

Gli algoritmi

- Analisi e programmazione
- Gli algoritmi
 - ➔ Proprietà ed esempi
 - ➔ Costanti e variabili, assegnazione, istruzioni, proposizioni e predicati
 - ➔ Vettori e matrici
 - ➔ I diagrammi a blocchi
 - ➔ Analisi strutturata
 - ➔ Gli algoritmi iterativi
 - ➔ La pseudocodifica
 - ➔ Gli algoritmi ricorsivi





Analisi e programmazione

Analisi e programmazione – 1

- Tramite un elaboratore si possono risolvere problemi di varia natura: emissione di certificati anagrafici, gestione dei c/c di un istituto di credito, prenotazioni ferroviarie...
- Il problema deve essere formulato in modo opportuno, perché sia possibile utilizzare un elaboratore per la sua soluzione
- Per **analisi e programmazione** si intende l'insieme delle attività preliminari atte a risolvere problemi utilizzando un elaboratore, dalla formulazione del problema fino alla predisposizione dell'elaboratore
 - **Scopo dell'analisi** ⇨ definire un **algoritmo**
 - **Scopo della programmazione** ⇨ definire un **programma**

Analisi e programmazione – 2

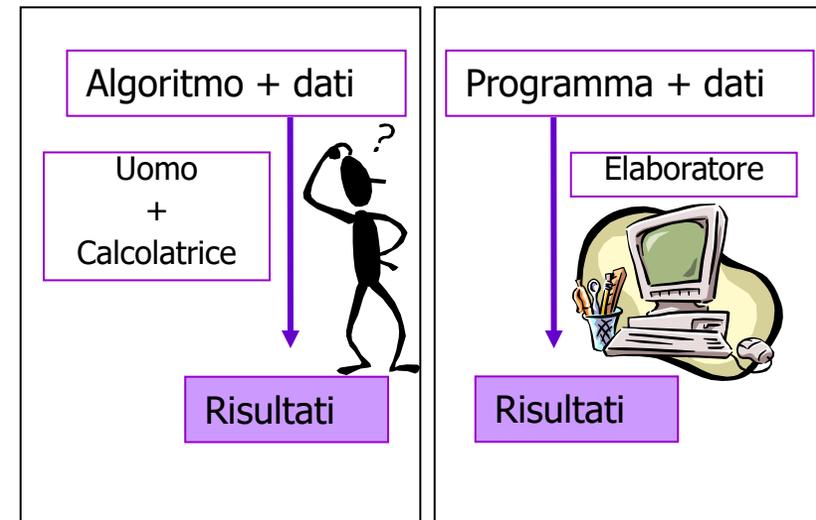
- **Algoritmo** — elenco finito di istruzioni, che specificano le operazioni eseguendo le quali si risolve una classe di problemi
 - Un particolare problema della classe viene risolto utilizzando l'apposito algoritmo sui dati che lo caratterizzano
 - **Un algoritmo non può essere eseguito direttamente dall'elaboratore**
- **Programma** — ricetta che traduce l'algoritmo ed è direttamente comprensibile, pertanto eseguibile, da parte di un elaboratore
- **Linguaggio di programmazione** — linguaggio rigoroso che permette la formalizzazione di un algoritmo in un programma

Analisi e programmazione – 3

- **Esempio**

Problema: Effettuare un accredito su un c/c bancario

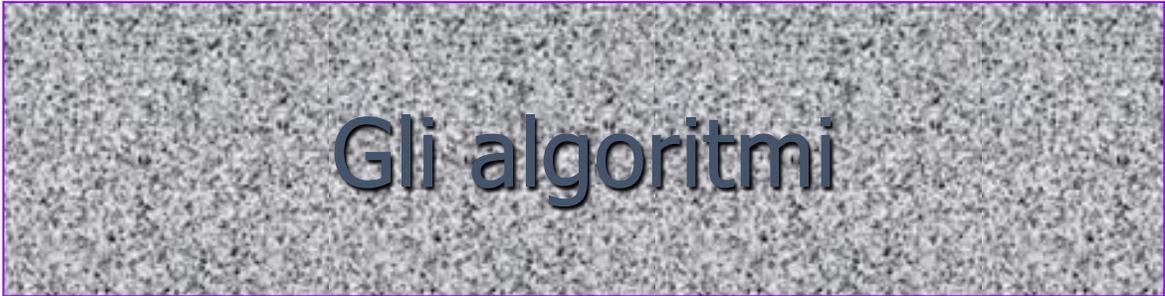
Soluzione: Utilizzare un programma che serva per predisporre il calcolatore all'accredito di una qualunque cifra su un qualunque c/c; cifra da accreditare e numero di c/c sono i dati caratteristici del problema



Analogie tra le azioni che devono essere eseguite da un operatore umano e, in modo automatico, tramite un elaboratore

Le fasi del procedimento di analisi e programmazione





Gli algoritmi

Definizione di algoritmo

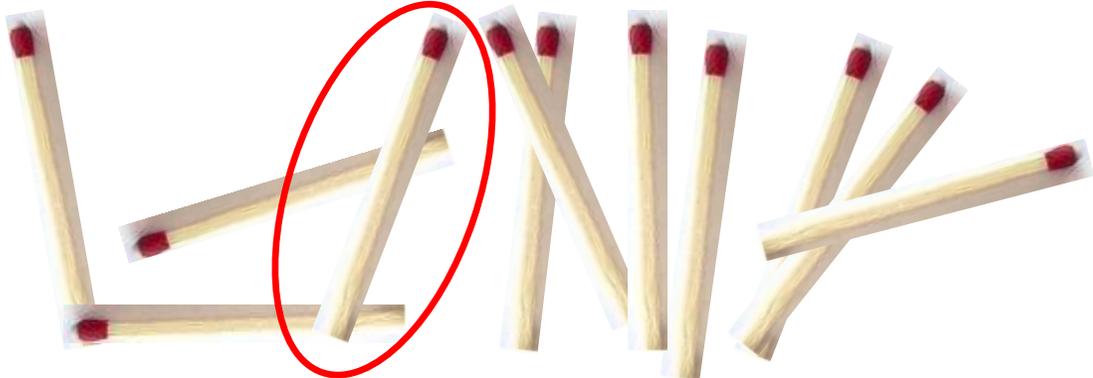
- **Algoritmo** deriva dal nome del matematico arabo *Al Khuwarizmi*, vissuto nel IX secolo d.C.
- Un algoritmo è una successione di **istruzioni** o **passi** che definiscono le operazioni da eseguire sui dati per ottenere i risultati; un algoritmo fornisce la soluzione ad una **classe di problemi**
- Lo **schema di esecuzione** di un algoritmo specifica che i passi devono essere eseguiti in sequenza, salvo diversa indicazione
- Ogni algoritmo è concepito per interagire con l'ambiente esterno per acquisire dati e comunicare messaggi o risultati; i dati su cui opera un'istruzione sono forniti dall'esterno o sono frutto di istruzioni eseguite in precedenza



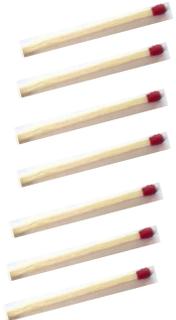
Esempio: Il gioco dell'undici

- **Problema:** Undici fiammiferi sono disposti su un tavolo: il primo giocatore (A) può raccogliere da 1 a 3 fiammiferi, il secondo (B) ne raccoglie a sua volta 1, 2 o 3; i giocatori alternano le loro mosse finché sul tavolo non ci sono più fiammiferi; il giocatore che è costretto a raccogliere l'ultimo fiammifero è il perdente
- **Algoritmo:** Strategia vincente per il giocatore A che gioca per primo
 - prima mossa: A raccoglie 2 fiammiferi
 - mosse successive: se B raccoglie k fiammiferi ($k \leq 3$), allora A raccoglie $4-k$ fiammiferi

Esempio: Il gioco dell'undici



Topolino perde!



Esempio: Ordinamento di un mazzo di carte

- ✦ **Problema:** Sia dato un mazzo da 40 carte da ordinare in modo che le cuori precedano le quadri, che a loro volta precedono fiori e picche; le carte di uno stesso seme sono ordinate dall'asso al re
- ✦ **Algoritmo:**
 - Si suddivida il mazzo in 4 mazzetti, ciascuno costituito da tutte le carte dello stesso seme
 - Si ordinino le carte di ciascun mazzetto dall'asso al re
 - Si prendano nell'ordine i mazzetti delle cuori, quadri, fiori e picche

Esempio: Ricerca in un mazzo di chiavi

- ✦ **Problema:** Si vuole ricercare, all'interno di un mazzo di chiavi, quella che apre un dato lucchetto
- ✦ **Algoritmo:**
 - 1) Si seleziona una chiave dal mazzo e la si marca con un pennarello
 - 2) Si tenta di aprire il lucchetto con la chiave appena marcata; se funziona, si va al passo 4)
 - 3) Altrimenti, si controlla la chiave successiva
 - i. Se non è marcata, la si marca e si torna al passo 2)
 - ii. Viceversa, si prende atto che nel mazzo non è presente la chiave che apre il lucchetto
 - 4) Fine della ricerca

Esempio: Radici delle equazioni di 2° grado

- **Problema:** Calcolo delle radici reali di $ax^2+bx+c=0$
- **Algoritmo:**
 - 1) Acquisire i coefficienti a, b, c
 - 2) Calcolare $\Delta = b^2 - 4ac$
 - 3) Se $\Delta < 0$ non esistono radici reali, eseguire l'istruzione 7)
 - 4) Se $\Delta = 0$, $x_1 = x_2 = -b/2a$, poi eseguire l'istruzione 6)
 - 5) $x_1 = (-b + \sqrt{\Delta})/2a$, $x_2 = (-b - \sqrt{\Delta})/2a$
 - 6) Comunicare i valori x_1, x_2
 - 7) Fine

Esempio: Calcolo del M.C.D. – 1

- **Problema:** Calcolare il M.C.D. di due interi a, b , con $a > b$
- **Algoritmo:** Formalizzato da Euclide nel 300 a.C., si basa sul fatto che ogni divisore comune ad a e b è anche divisore del resto r della divisione intera di a per b , quando $a > b$ e $r \neq 0$; se $r = 0$, b è il M.C.D.

$$\text{MCD}(a, b) = \text{MCD}(b, r), \text{ se } r \neq 0$$

$$\text{MCD}(a, b) = b, \text{ se } r = 0$$

- **Nota**
L'algoritmo garantisce la determinazione del M.C.D. senza il calcolo di tutti i divisori di a e b

Esempio: Calcolo del M.C.D. – 2

- 1) Acquisire i valori di a e b
- 2) Se $b > a$, scambiare i valori di a e b
- 3) Calcolare il resto r della divisione intera di a per b
- 4) Se $r = 0$, $MCD(a,b) = b$; comunicare il risultato all'esterno; eseguire l'istruzione 6)
- 5) Se $r \neq 0$, sostituire il valore di a con il valore di b ed il valore di b con il valore di r; tornare al passo 3)
- 6) Fine

Proprietà degli algoritmi

- Affinché una “*ricetta*”, un elenco di istruzioni, possa essere considerato un algoritmo, devono essere soddisfatti i seguenti requisiti:
 - **Finitezza**: ogni algoritmo deve essere finito, cioè ogni singola istruzione deve poter essere eseguita in tempo finito ed un numero finito di volte
 - **Generalità**: ogni algoritmo deve fornire la soluzione per una classe di problemi; deve pertanto essere applicabile a qualsiasi insieme di dati appartenenti all'**insieme di definizione** o **dominio dell'algoritmo** e deve produrre risultati che appartengano all'**insieme di arrivo** o **codominio**
 - **Non ambiguità**: devono essere definiti in modo univoco i passi successivi da eseguire; devono essere evitati paradossi, contraddizioni ed ambiguità; il significato di ogni istruzione deve essere univoco per chiunque esegua l'algoritmo

