blocchi

Significato degli elementi

BLOCCO ESECUTIVO:



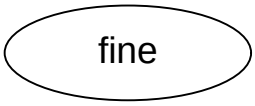
identifica un'operazione da eseguire (espressa in forma testuale al suo interno). Per semplicità di rappresentazione può anche contenere un insieme di operazioni da eseguirsi in modo sequenziale

BLOCCO DI INIZIO DELL'ESECUZIONE:

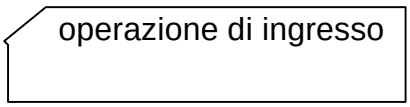
identifica il primo blocco che deve essere eseguito

BLOCCO DI TERMINAZIONE:

contrassegna una possibile terminazione dell'esecuzione dell'algoritmo



BLOCCO DI INGRESSO DATI:

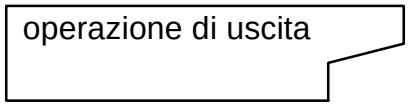


identifica un particolare blocco esecutivo che contiene una direttiva di ingresso dati, cioè un'operazione che consente di acquisire in ingresso al «programma» valori assegnandoli a variabili. Questi valori determinano la particolare *istanza* di esecuzione dell'algoritmo.
Ad esempio l'operazione di ingresso ***Leggi num*** determina l'acquisizione di un valore, inserito dall'utente, che viene assegnato alla variabile di nome simbolico ***num***.
Le direttive di ingresso consentono l'interazione tra «mondo esterno» e calcolatore (esecutore

Le direttive di ingresso più semplici e di uso più comune sono quelle che consentono di acquisire

Ci sono anche operazioni di ingresso da altri dispositivi.

BLOCCO DI USCITA DATI:



identifica un blocco esecutivo che contiene una direttiva di uscita, cioè un'operazione che consente di emettere in uscita dal «programma» frasi di testo e/o valori assunti da variabili.
Ad esempio l'operazione di uscita ***Scrivi num*** produce la visualizzazione del valore che in quel momento è contenuto nella variabile di nome simbolico ***num***.
Le direttive di uscita consentono l'interazione tra calcolatore (esecutore dell'algoritmo) e «mondo

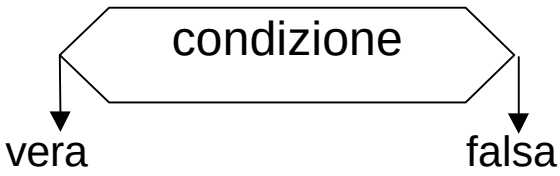
Le operazioni di uscita più semplici e di uso più comune sono quelle che consentono di

Analogamente a quanto disponibile per l'acquisizione di valori in ingresso, ci sono operazioni per l'emissione di dati in uscita su altri dispositivi.

FRECCE E BLOCCO CONDIZIONALE:

consentono di rappresentare in modo completo e univoco il flusso di controllo.

- Generalmente dopo l'esecuzione di un blocco sarà unico il blocco che costituisce il passo successivo.
  La freccia uscente da un blocco esecutivo indica l'unico blocco che segue nel flusso di controllo.
- Quando si presentano due alternative si introducono i blocchi condizionali.



Un blocco condizionale propone un flusso di controllo di selezione tra due alternative. Esso contiene una **condizione**, cioè un'espressione, che può risultare **vera** o **falsa**, a seconda dei valori assunti in quel momento da variabili. In base al risultato della valutazione di tale condizione, il flusso di esecuzione segue uno o l'altro dei due rami rappresentati da frecce uscenti dal blocco condizionale stesso.

Schemi a blocchi

Regole di composizione degli elementi di uno schema a blocchi

La composizione degli elementi ha lo scopo di definire il **flusso di controllo** dell'algoritmo, cioè di definire **in modo univoco** quali sono tutte le possibili sequenze di blocchi da eseguire.
Per garantire questa univocità occorre rispettare le seguenti regole sintattiche.

- Deve esistere un solo blocco d'inizio
- Deve esistere almeno un blocco di terminazione
- Dal blocco d'inizio e da ogni blocco esecutivo (compresi quelli di ingresso e uscita dati) deve uscire una sola freccia
- Da ogni blocco condizionale devono uscire due frecce contrassegnate dalle indicazioni vero (si) e falso (no).

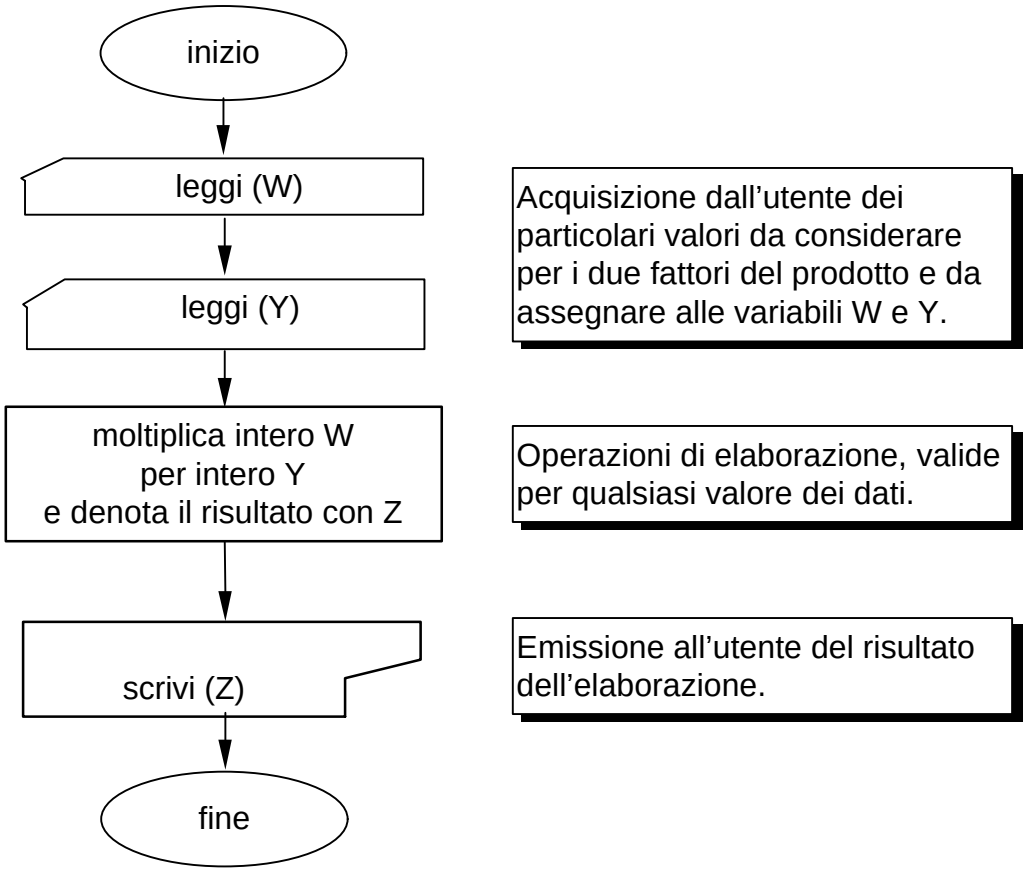
Se per ogni blocco c'è un solo blocco successivo, si parla di flusso di controllo sequenziale (*sequenza*)