Algoritmi

- Un algoritmo deve poter essere eseguito da chiunque, senza che l'esecutore sia stato necessariamente coinvolto nell'analisi del problema o nella descrizione dell'algoritmo
- Gli algoritmi devono essere formalizzati per mezzo di appositi linguaggi, dotati di strutture linguistiche che garantiscano precisione e sintesi
- I linguaggi naturali non soddisfano questi requisiti, infatti...
 - ...sono **ambigui**: la stessa parola può assumere significati diversi in contesti differenti (*pesca* è un frutto o un'attività sportiva)
 - ...sono **ridondanti**: lo stesso concetto può essere espresso in molti modi diversi, ad esempio "somma 2 a 3", "calcola 2+3", "esegui l'addizione tra 2 e 3"

Costanti e variabili – 1

- I dati su cui opera un algoritmo sono **costanti** e **variabili**
 - Un dato è costante quando il suo valore non può essere aggiornato durante l'esecuzione dell'algoritmo o per esecuzioni successive
 - Una variabile è una coppia $\langle \text{nome}, \text{valore} \rangle$: può essere immaginata come una scatola sulla quale è scritto un nome e che può contenere un valore



Rappresentazione di una variabile

Costanti e variabili – 2

- Il valore di una variabile deve appartenere all'**insieme di definizione**, su cui si opera mediante regole opportune, specifiche dell'insieme
- Data una variabile $\langle x, v \rangle$, x è il **nome** della variabile e v è il suo **valore attuale**; le variabili sono indeterminate in fase di definizione dell'algoritmo, ma corrispondono a valori specifici durante ogni esecuzione
- **Esempio:** Nell'algoritmo di risoluzione delle equazioni di 2° grado, a, b, c non corrispondono a nessun valore finché non si esegue l'algoritmo per trovare le soluzioni di una data equazione, ad esempio $x^2 - 9x - 4 = 0$; in fase di esecuzione, $a=1$, $b=-9$, $c=-4$; nell'istruzione $\Delta = b^2 - 4ac$, Δ è la variabile che contiene il valore del discriminante

Assegnazione – 1

- L'istruzione di **assegnazione** definisce il valore attuale di una variabile, che resta inalterato fino all'assegnazione successiva
- L'assegnazione si rappresenta con il simbolo " \leftarrow ":
nome di variabile \leftarrow espressione
che si legge *assegna alla variabile "nome di variabile" il valore di "espressione"*; l'espressione a destra di \leftarrow è costituita da variabili, costanti e operatori
- L'assegnazione viene così eseguita:
 - a) si valuta l'espressione a destra di \leftarrow , sostituendo ai nomi di variabile i loro valori attuali; il risultato deve appartenere all'insieme di definizione della variabile a sinistra di \leftarrow
 - b) il valore calcolato diventa il nuovo valore della variabile il cui nome appare a sinistra di \leftarrow

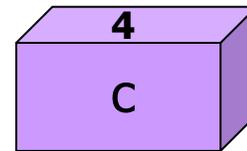
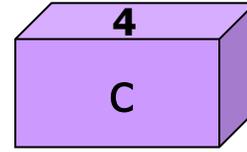
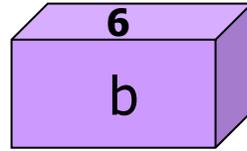
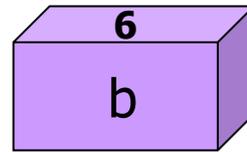
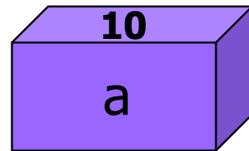
Assegnazione – 2

- I nomi delle variabili possono essere scelti in modo arbitrario, ma è opportuno selezionare nomi significativi del contenuto della variabile
- È necessario rispettare la **regola dell'ordinamento**:
Quando una variabile appare a destra di ← in una assegnazione deve essere già istanziata
cioè le deve essere già stato assegnato un valore

Assegnazione – 3

- Esempi

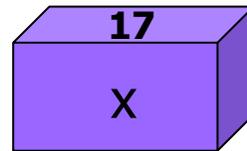
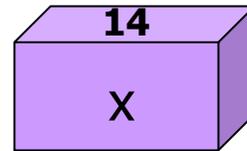
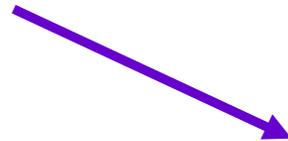
$a \leftarrow b+c$



Prima dell'assegnazione

Dopo l'assegnazione

$x \leftarrow x+3$



Prima dell'assegnazione

Dopo l'assegnazione

Le istruzioni – 1

- **Istruzioni operative**, che producono risultati
- **Istruzioni di controllo**, che controllano il verificarsi di condizioni specificate e, in base al risultato del controllo, determinano il flusso di istruzioni da eseguire
- **Istruzioni di salto**, che alterano il normale flusso di esecuzione sequenziale delle istruzioni di un algoritmo, specificando quale sia la successiva istruzione da eseguire
 - nelle istruzioni di **salto condizionato**, l'effettiva esecuzione del salto è legata al verificarsi di una condizione specificata
 - l'istruzione di **salto incondizionato** produce sempre un salto
- **Istruzioni di ingresso/uscita**, che specificano come debba essere effettuata una trasmissione di dati o messaggi fra l'algoritmo e l'ambiente esterno
- **Istruzioni di inizio/fine esecuzione**, che indicano l'inizio/la fine dell'algoritmo

Le istruzioni – 2

- **Esempio: Calcolo delle radici di equazioni di 2° grado**
 - a) "acquisire i coefficienti a , b , c " è un'istruzione di lettura (ingresso)
 - b) "calcolare $\Delta=b^2-4ac$ " è un'istruzione operativa
 - c) "se $\Delta=0$, $x_1=x_2=-b/2a$ " è un'istruzione di controllo: l'istruzione di assegnazione $x_1=x_2=-b/2a$ viene eseguita solo se $\Delta=0$
 - d) "comunicare i valori x_1 , x_2 " è un'istruzione di scrittura (uscita)
 - e) "eseguire l'istruzione 6)" è un'istruzione di salto incondizionato
 - f) "se $\Delta<0$ eseguire l'istruzione 7)" è un'istruzione di salto condizionato, perché l'istruzione 7) è la prossima istruzione da eseguire solo se $\Delta<0$

Proposizioni e predicati – 1

- # Una **proposizione** è un costrutto linguistico del quale si può asserire o negare la veridicità
- # **Esempi**
 - 1) *“Roma è la capitale della Gran Bretagna”* **falsa**
 - 2) *“3 è un numero intero”* **vera**
- # Il **valore di verità** di una proposizione è il suo essere vera o falsa
- # Una proposizione è un **predicato** se il suo valore di verità dipende dall'istanziamento di alcune variabili
- # **Esempi**
 - 1) *“la variabile età è minore di 30”*
 - 2) *“la variabile base è maggiore della variabile altezza”*

Proposizioni e predicati – 2

- La **valutazione di un predicato** è l'operazione che permette di determinare se il predicato è vero o falso, sostituendo alle variabili i loro valori attuali
- I valori **vero** e **falso** sono detti **valori logici** o **booleani**
- Proposizioni e predicati possono essere espressi concisamente per mezzo degli **operatori relazionali**:

= (uguale)

≠ (diverso)

> (maggiore)

< (minore)

≥ (maggiore o uguale)

≤ (minore o uguale)

- I predicati che contengono un solo operatore relazionale sono detti **semplici**

Proposizioni e predicati – 3

- Dato un predicato p , il predicato **not** p , detto **opposto** o **negazione logica** di p , ha i valori di verità opposti rispetto a p
- Dati due predicati p e q , la **congiunzione logica** p and q è un predicato vero solo quando p e q sono entrambi veri, e falso in tutti gli altri casi
- Dati due predicati p e q , la **disgiunzione logica** p or q è un predicato falso solo quando p e q sono entrambi falsi, e vero in tutti gli altri casi
- I predicati nei quali compare almeno un operatore logico, **not**, **and**, **or**, sono detti **composti**
- La **tavola di verità** di un predicato composto specifica il valore del predicato per ognuna delle possibili combinazioni dei suoi argomenti

Proposizioni e predicati – 4

‡ Esempio

not (base > altezza)

è vero solo quando il valore di base è minore o uguale del valore di altezza

età > 30 **and** età < 50

è vero solo quando il valore di età è compreso tra 30 e 50 (esclusi)

base > altezza **or** base > 100

è vero quando il valore di base è maggiore del valore di altezza, o quando il valore di base è maggiore di 100, o quando entrambe le condizioni sono verificate

Vettori e matrici – 1

- Le variabili definite come coppie $\langle nome, valore \rangle$ sono dette variabili **scalari**
- Una coppia $\langle nome, insieme\ di\ valori \rangle$ è una variabile **vettore** o **array** e può essere immaginata come un contenitore diviso in scomparti; ciascuno scomparto contiene un valore, detto **elemento** o **componente** del vettore
- Ciascuna componente è individuata dal nome del vettore, seguito dal relativo numero progressivo, racchiuso fra parentesi tonde: l'**indice** del vettore
- La **dimensione** di un vettore è il numero dei suoi elementi
- I vettori sono particolarmente utili per collezionare dati fra loro correlati, sui quali devono essere effettuate le stesse operazioni

Vettori e matrici – 2



Variabile vettoriale V , costituita dai 4 elementi $V(1)$, $V(2)$, $V(3)$, $V(4)$

- L'utilizzo di variabili vettoriali, in un algoritmo, presuppone la dichiarazione esplicita della loro dimensione
- La dimensione del vettore costituisce un limite invalicabile per la selezione delle componenti del vettore
- **Esempio:** $V(100)$ asserisce che il vettore V è costituito da 100 elementi; possono essere selezionati $V(12)$, $V(57)$, $V(89)$, ma non $V(121)$ o $V(763)$, che non esistono

Vettori e matrici – 3

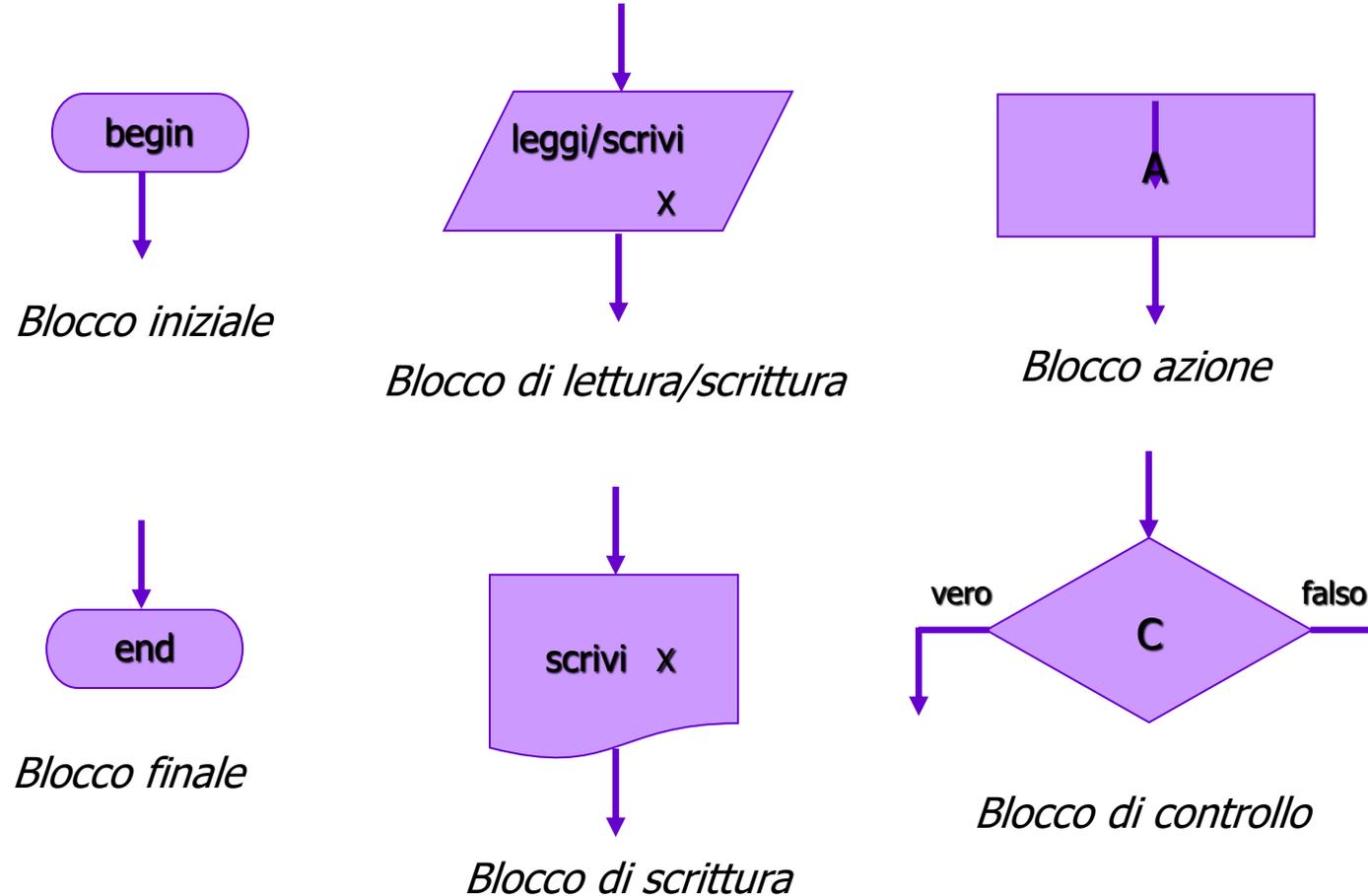
- Il concetto di **matrice** è un'estensione del concetto di vettore
- Una matrice è costituita da un insieme di valori, ciascuno dei quali viene individuato per mezzo della sua posizione, espressa da più indici
- Ad esempio, se una matrice M ha due dimensioni, i suoi elementi sono disposti su righe e colonne ed ogni suo elemento $M(i,j)$ è individuato da due indici, con i indice di **riga** e j indice di **colonna**

$$M = \begin{pmatrix} m_{11} & m_{12} & \dots & m_{1n} \\ \dots & \dots & \dots & \dots \\ m_{q1} & m_{q2} & \dots & m_{qn} \end{pmatrix}$$

I diagrammi a blocchi – 1

- Il linguaggio dei **diagrammi a blocchi** è un possibile formalismo per la descrizione di algoritmi
- Il diagramma a blocchi, o *flowchart*, è una rappresentazione grafica dell'algoritmo
- Un diagramma a blocchi descrive il **flusso** delle operazioni da eseguire per realizzare la trasformazione, definita nell'algoritmo, dai dati iniziali ai risultati
- Ogni istruzione dell'algoritmo viene rappresentata all'interno di un **blocco elementare**, la cui forma grafica è determinata dal tipo di istruzione
- I blocchi sono collegati tra loro da **linee di flusso**, munite di frecce, che indicano il susseguirsi di azioni elementari

I diagrammi a blocchi – 2



Blocchi elementari

I diagrammi a blocchi – 3

- Un **diagramma a blocchi** è un insieme di blocchi elementari composto da:
 - a) un blocco iniziale
 - b) un blocco finale
 - c) un numero finito n ($n \geq 1$) di blocchi di azione e/o di blocchi di lettura/scrittura
 - d) un numero finito m ($m \geq 0$) di blocchi di controllo

I diagrammi a blocchi – 4

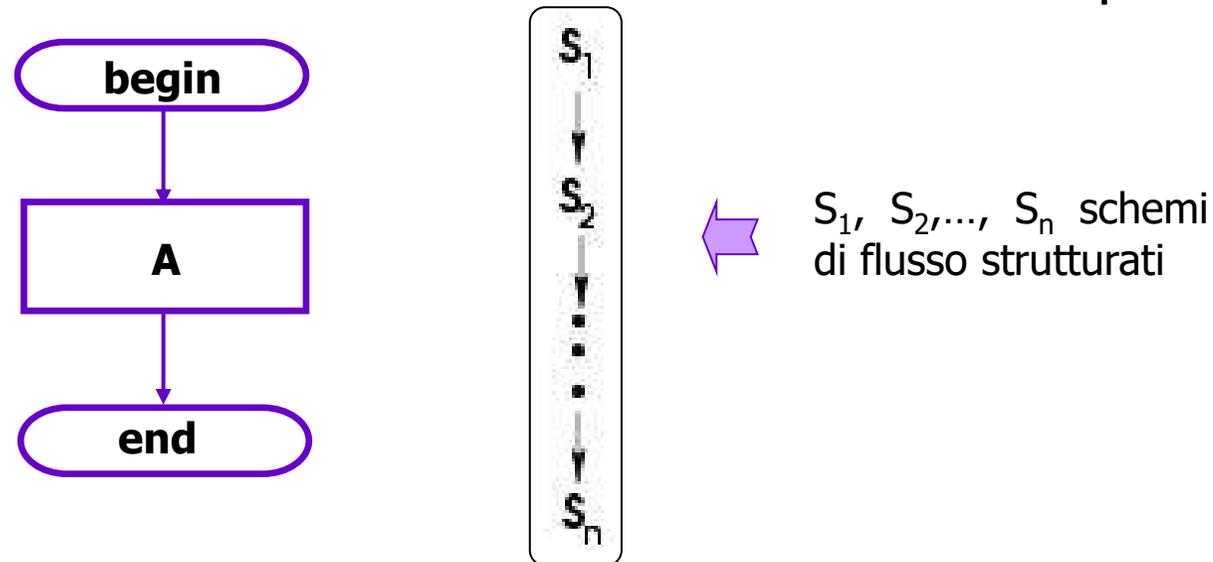
- L'insieme dei blocchi elementari che descrivono un algoritmo deve soddisfare le seguenti condizioni:
 - ciascun blocco di azione o di lettura/scrittura ha una sola freccia entrante ed una sola freccia uscente
 - ciascun blocco di controllo ha una sola freccia entrante e due frecce uscenti
 - ciascuna freccia entra in un blocco oppure si innesta in un'altra freccia
 - ciascun blocco è **raggiungibile** dal blocco iniziale
 - il blocco finale è **raggiungibile** da qualsiasi altro blocco
- Un blocco B è **raggiungibile** a partire da un blocco A se esiste una sequenza di blocchi X_1, X_2, \dots, X_n , tali che $A=X_1$, $B=X_n$, e $\forall X_i$, $i=1, \dots, n-1$, X_i è connesso con una freccia a X_{i+1}

Analisi strutturata – 1

- I programmatori inesperti tendono ad “aggrovigliare” il programma introducendo numerosi salti privi di regole (*spaghetti programming*)
- L'**analisi strutturata** favorisce, viceversa, la descrizione di algoritmi facilmente documentabili e comprensibili
- I blocchi di un diagramma a blocchi strutturato sono collegati secondo i seguenti schemi di flusso:
 - ◆ **Schema di sequenza** – più schemi di flusso sono eseguiti in sequenza
 - ◆ **Schema di selezione** – un blocco di controllo subordina l'esecuzione di due possibili schemi di flusso al verificarsi di una condizione
 - ◆ **Schema di iterazione** – si itera l'esecuzione di un dato schema di flusso

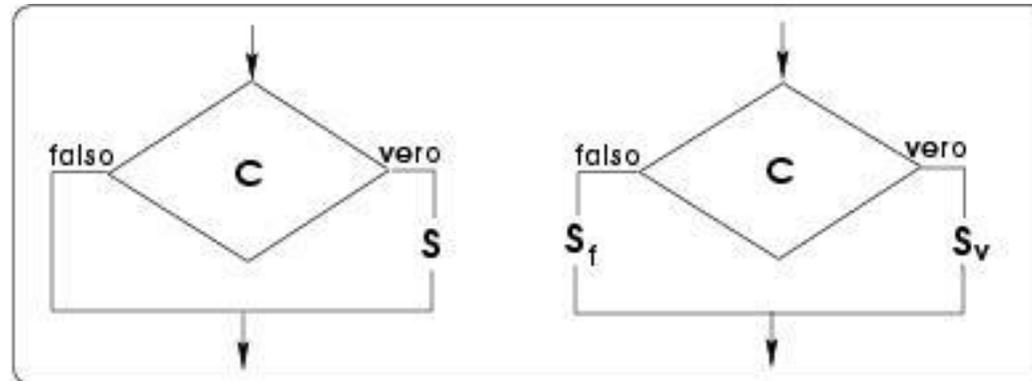
Analisi strutturata – 2

- Ovvero: un **diagramma a blocchi strutturato** è un diagramma a blocchi nel quale gli schemi di flusso sono **strutturati**
- Uno schema di flusso è strutturato quando soddisfa una delle seguenti proprietà...
 - 1) ...è uno schema elementare o uno schema di sequenza



Analisi strutturata – 3

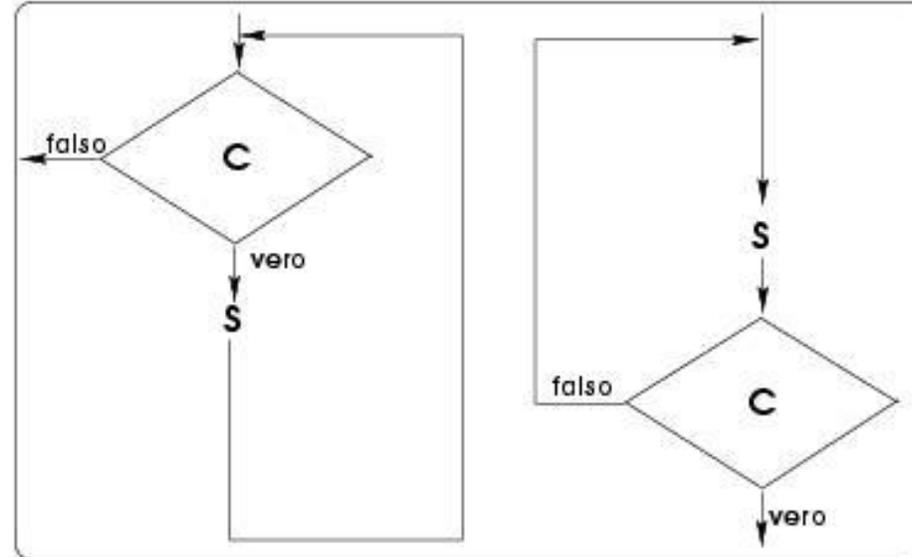
2) ...è uno schema di selezione



- Nel primo caso, lo schema S viene eseguito solo se la condizione C è vera; se C è falsa, non viene eseguita alcuna azione
- Nel secondo caso, viene eseguito solo uno dei due schemi S_v o S_f in dipendenza del valore di verità della condizione