Il **flusso di controllo** non è più sequenziale se dopo un blocco si possono presentare diverse *alternative*, cioè diverse sequenze di operazioni da eseguire in alternativa a seconda dell'esito della valutazione di una condizione (in questo caso, come si è detto, si deve usare un *blocco condizionale*).

In **tutti** i casi – **in esecuzione** – e' **unica** la scelta del blocco successivo da eseguire

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

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:

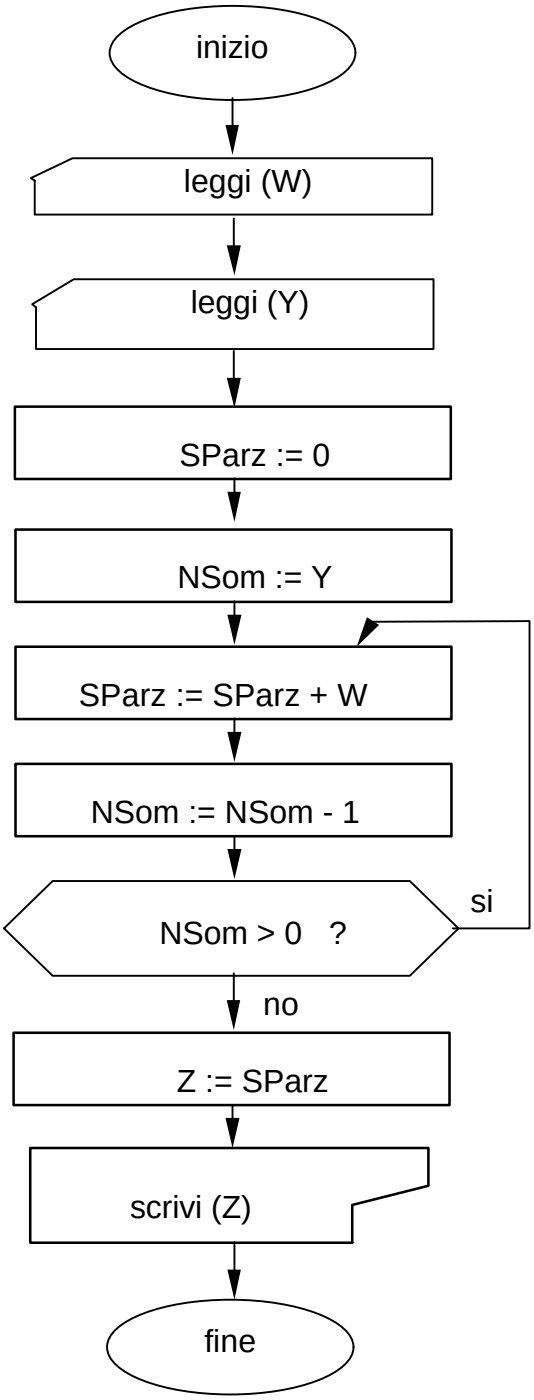
NSom intera (utilizzata per tener conto del numero di somme ancora da eseguire)