Analisi strutturata – 4

3) ...è uno schema di iterazione



- Nel primo caso, S può non venire mai eseguito, se la condizione C è subito falsa; nel secondo caso, S viene eseguito almeno una volta
- Quando lo schema S viene eseguito finché la condizione C si mantiene vera si parla di **iterazione per vero**; si ha un'**iterazione per falso** quando S viene eseguito finché C è falsa

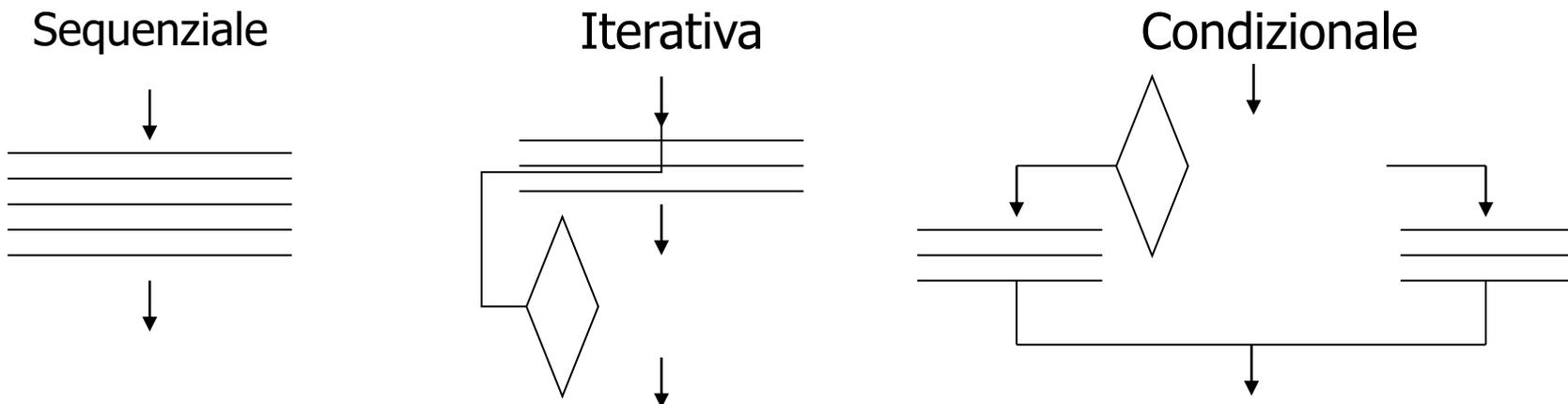
Analisi strutturata – 5

- ‡ Gli schemi di flusso sono **aperti** quando consentono una sola esecuzione di una sequenza di blocchi elementari, sono **chiusi** quando permettono più di un'esecuzione della sequenza di blocchi
- ‡ Gli schemi di sequenza e di selezione sono aperti, lo schema di iterazione è chiuso
- ‡ **Ogni diagramma a blocchi non strutturato è trasformabile in un diagramma a blocchi strutturato equivalente**
- ‡ Due diagrammi a blocchi sono **equivalenti** se, operando sugli stessi dati, producono gli stessi risultati
- ‡ L'uso dell'analisi strutturata garantisce:
 - facilità di comprensione e modifica dei diagrammi a blocchi
 - maggiore uniformità nella descrizione degli algoritmi

Analisi strutturata – 6

■ Inoltre...

- È stato dimostrato (teorema fondamentale della programmazione di Bohm–Jacopini, 1966) che ogni programma può essere codificato riferendosi esclusivamente ad un algoritmo strutturato e quindi attenendosi alle tre strutture fondamentali:

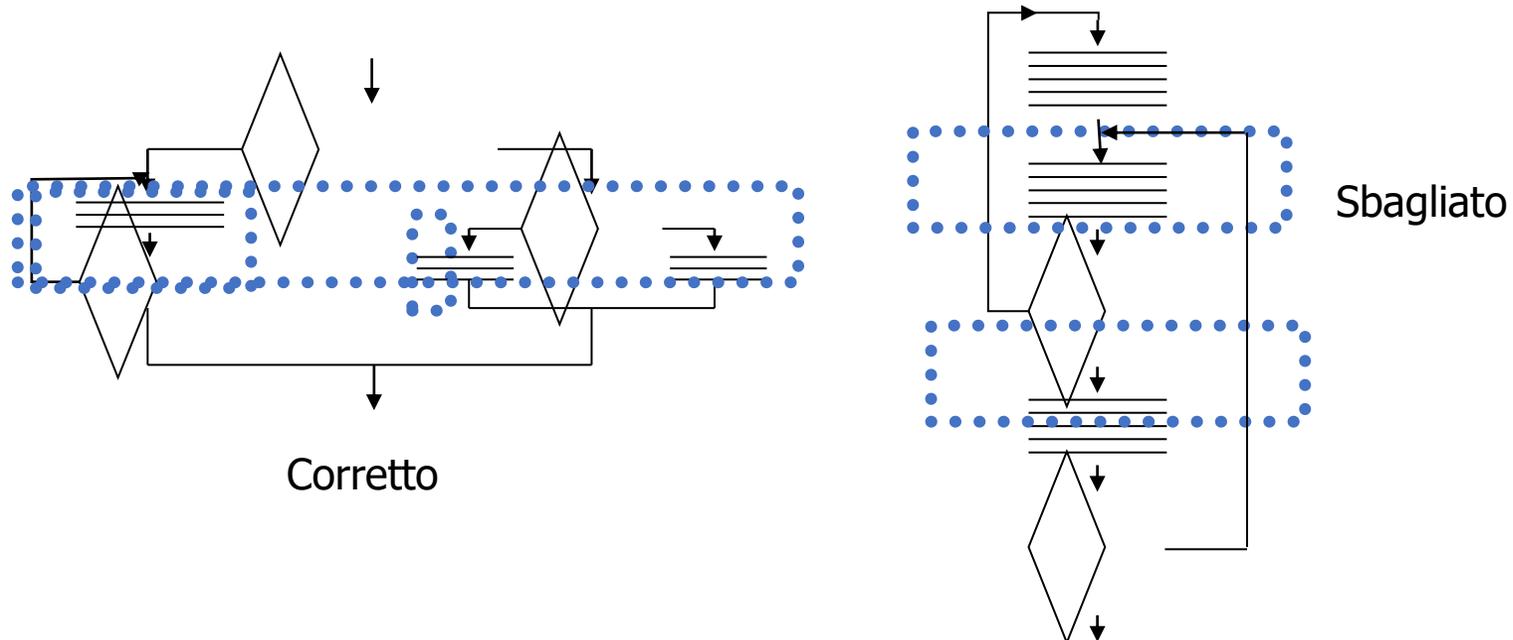


Analisi strutturata – 7

- Il teorema di Bohm–Jacopini ha un interesse soprattutto teorico, in quanto i linguaggi di programmazione tendono a dotarsi di più tipi di istruzioni, non sempre “rispettose” del teorema, ma utili per la realizzazione di programmi di facile scrittura e comprensione
- Il suo valore consiste nella capacità di fornire indicazioni generali per le attività di progettazione di nuovi linguaggi e di strategie di programmazione
- In effetti, esso ha contribuito alla critica dell’uso sconsiderato delle istruzioni *go to* e alla definizione delle linee guida della programmazione strutturata, sviluppate negli anni `70

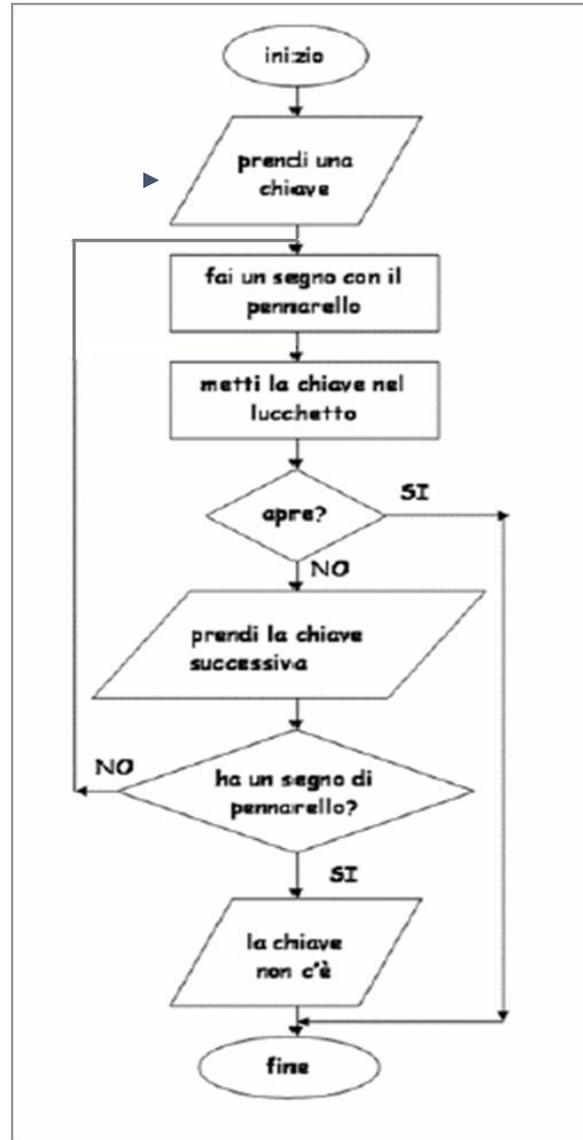
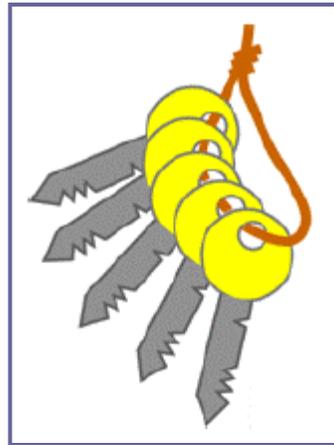
Analisi strutturata – 8

- In un diagramma strutturato non apparirà mai una istruzione di salto incondizionato
- I tre schemi fondamentali possono essere *concatenati*, uno di seguito all'altro, o *nidificati*, uno dentro l'altro; non possono in nessun caso essere "intrecciati" o "accavallati"



Esempio

- Diagramma a blocchi per la selezione, in un mazzo di chiavi, di quella che apre un lucchetto

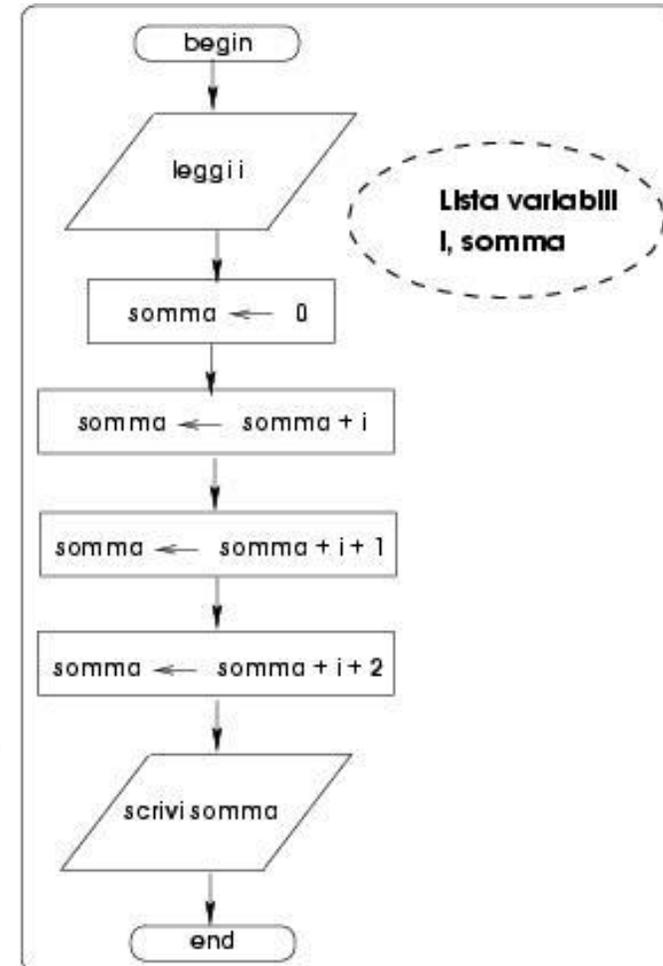


Esercizi

- Scrivere un algoritmo, e rappresentarlo tramite diagramma a blocchi, per la soluzione dei seguenti problemi:
 - calcolare l'area del triangolo
 - trovare il max di due numeri
 - moltiplicare due numeri (usando solo l'operazione di somma)
- Formalizzare, tramite diagramma a blocchi, l'algoritmo per...
 - ...calcolare le radici reali di equazioni di 2° grado
 - ...calcolare il M.C.D. di due numeri con il metodo di Euclide

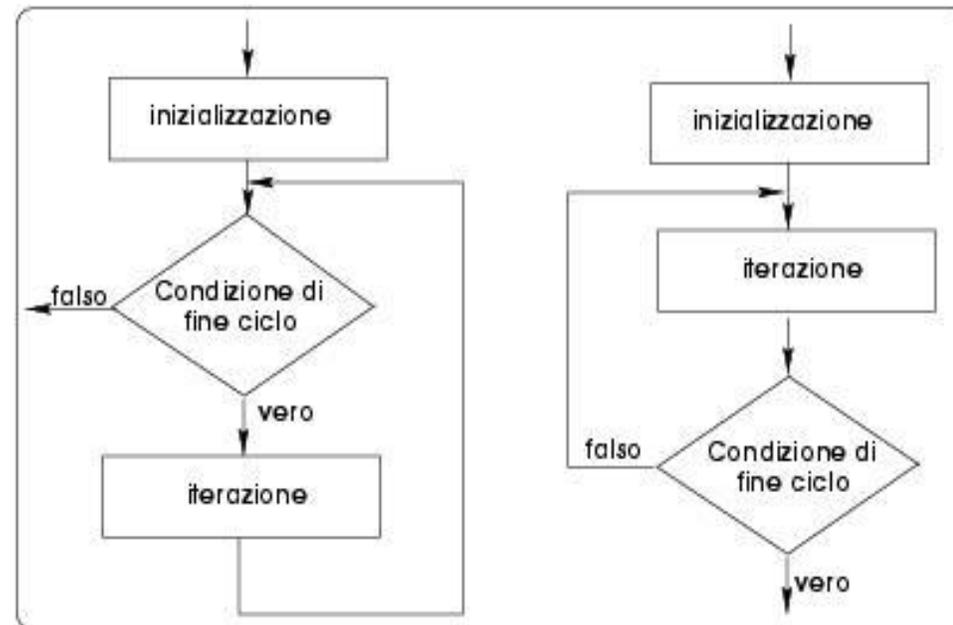
Gli algoritmi iterativi – 1

- **Problema:** Calcolare la somma di tre interi consecutivi
- **Note:**
 - ◆ La variabile **somma** è un contenitore di somme parziali, finché non si ottiene la somma totale richiesta
 - ◆ La soluzione del problema viene raggiunta eseguendo azioni simili per un numero opportuno di volte



Gli algoritmi iterativi – 2

- Il **ciclo** o *loop* è uno schema di flusso per descrivere, in modo conciso, situazioni in cui un gruppo di operazioni deve essere ripetuto più volte
- ‡ La **condizione di fine ciclo** viene verificata ogni volta che si esegue il ciclo; se la condizione assume valore vero (falso), le istruzioni vengono reiterate, altrimenti si **esce dal ciclo**
- ‡ La condizione di fine ciclo può essere verificata prima o dopo l'esecuzione dell'iterazione
- ‡ Le **istruzioni di inizializzazione** assegnano valori iniziali ad alcune variabili (almeno a quella che controlla la condizione di fine ciclo)

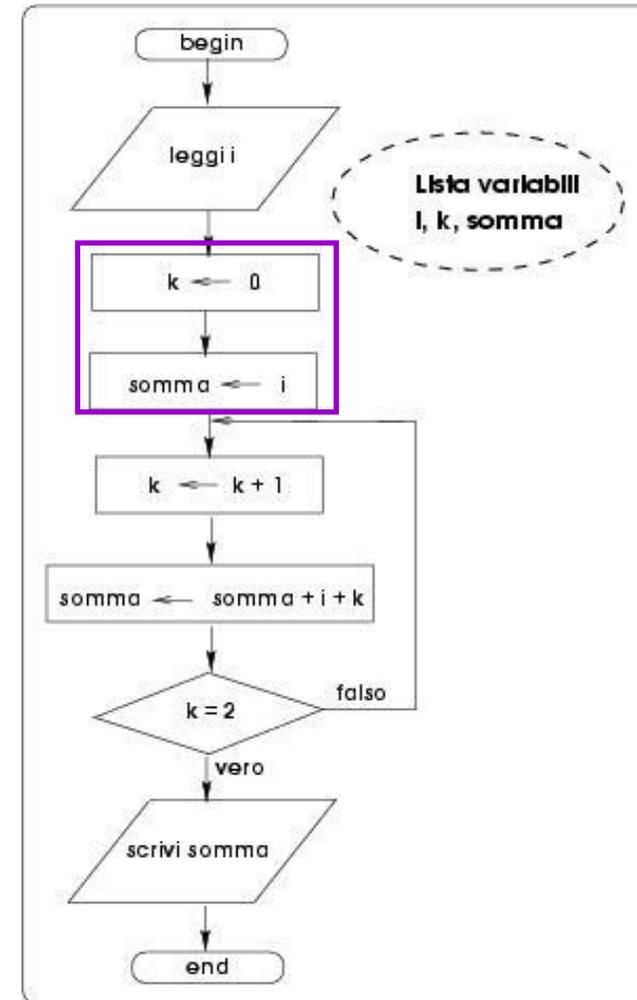
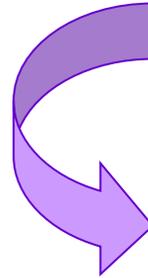


Ciclo con controllo in testa

Ciclo con controllo in coda

Gli algoritmi iterativi – 3

- **Problema:** Calcolare la somma di tre interi consecutivi
- **Note:**
 - ◆ La fase di inizializzazione riguarda la somma e l'indice del ciclo
 - ◆ Il controllo di fine ciclo viene effettuato in coda



Gli algoritmi iterativi – 4

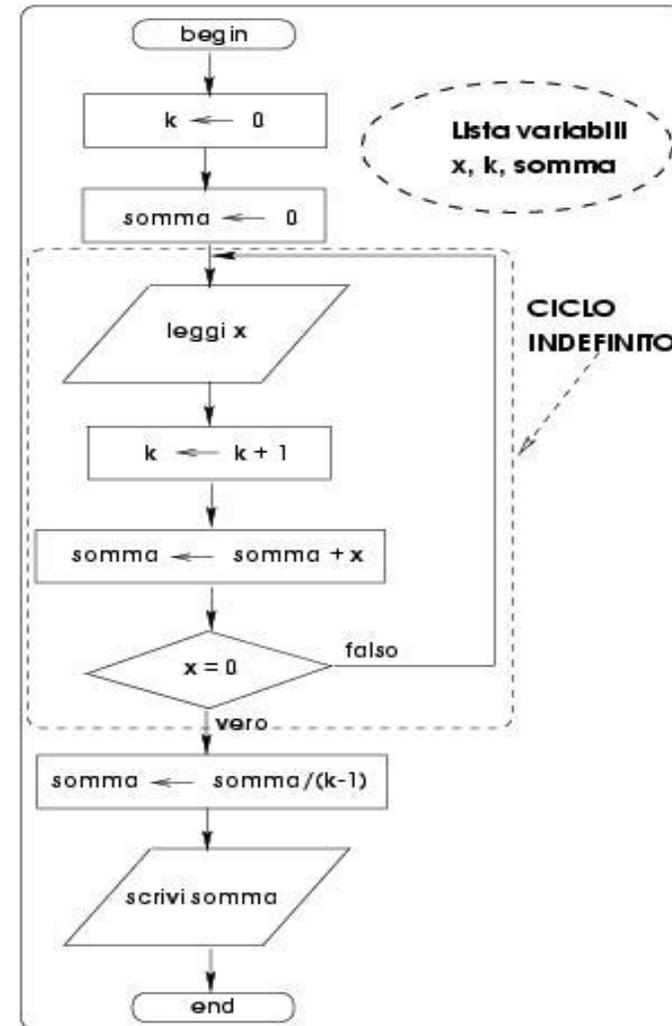
- Un ciclo è **definito** quando è noto a priori quante volte deve essere eseguito; un ciclo definito è detto anche **enumerativo**
- Un **contatore del ciclo** tiene memoria di quante iterazioni sono state effettuate; può essere utilizzato in due modi:
 - **incremento del contatore**: il contatore viene inizializzato ad un valore minimo (ad es. 0 o 1) e incrementato ad ogni esecuzione del ciclo; si esce dal ciclo quando il valore del contatore eguaglia il numero di iterazioni richieste
 - **decremento del contatore**: il contatore viene inizializzato al numero di iterazioni richiesto e decrementato di uno ad ogni iterazione; si esce dal ciclo quando il valore del contatore raggiunge 0 (o 1)

Gli algoritmi iterativi – 5

- Un ciclo è **indefinito** quando non è possibile conoscere a priori quante volte verrà eseguito
- La condizione di fine ciclo controlla il valore di una o più variabili modificate da istruzioni che fanno parte dell'iterazione
- Comunque, un ciclo deve essere eseguito un numero finito di volte, cioè si deve sempre verificare la **terminazione** dell'esecuzione del ciclo

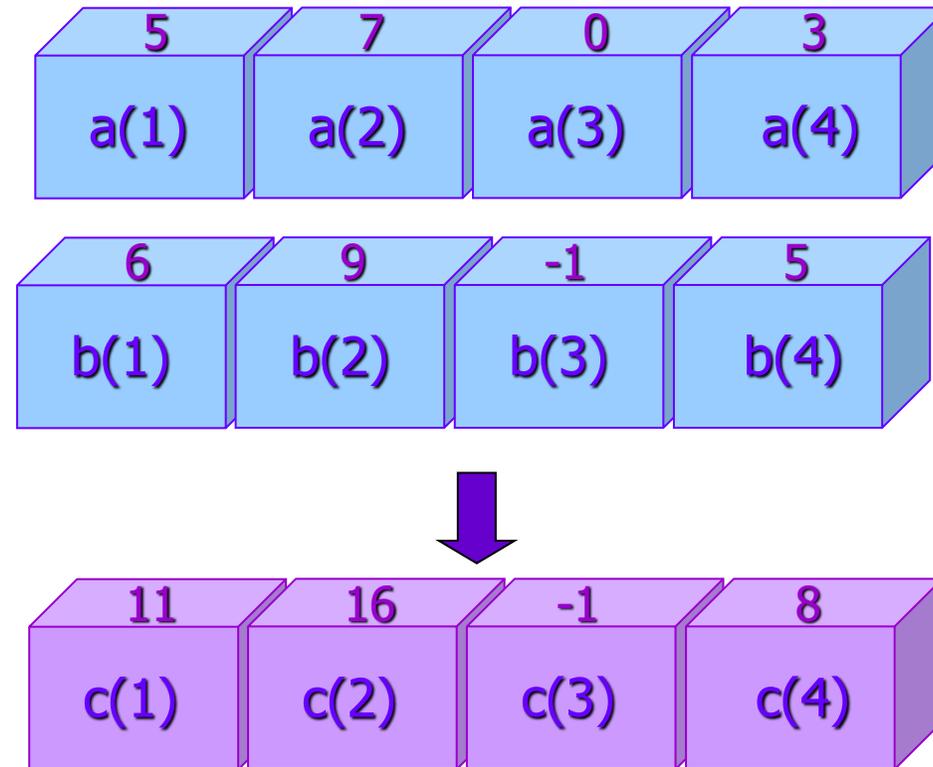
Gli algoritmi iterativi – 6

- **Problema:** Calcolo della media di un insieme di numeri; non è noto a priori quanti sono i numeri di cui deve essere calcolata la media
 - I numeri vengono letti uno alla volta fino a che non si incontra un $x=0$, che segnala la fine dell'insieme