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

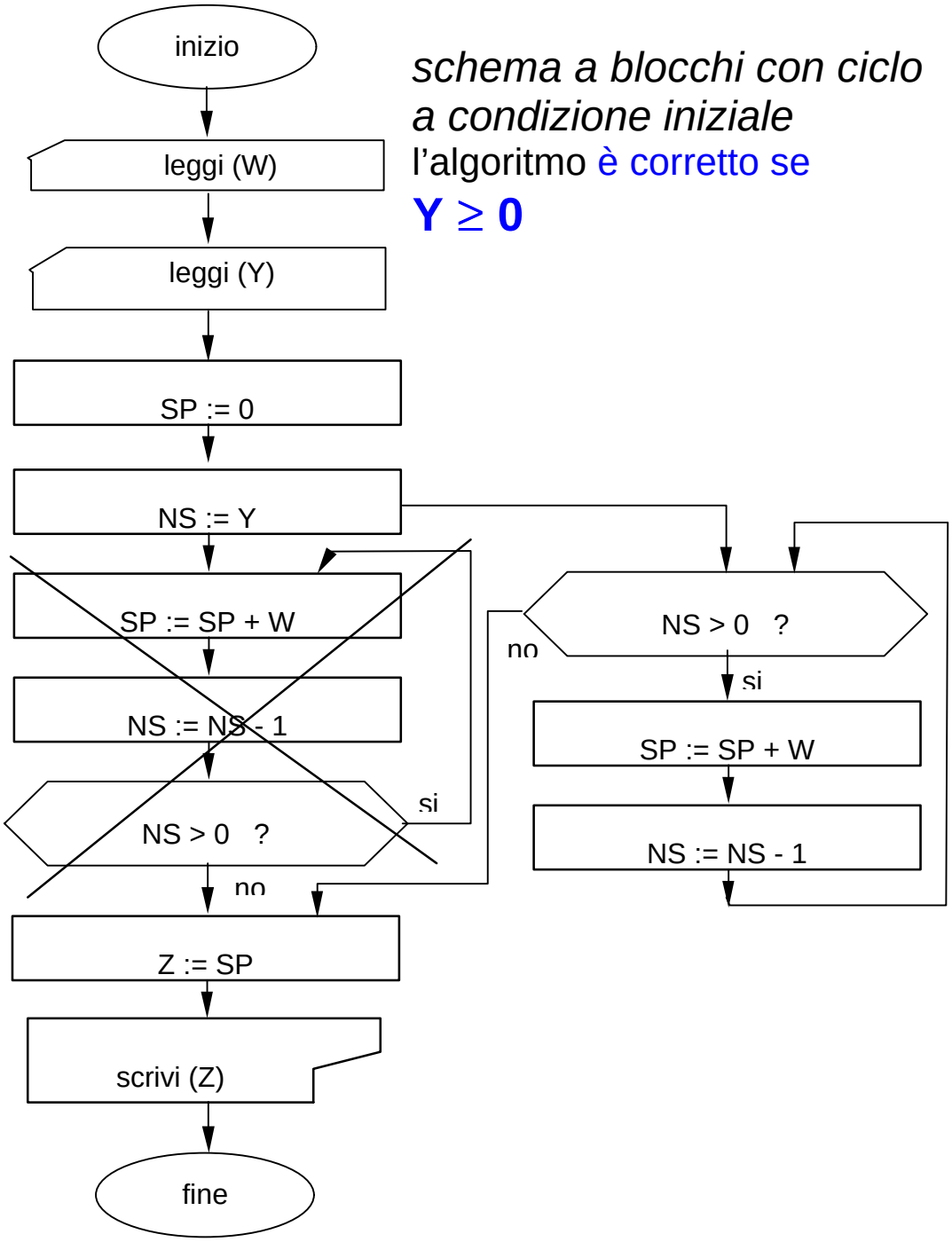
- variabile di controllo del ciclo (nell'esempio NSom), inizializzata prima di entrare nel ciclo stesso.
- condizione di permanenza nel ciclo, funzione della variabile di controllo
- corpo del ciclo che deve contenere la modifica della variabile di controllo del ciclo

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

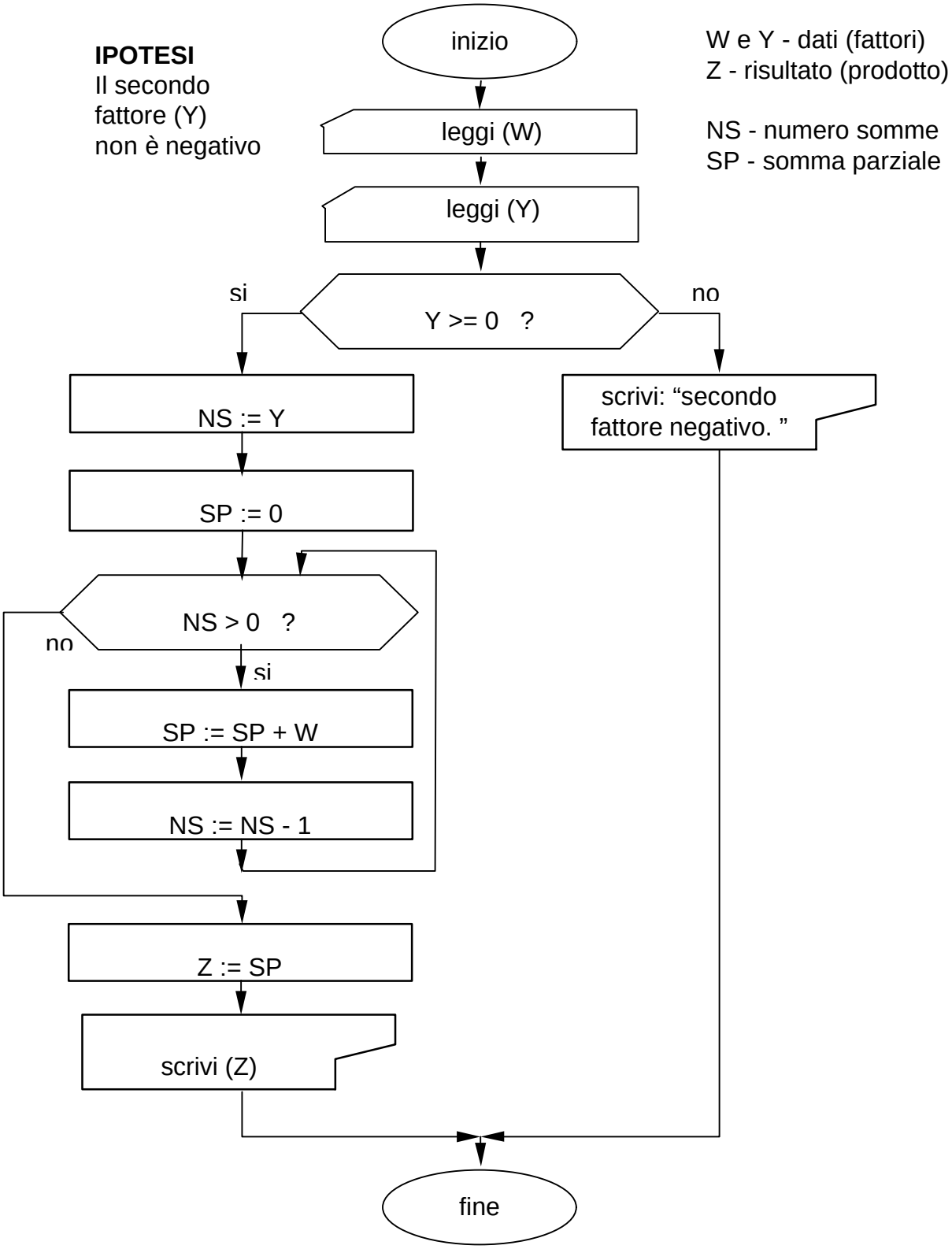
Uno dei raffinamenti tipici e molto importanti per un buon algoritmo consiste nell'introdurre operazioni che verificano la validità dei dati in ingresso. Data la possibilità che l'utente commetta errori nel fornire i dati in ingresso è opportuno verificare che i loro valori siano accettabili rispetto 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



Flusso di controllo condizionale

Selezione semplice

Si utilizza un costrutto di selezione semplice quando alcune operazioni vanno eseguite oppure non eseguite, in dipendenza del verificarsi di una data condizione.