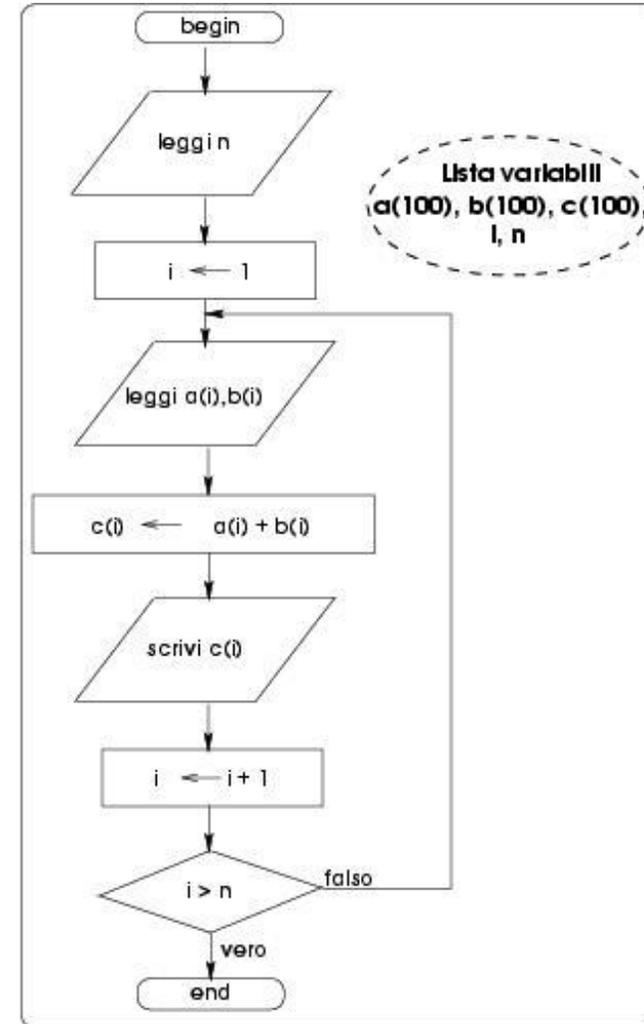
Gli algoritmi iterativi – 7

- **Problema:** Calcolare il vettore somma di due vettori di uguale dimensione n



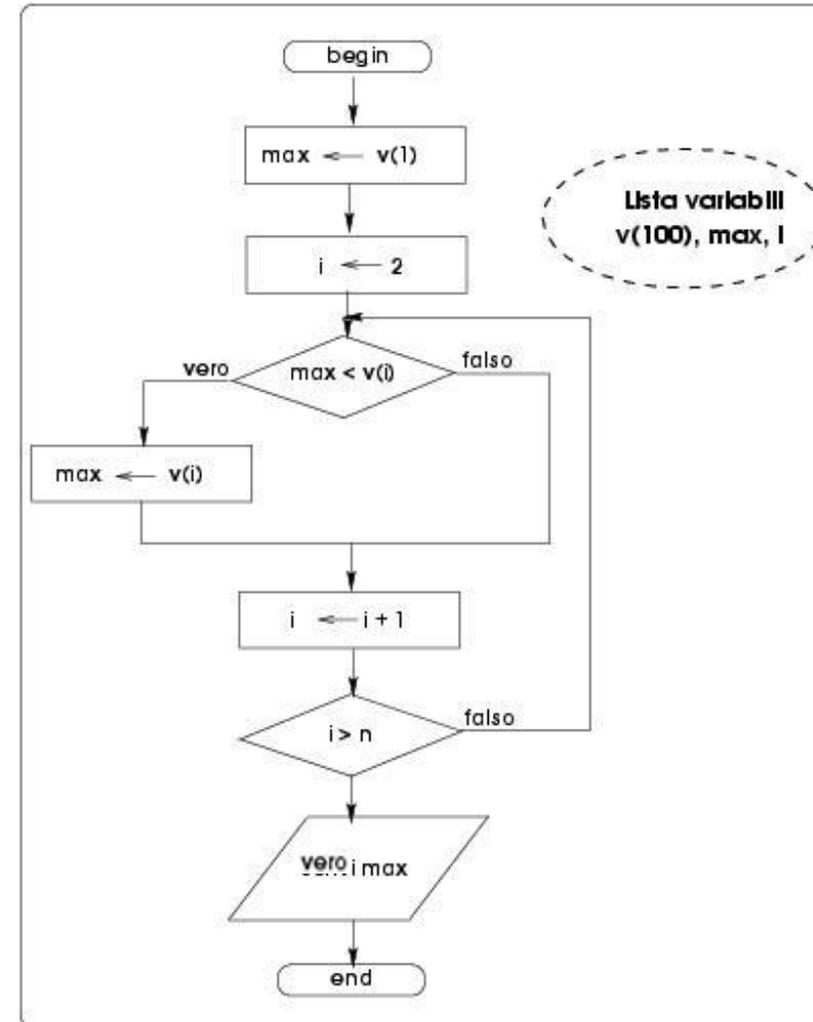
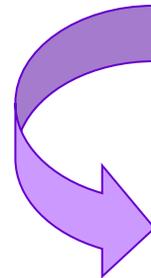
Gli algoritmi iterativi – 8

- L'utilità dei vettori consiste nel poter usare la tecnica iterativa in modo da effettuare la stessa operazione su tutti gli elementi del vettore
- Usando la variabile contatore di un ciclo come indice degli elementi di un vettore è possibile considerarli tutti, uno alla volta, ed eseguire su di essi l'operazione desiderata



Gli algoritmi iterativi – 9

- **Problema:** Calcolo del massimo elemento di un vettore



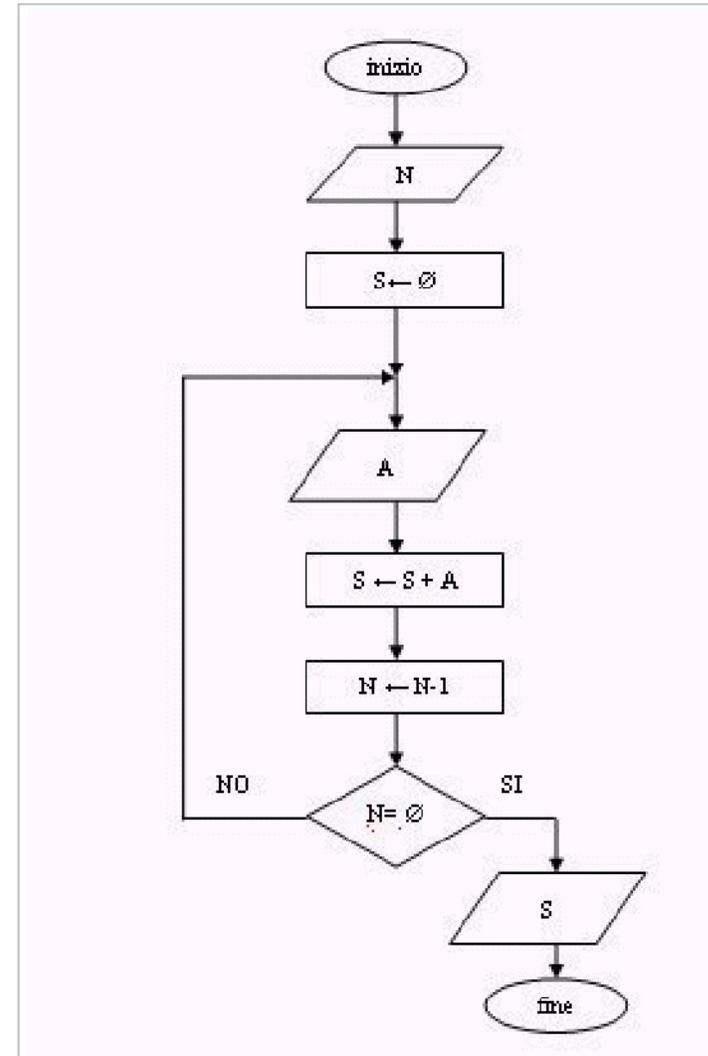
Ancora esempi...

- **Problema:** Somma di una sequenza di numeri

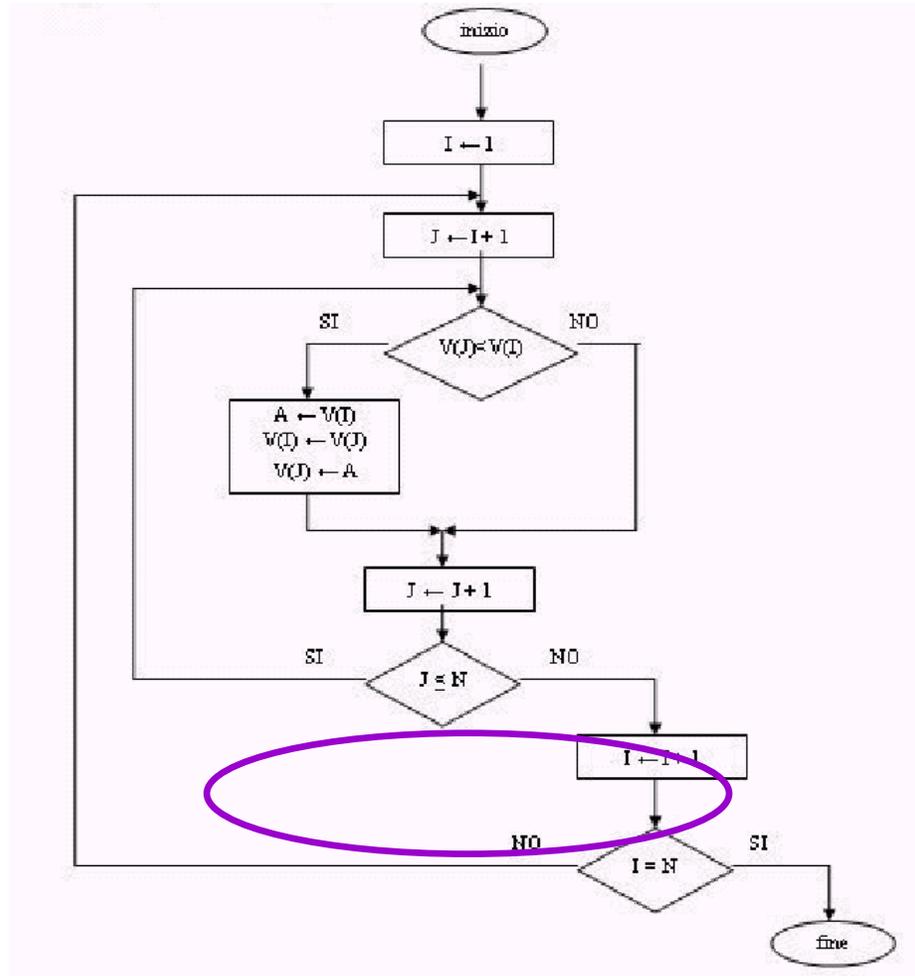
- Indicando con a_i il generico elemento da sommare, la formula generale è

$$S = a_1 + a_2 + \dots + a_n$$

- La variabile n conta quante volte si ripete l'iterazione: n viene decrementata di 1 ad ogni iterazione ed il ciclo termina quando n vale 0
- La variabile A è usata per l'input degli a_i , S per le somme parziali e totale



Ancora esempi...



- **Problema:** Ordinamento per scambio di una sequenza di numeri (crescente)

➤ Indicando con a_i i valori da ordinare, si deve ottenere

$$a_1 < a_2 < a_3 < \dots < a_{n-1} < a_n$$

- Si applica l'algoritmo di ricerca del minimo su tutti gli elementi del vettore e si sposta il minimo in prima posizione
- Si procede analogamente sui rimanenti $n-1$ elementi, $n-2$ elementi, etc.

La pseudocodifica – 1

- La **pseudocodifica** è un linguaggio per la descrizione di algoritmi secondo le regole della programmazione strutturata
- La descrizione di un algoritmo in pseudocodifica si compone di due parti...
 - ◆ la **dichiarazione delle variabili** usate nell'algoritmo
 - ◆ la **descrizione delle azioni** dell'algoritmo

La pseudocodifica – 2

- **Tipo delle variabili**
 - ◆ Il **tipo** di una variabile indica l'insieme dei valori che possono essere assegnati a quella variabile
 - ◆ Su costanti e variabili di un tipo è possibile effettuare le operazioni che sono proprie di quel tipo e tutte le operazioni di confronto
 - ◆ Sono permessi i seguenti 4 tipi: **integer**, **real**, **boolean**, **string-q**

La pseudocodifica – 3

- **integer**: sono le variabili cui possono essere assegnati numeri interi; le costanti di tipo integer sono numeri interi, ad es. 1, -3, 150
- **real**: sono le variabili cui possono essere assegnati numeri razionali; le costanti real possono essere rappresentate in notazione decimale, con un "." che separa la parte intera dalla parte decimale (ad es., 5.17, 12.367, -123., 0.005) o in **notazione scientifica** ($23.476E+3=23476$, $456.985E-3=0.456985$)
- **boolean**: sono le variabili cui possono essere assegnati i valori logici; le costanti logiche sono **true** e **false**
- **string-q**: sono le variabili cui possono essere assegnate parole (o **stringhe**) costituite da q caratteri; le costanti string-q sono costituite da parole di q caratteri racchiuse tra apici (che non fanno parte della costante); ad es., 'FABIO' è una costante string-5, '+' è una costante string-1 e '124' string-3

La pseudocodifica – 4

- Dichiarazione delle variabili

- La dichiarazione delle variabili nella pseudocodifica è un elenco, preceduto dalla parola **var**, delle variabili sulle quali l'algoritmo opera
- Le variabili sono suddivise per tipo: quelle dello stesso tipo sono separate l'una dall'altra da una ","; l'elenco delle variabili dello stesso tipo è seguito dai ":" e dall'indicazione del tipo; gli elenchi di variabili di tipo diverso sono separati dal ";", l'ultimo elenco è seguito da un "."

- Esempio:

```
var i, j, a(20): integer;  
p, q: real;  
nome: string-20;  
sw: boolean.
```

La pseudocodifica – 5

- **Descrizione delle azioni**
 - ◆ Gli schemi di flusso fondamentali sono descritti utilizzando convenzioni linguistiche: ad ogni schema strutturato corrisponde una convenzione linguistica
 - ◆ La descrizione di un algoritmo deve soddisfare le seguenti regole:
 - a) La prima azione dell'algoritmo è preceduta dalla parola **begin**;
 - b) L'ultima azione dell'algoritmo è seguita dalla parola **end**;
 - c) L'azione di lettura è rappresentata dalla parola **read**;
 - d) L'azione di scrittura è rappresentata dalla parola **write**;
 - e) Lo schema di sequenza di n flussi S_1, S_2, \dots, S_n è rappresentato come

$S_1;$

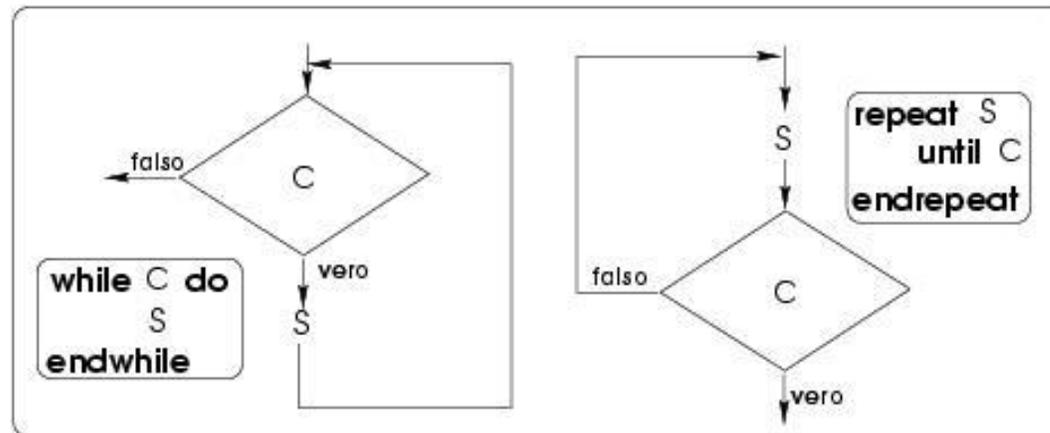
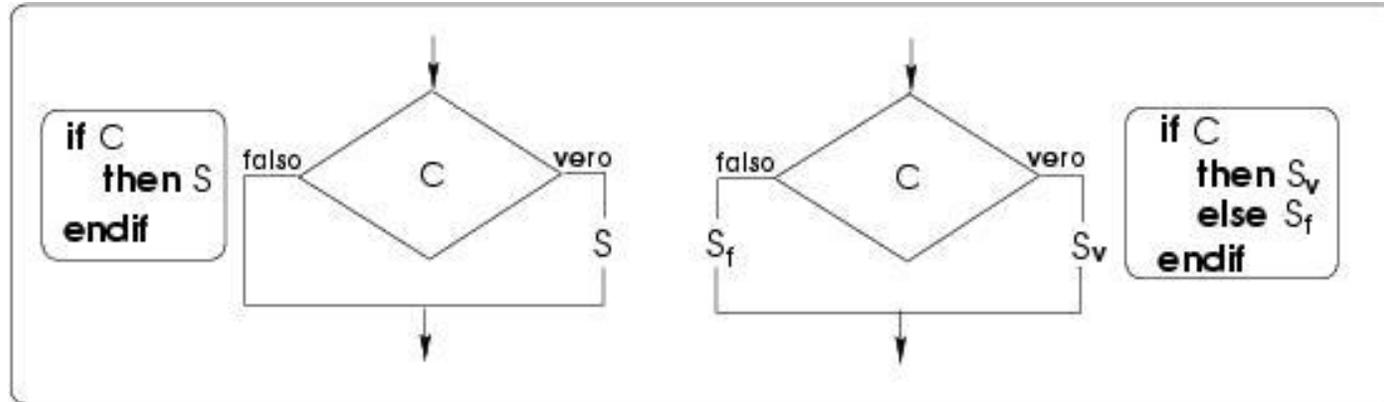
$S_2;$

$S_n;$
 $..!$

La pseudocodifica – 6

Gli schemi di selezione sono rappresentati come:

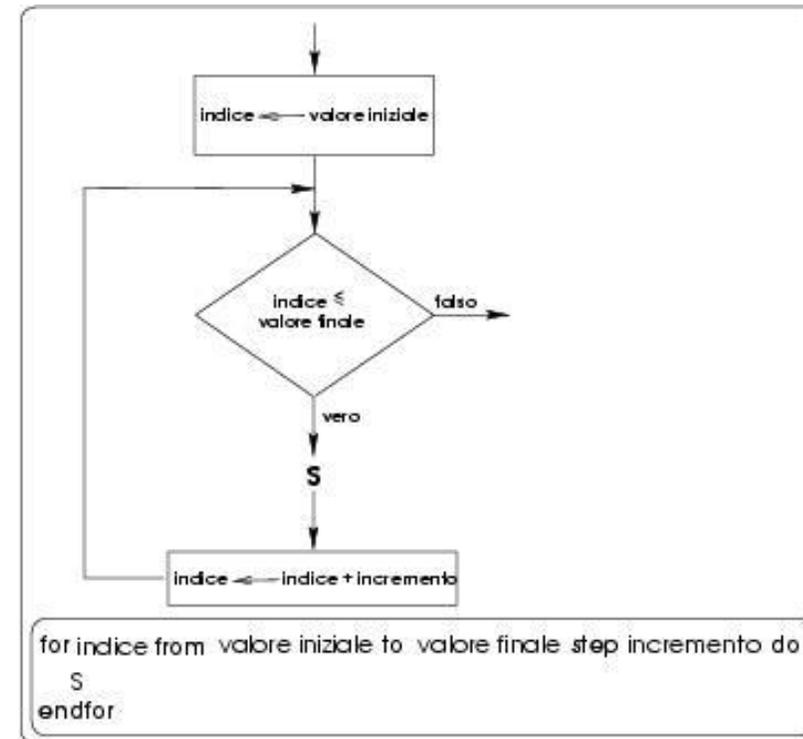
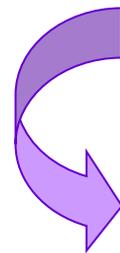
S, *S_f*, *S_v* sono schemi di flusso strutturati



La pseudocodifica – 7

- Esistono convenzioni linguistiche alternative in relazione a particolari schemi di flusso
- **Esempio:** Ciclo enumerativo

Se il valore di "incremento" è 1, la parte "**step** incremento" della frase **for...endfor** può essere omessa



La pseudocodifica – 8

- **Esempio:** Algoritmo per il calcolo del vettore somma di due vettori di numeri razionali

```
var a(100), b(100), c(100): real;  
    i, n: integer.  
  
begin  
    read n;  
    for i from 1 to n do  
        read a(i), b(i);  
        c(i) ← a(i) + b(i);  
        write c(i)  
    endfor  
end
```

La pseudocodifica – 9

- **Esempio:** Algoritmo per il calcolo del massimo elemento di un vettore di numeri razionali

```
var max, v(100): real;  
    i, n: integer.  
begin  
    read n;  
    for i from 1 to n do  
        read v(i)  
    endfor  
    max ← v(1);  
    for i from 2 to n do  
        if max < v(i)  
            then max ← v(i)  
        endif  
    endfor  
    write max  
end
```

La pseudocodifica – 10

- **Esempio:** Algoritmo per il calcolo delle radici di equazioni di 2° grado

```
var x1, x2, a, b, c, delta: real.  
begin  
  read a, b, c;  
  delta ←  $b^2 - 4ac$ ;  
  if delta < 0  
    then write "non esistono radici reali"  
  else if delta = 0  
    then x1 ←  $-b/2a$ ;  
         x2 ← x1  
    else x1 ←  $(-b + \sqrt{\text{delta}})/2a$ ;  
         x2 ←  $(-b - \sqrt{\text{delta}})/2a$   
    endif  
  write x1, x2  
endif  
end
```

Ancora esempi...