> Il valore della condizione dipenderà a sua volta dai valori di opportune variabili a cui sono stati precedentemente assegnati valori forniti in ingresso dall'utente oppure ottenuti come risultato del calcolo di espressioni.

Con gli schemi a blocchi il costrutto di selezione semplice viene realizzato con un blocco condizionale. Lungo uno dei due rami uscenti da tale blocco sono collocati i blocchi che descrivono le operazioni da eseguire o da «saltare».

Selezione doppia

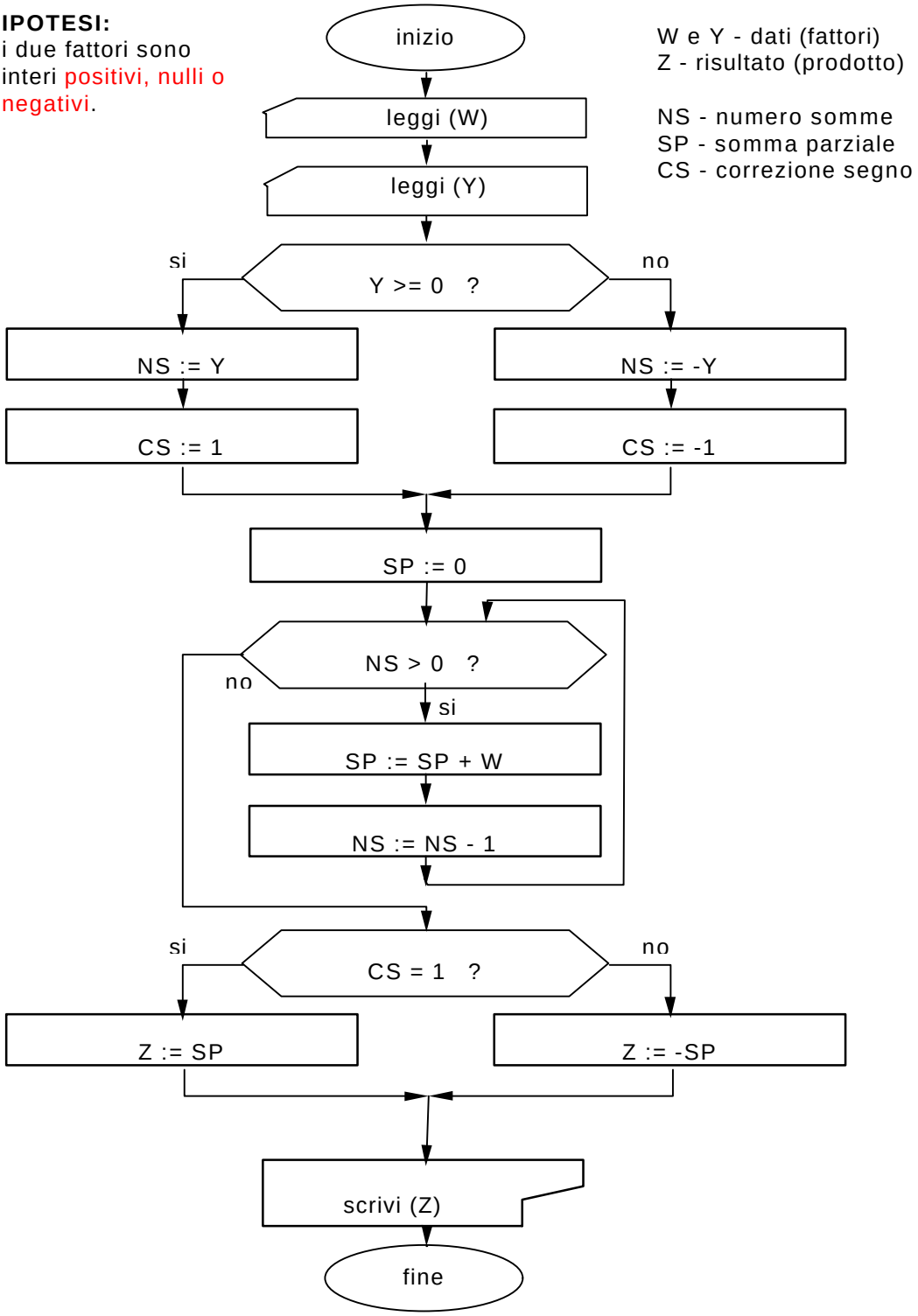
Si utilizza un costrutto di selezione doppia quando, in dipendenza del verificarsi o meno di una data condizione si devono eseguire in alternativa alcune operazioni o altre.

> Il valore della condizione dipenderà a sua volta dai valori di opportune variabili a cui sono stati precedentemente assegnati valori forniti in ingresso dall'utente oppure ottenuti come risultato del calcolo di espressioni.

Con gli schemi a blocchi il costrutto di selezione doppia viene realizzato con un blocco condizionale. Lungo uno dei due rami uscenti da tale blocco sono collocati i blocchi che descrivono le operazioni da eseguire in un caso, mentre lungo l'altro ramo sono collocati i blocchi relativi alle operazioni da eseguire nell'altro caso.

Esempio 4: prodotto di due numeri interi tramite somme ripetute

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



W e Y - dati (fattori)
Z - risultato (prodotto)

NS - numero somme
SP - somma parziale
CS - correzione segno

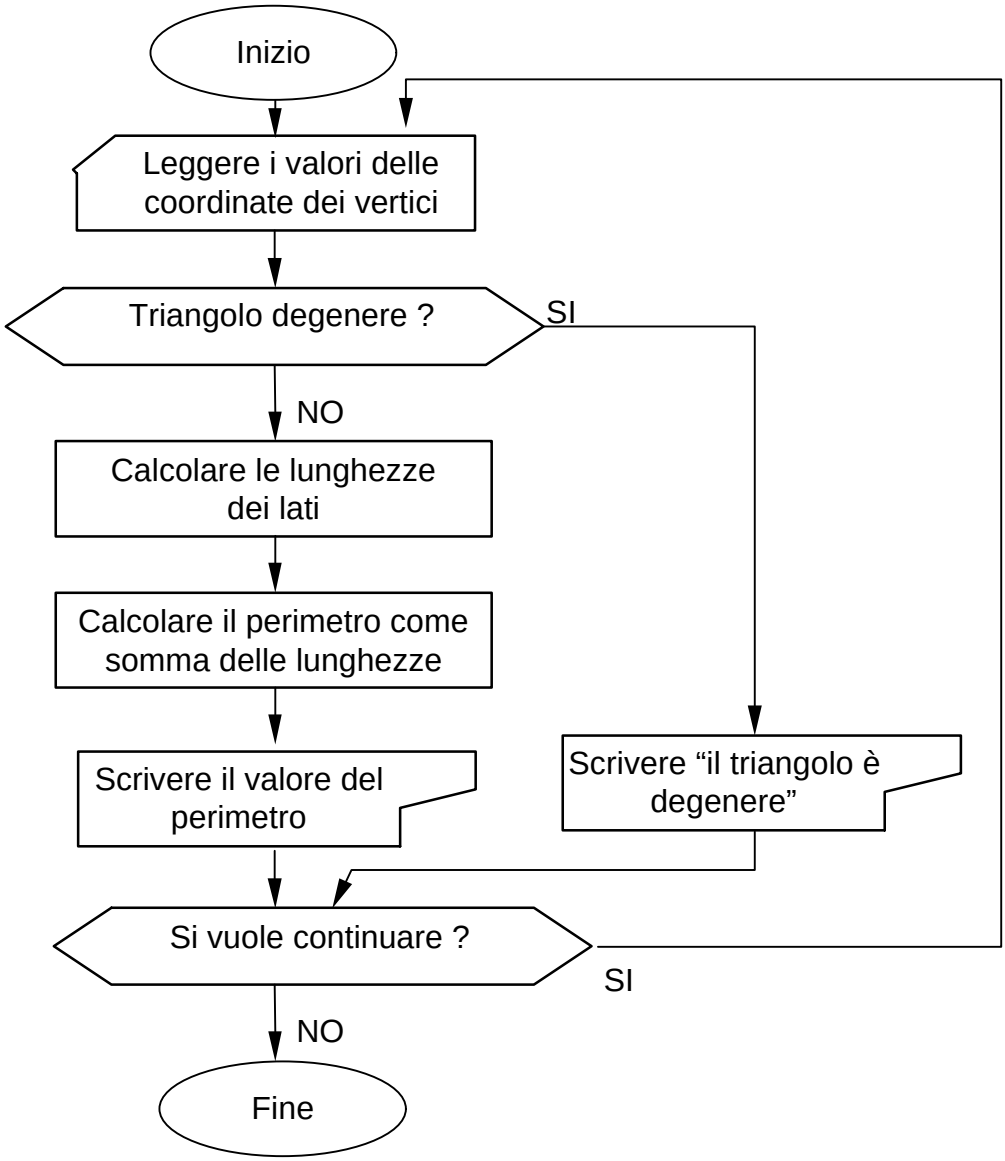
Codifica in C di esempio 4

```
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);
}
```

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

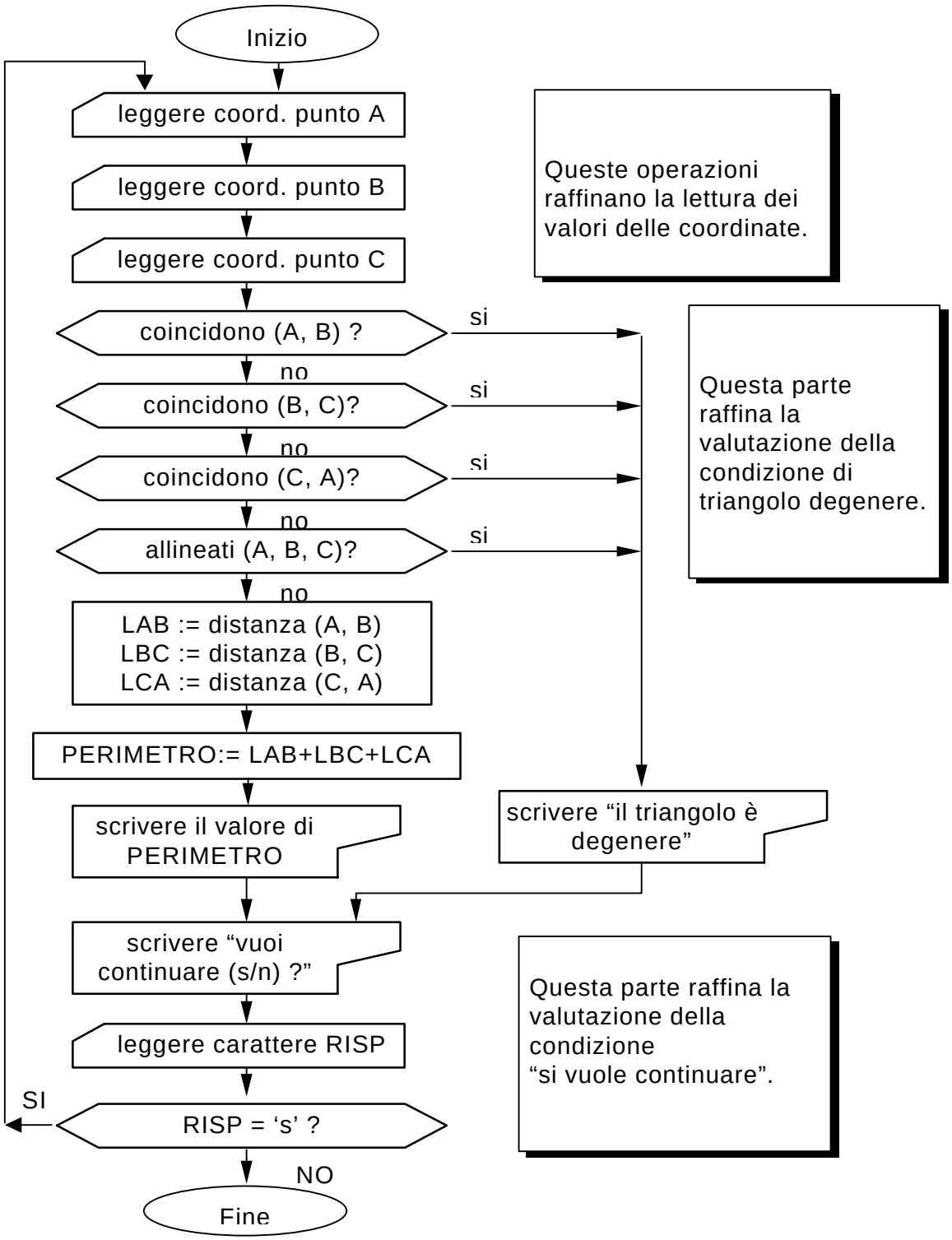
Uso dei costrutti di controllo del linguaggio C con le parole-chiave