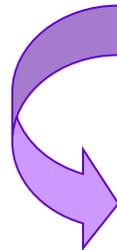
- **Esempio:** Algoritmo per il calcolo della somma di una sequenza di numeri

```
var a, s: real;  
    n: integer.  
begin  
    read n;  
    s ← 0;  
    repeat  
        read a;  
        s ← s+a;  
        n ← n-1  
    until n=0  
endrepeat  
write s  
end
```

Ancora esempi...

- **Esempio:** Ordinamento crescente per scambio

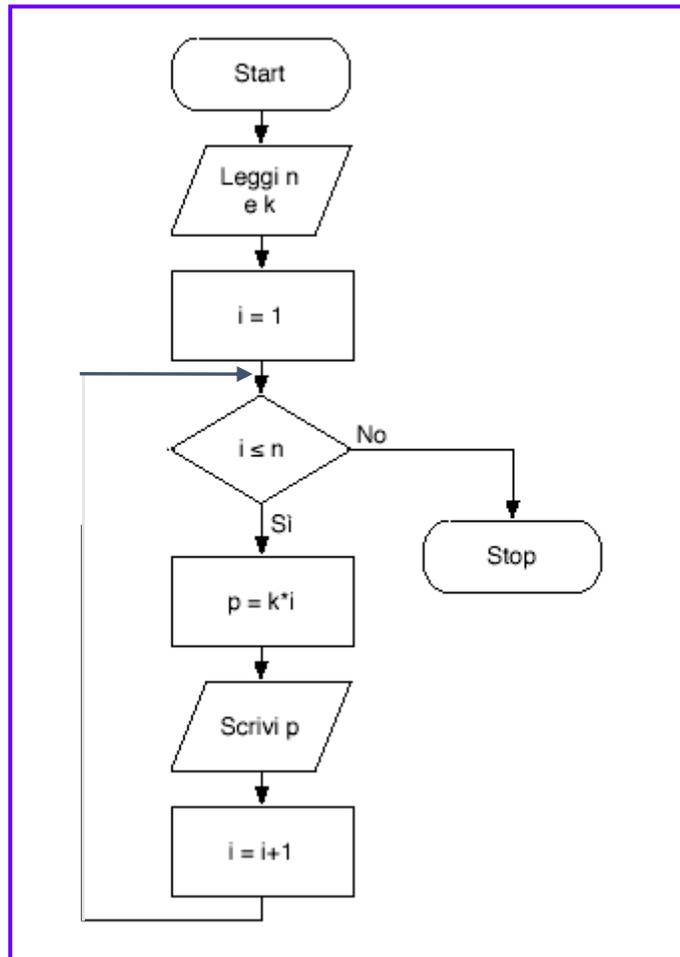
Si suppone che (la dimensione e)
gli elementi del vettore siano già
stati letti e memorizzati



```
var a, v(100): real;  
    i, j, n: integer.  
begin  
    i ← 1;  
    repeat  
        j ← i+1;  
        repeat  
            if v(j) < v(i)  
                then a ← v(i);  
                    v(i) ← v(j);  
                    v(j) ← a  
            endif  
            j ← j+1  
        until j > n  
    endrepeat  
    i ← i+1  
    until i = n  
endrepeat  
end
```

Un esempio "comparativo"

- Letti due interi n e k , entrambi maggiori di zero, stampare i primi n multipli di k



```

var i, n, k, p: integer.
begin
  read n;
  read k;
  for i from 1 to n do
    p ← k*i;
    write p
  endfor
end
    
```

```

i ← 1;
while i ≤ n
  p ← k*i;
  write p;
  i ← i+1;
endwhile
    
```

```

#include <stdio.h>
main()
{
  int i, n, k, p;
  scanf("%d", &n);
  scanf("%d", &k);
  for (i=1; i<=n; i++)
  {
    p = k*i;
    printf("%d", p);
  }
  exit(0);
}
    
```

Gli algoritmi ricorsivi – 1

- Un algoritmo si dice **ricorsivo** quando è definito in termini di se stesso, cioè quando una sua istruzione richiede una nuova esecuzione dell'algoritmo stesso
- La definizione ricorsiva di un algoritmo è suddivisa in due parti:
 - a) la **base della ricorsione**, che stabilisce le condizioni iniziali, cioè il risultato che si ottiene per i dati iniziali (in generale per 0 e/o 1)
 - b) la **regola di ricorsione**, che definisce il risultato per un valore n , diverso dal valore (i) iniziale per mezzo di un'espressione nella quale si richiede il risultato dell'algoritmo calcolato per $n-1$

Gli algoritmi ricorsivi – 2

- **Esempio:** Prodotto di numeri interi

$$a \times b = \begin{cases} 0 & \text{se } b=0 \text{ (base della ricorsione)} \\ a \times (b-1) + a & \text{se } b \neq 0 \text{ (regola di ricorsione)} \end{cases}$$

- Secondo la definizione ricorsiva si ha:

$$3 \times 2 = 3 \times 1 + 3 = 3 \times 0 + 3 + 3 = 0 + 3 + 3 = 6$$

- L'esecuzione di un algoritmo ricorsivo termina sempre: la regola di ricorsione prevede nuove esecuzioni su dati decrescenti, fino ad ottenere i dati di inizio ricorsione

Gli algoritmi ricorsivi – 3

- **Esempio:** Calcolo del fattoriale di un numero intero
 - Il fattoriale di n è il prodotto di tutti gli interi da 1 ad n , cioè
$$n! = n \times (n-1) \times (n-2) \times \dots \times 2 \times 1$$
 - Per definizione, $0! = 1$

```
begin fattoriale(n)
  if n = 0
    then r ← 1
    else r ← n × fattoriale(n-1)
  endif
end
```

Esercizio – 1

- **La successione di Fibonacci**

- Leonardo Pisano, detto Fibonacci, pose il seguente quesito:

- ◆ Una coppia di conigli giovani impiega una unità di tempo a diventare adulta; una coppia adulta impiega una unità di tempo a riprodursi e generare un'altra coppia di conigli (chiaramente giovani); i conigli non muoiono mai
 - ◆ Quante coppie di conigli abbiamo al tempo t generico se al tempo $t=0$ non abbiamo conigli e al tempo $t=1$ abbiamo una coppia di giovani conigli?

Esercizio – 2

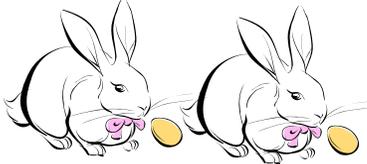
t=0

t=1

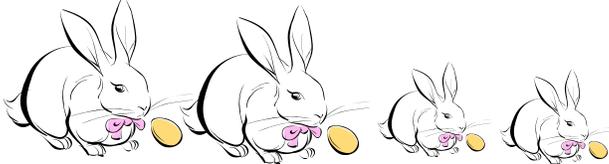


Dio genera la prima coppia di conigli

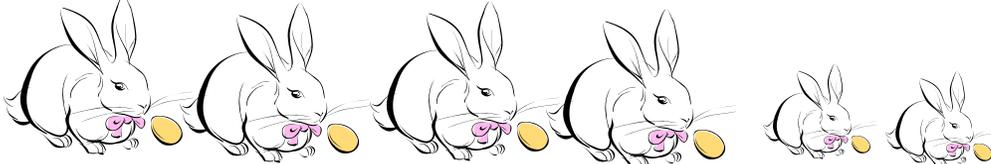
t=2



t=3

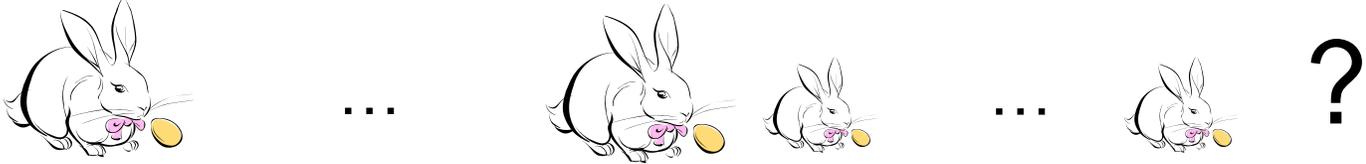


t=4



...

t=N



Esercizio – 3

- **La successione di Fibonacci**

- Il calcolo di F_n (numero di coppie di conigli), per qualsiasi tempo t , genera la successione dei numeri di Fibonacci
- La relazione di ricorsione è

$$F_0=0, F_1=1,$$
$$F_n = F_{n-1} + F_{n-2}$$

Considerazioni finali – 1

- **Attenzione alla scelta di un “buon” algoritmo...**
 - Due algoritmi si dicono **equivalenti** quando:
 - ◆ hanno lo stesso dominio di ingresso
 - ◆ hanno lo stesso dominio di uscita
 - ◆ in corrispondenza degli stessi valori nel dominio di ingresso producono gli stessi valori nel dominio di uscita
 - Due algoritmi equivalenti forniscono lo stesso risultato, ma possono avere diversa efficienza e possono essere profondamente diversi

Considerazioni finali – 2

- Un esempio di due algoritmi equivalenti, ma con diversa efficienza, per la moltiplicazione fra interi è...

Algoritmo 1	Algoritmo 2 (somma e shift)
Somme successive: $12 \times 12 = 12 + 12 + \dots + 12 = 144$	$12 \times$ <u>12 =</u> 24 <u>12 =</u> 144

Considerazioni finali – 3

- Esistono problemi che non possono essere risolti tramite un calcolatore elettronico perché...
 - ...la soluzione del problema non esiste
 - ...la soluzione del problema richiederebbe un tempo di calcolo eccessivo (anche infinito)
 - ...la natura del problema è percettiva e/o la soluzione del problema è "soggettiva"

Considerazioni finali – 4

- Un esempio di problema **indecidibile**, tale cioè che non esista alcun algoritmo capace di risolverlo, è il *problema decisionale della terminazione*:

Dato un algoritmo B ed i suoi dati D , stabilire se la computazione $B(D)$ termina

- In questo caso, infatti, non esiste un algoritmo A , che accettata una qualsiasi coppia B, D come dato in ingresso, stabilisca *sempre* in tempo finito se $B(D)$ termina o meno
- Nota: A non può semplicemente consistere nel comandare l'esecuzione $B(D)$ e controllarne il comportamento, poiché, se tale esecuzione non terminasse, A non risponderebbe in tempo finito

Considerazioni finali – 5

- Un esempio di problema la cui soluzione richiederebbe un tempo infinito consiste nello stabilire se, data una funzione intera f , $f(x)$ è costante per ogni valore di x
- Infine, un esempio di problema la cui soluzione è soggettiva è rappresentato dalla scelta, dato un insieme di immagini di paesaggi, del paesaggio più rilassante

Esercizi

- Formalizzare l'algoritmo, attraverso diagramma a blocchi o pseudocodifica, per risolvere i problemi:
 - Siano dati in input due vettori di interi, a e b , di dimensione n (in input). Si calcoli la somma incrociata degli elementi $a(1)+b(n)$, $a(2)+b(n-1)$, etc., la si memorizzi nel vettore c , e lo si stampi.
 - Siano dati in input un vettore v_1 di interi (di dimensione n , in input) ed un intero k . Si determini l'elemento di v_1 più prossimo a k , e lo si stampi assieme all'indice corrispondente.
 - Dato l'insieme dei risultati d'esame (nell'intervallo da 0 a 100) di n studenti, contare il numero di studenti che hanno superato la prova, sapendo che l'esame si intende superato con un voto maggiore o uguale a 50.