- `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.

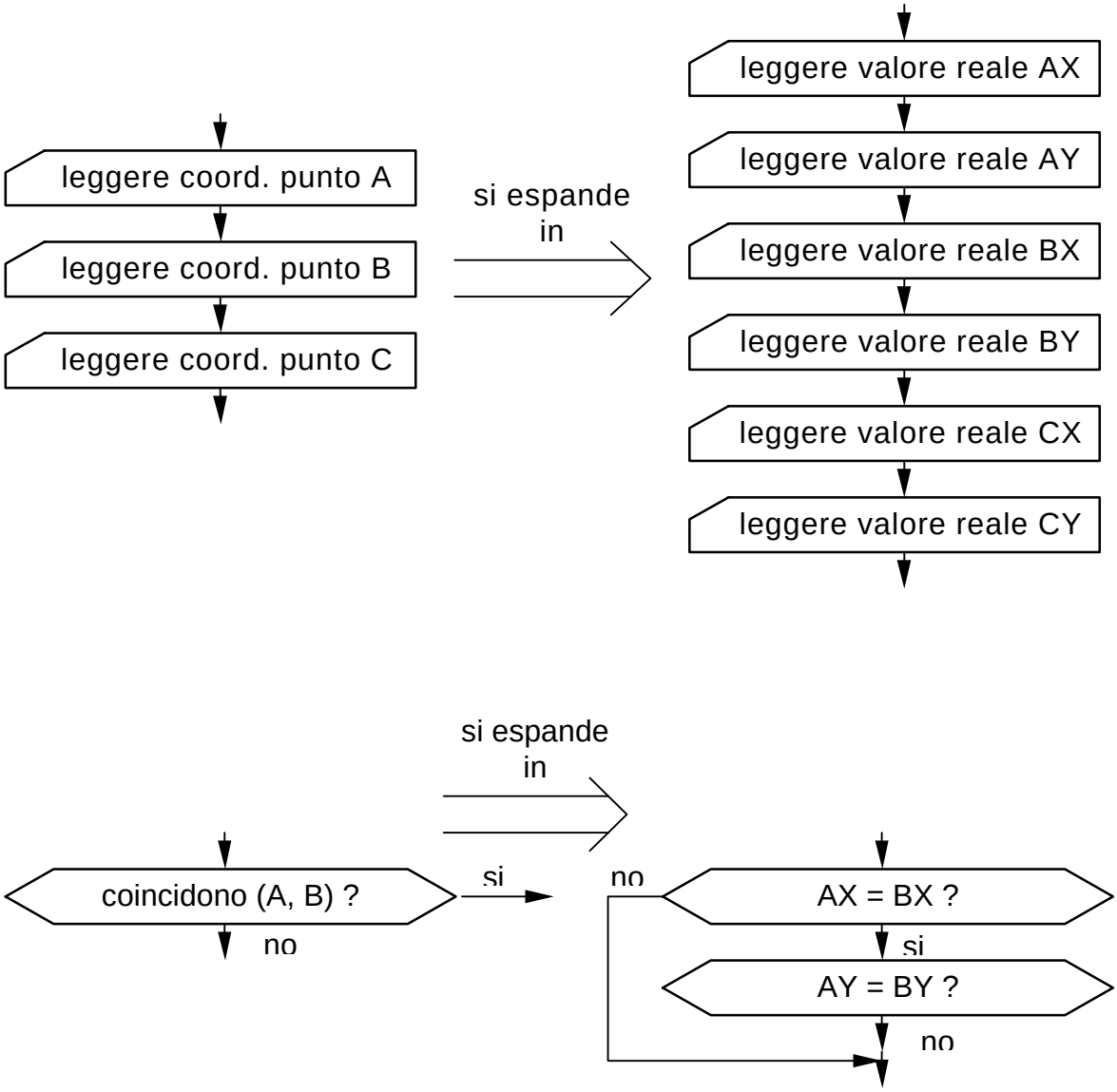
I sottoprogrammi sono molto utilizzati nella soluzione di problemi reali, dato che il loro impiego presenta diversi e importantissimi vantaggi.

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.

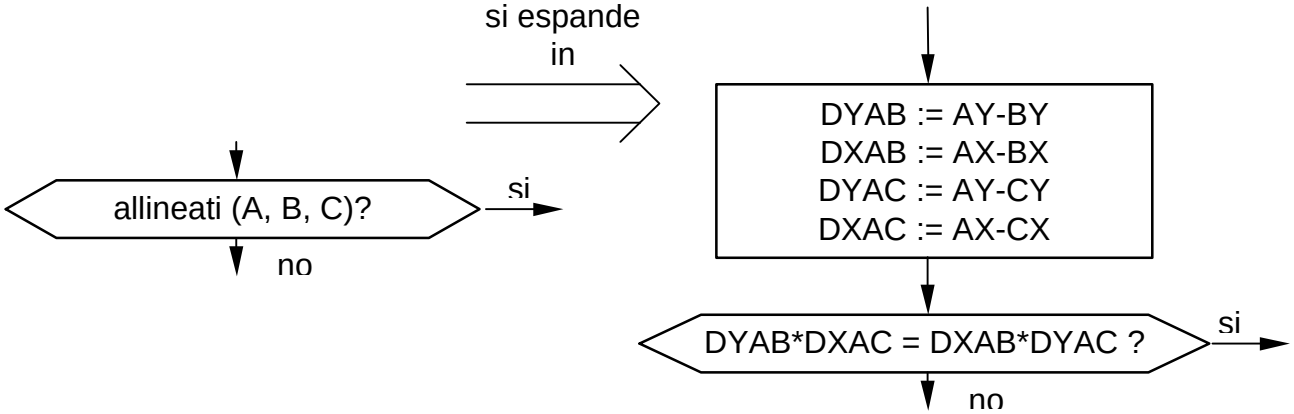
Si evitano ripetizioni. Spesso alcuni sottoproblemi devono essere affrontati più volte nell'ambito della soluzione di un problema principale. Affidando ad un sottoprogramma il compito di risolvere il sottoproblema si potrà richiamare il sottoprogramma tutte le volte che sia necessario, evitando di inserire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

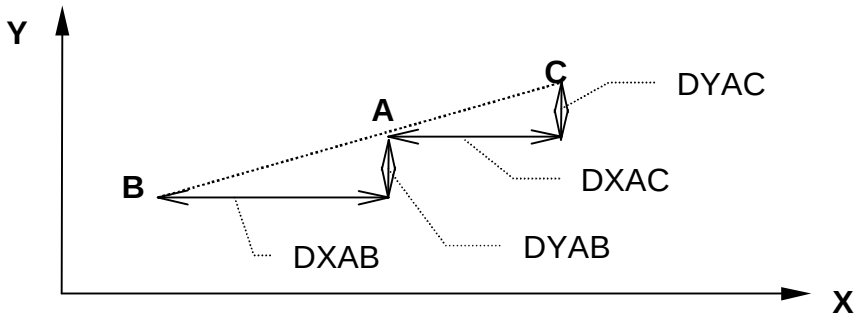
Sottoprogrammi «prefabbricati». Alcuni sottoproblemi di uso comune sono disponibili, già risolti da esperti programmatori, raccolti nelle cosiddette librerie di sottoprogrammi, evitando al programmatore di reinventare algoritmi già risolti.

Raffinamento 2: espansione delle direttive complesse



Raffinamento 2 - seguito



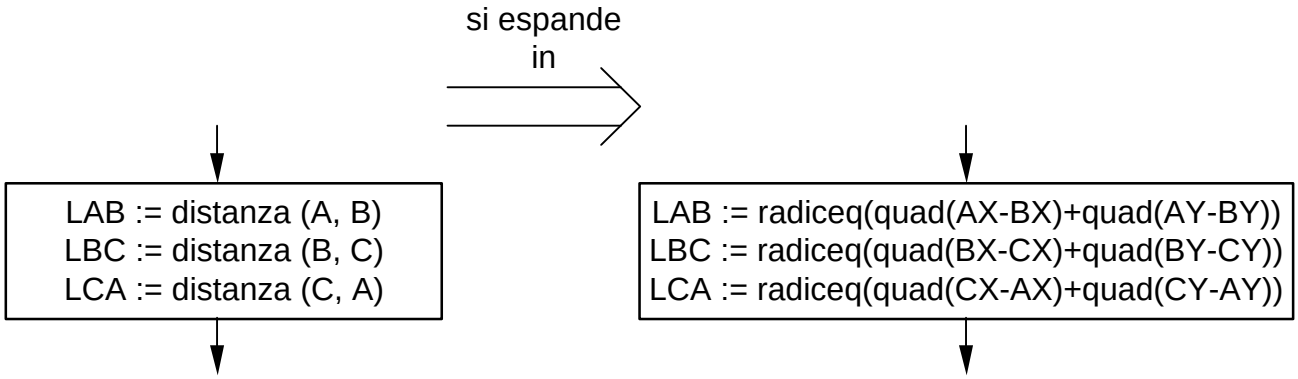


NOTA:

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 $\sqrt{N}$

Esempio 6

Si vuole leggere una sequenza di N valori interi in ingresso (con N qualsiasi) e trovare il valore massimo all'interno della sequenza e la sua posizione. Il valore e la posizione devono essere visualizzati in uscita.

variabili:

`N`	numero di elementi da acquisire
`elemento`	valore dell'elemento letto
`max`	valore del massimo
`posmax`	posizione del massimo all'interno della sequenza
`i`	*`indice corrente dell'elemento letto`*

Algoritmo:

1. acquisizione di N (e verifica dell'accettabilità)
2. acquisizione del primo valore
3. il valore è considerato il massimo attuale
4. da ripetersi finché non si sono acquisiti tutti gli N valori:
   acquisizione del prox valore: se è maggiore del massimo attuale
   aggiornamento del valore del max e della sua posizione
5. visualizzazione del valore del massimo e della sua posizione.

Esempio 6: pseudo-codice

```
main ()

{
int N;              numero di elementi da acquisire
int elemento;       valore dell'elemento letto
int max;            valore del massimo
int posmax;         posizione del massimo all'interno della sequenza
int i;              indice corrente relativo al numero di elementi già letti

leggi (N);
if (N è un valore accettabile)
  {
     leggi (elemento);
      i = 1; /* aggiorna l'indice corrente */
      considera questo elemento come massimo attuale

    /* ciclo acquisizione e ricerca massimo per i  rimanenti
valori */

    while (non si sono letti N valori)
    {
     leggi (elemento);
      i = i +1; /* aggiorna l'indice corrente */
      if (l'elemento letto è maggiore del massimo attuale)
         {
          considera questo elemento come massimo attuale
         }
    } /* fine del costrutto while */

    scrivi (max);
    scrivi (posmax);
  } /*fine ramo if eseguito se N è  valore accettabile*/

  else /* N non è un valore accettabile */
  {
    scrivi («il valore di N non è accettabile»)
  }/*fine ramo if eseguito se N non è  accettabile */
                                  }
```

Esempio 6: pseudo-codice

```
main ()
{

int N;            /* numero di elementi da acquisire */
int elemento;     /* valore dell'elemento letto */
int max;          /* valore del massimo */
int posmax;       /* posizione massimo nella sequenza */
int i;            /*  indice  corrente  relativo  al  numero  di
                  elementi già letti */

leggi (N);
if (N > 0 ) /*l'utente vuole inserire almeno un valore */
  {
     leggi (elemento);
      i = 1; /* aggiorna l'indice corrente */
      max = elemento;
      posmax = i;

    /* ciclo acquisizione e ricerca massimo per i  rimanenti
valori */

    while (i < N)
    {
     leggi (elemento);
      i = i +1; /* aggiorna l'indice corrente */
      if (elemento > max)
          {
           max = elemento;
           posmax = i;
          }
    } /* fine del costrutto while */

    scrivi (max);
    scrivi (posmax);
  } /* fine ramo if eseguito se N è valore accettabile */

  else /*N non è un valore accettabile */
  {
    scrivi («il valore di N non è accettabile»)
  }/* fine ramo if eseguito se N non è accettabile */
}
```

Esempio 7

Si vuole costruire la retta passante per due punti del piano X-Y.

Algoritmo:
1. acquisizione delle coordinate X e Y dei due punti.
2. calcolo dei parametri caratteristici dell'equazione della retta:
   - $x=k_x$
   - $y= k_y$
   - $y=mx+q$
3. visualizzazione dei parametri.

variabili:

`x1,x2,y1,y2`	coord. reali dei punti
`kx,ky, m, q`	param. della retta (reali)

Esempio 7: pseudo-codice

```
main ()
{
  float x1, x2, y1,y2;    /*coordinate dei punti */
  float kx;               /*parm retta parallela asse y*/
  float ky;               /*parm retta parallela asse x*/
  float m, q;             /*parm retta generica*/

  leggi (x1);
  leggi (x2);
  leggi (y1);
  leggi (y2);

  if (i due punti sono coincidenti)
    {
     scrivi («problema mal posto: punti coincidenti»);
    }

    else  /* esiste la retta */
    {
     if (ascisse coincidenti)
         {
           kx=x1;
         scrivi(«la retta e' parall. asse y e ......»);
         }
         else  /* ascisse non coincidenti */
         {
             if (ordinate coincidenti)
               {
               ky=y1;
               scrivi(«retta parallela  asse x e ......»);
               }
             else /*ascisse e ordinate non coincidenti */
               {
               calcolare m e q;
               scrivi(«i parametri della retta sono....»);
               } /*fine ramo retta generica */
         } /*fine ramo ascisse non coincidenti */
    } /*fine ramo esiste retta */
} /*fine del programma */
```

Esempio 7: pseudo-codice

```
main ()

{
  float x1, x2, y1,y2;    /*coordinate dei punti */
  float kx;               /*parm retta parallela asse y*/
  float ky;               /*parm retta parallela asse x*/
  float m, q;             /*parm retta generica*/

  leggi (x1);
  leggi (x2);
  leggi (y1);
  leggi (y2);

  if (i due punti sono coincidenti)
    {
     scrivi («problema mal posto: punti  coincidenti»);
    }

    else  /* esiste la retta */
    {
     if (x1==x2)
         {
           kx=x1;
         scrivi(«retta parallela asse y e ......»);
         }
         else  /* ascisse non coincidenti */
         {
             if (y1==y2)
               {
               ky=y1;
               scrivi(«retta parallela asse x e ......»);
               }
             else  /*ascisse e ordinate non coincidenti */
               {
               m=(y2-y1)/(x2-x1);
               calcolare q;
               scrivi(«i parametri della retta sono.....»);
               } /*fine ramo retta generica */
         } /*fine ramo ascisse non coincidenti */
    } /*fine ramo esiste retta */
} /*fine del programma */
```

Esempio 8 (esercitazione)

Si vuole scrivere un programma per la ricerca delle radici di una funzione polinomiale di grado *n* del tipo

$$y = \sum_{i=0}^{n} a_i x^n$$

in un intervallo compreso tra x_1 e x_2.
La ricerca della radice (supposta unica e esistente) avviene con il **metodo di bisezione**:

- si calcola il punto medio $x_m = (x_1+x_2)/2$ tra i punti x_1 e x_2;
- si valuta il segno del polinomio *y* in corrispondenza dei punti x_1, x_2 e x_m;
- si scarta l'intervallo in cui il segno del polinomio valutato agli estremi è uguale (nell'esempio in figura, si scarta l'intervallo tra x_m e x_2);
- si riapplica il procedimento all'intervallo i cui estremi hanno segno del polinomio diverso (nell'esempio in figura, si riapplica il procedimento all'intervallo tra x_1 e x_m);
- il procedimento continua fino a che l'ampiezza dell'intervallo considerato diventa minore di un valore di tolleranza *t*;
- la soluzione cercata è il punto medio dell'ultimo intervallo trovato.