

# Gli algoritmi

## # Analisi e programmazione

## # Gli algoritmi

- Proprietà ed esempi
- Proposizioni e predicati
- I diagrammi a blocchi
- Analisi strutturata
- Gli algoritmi iterativi
- Gli algoritmi ricorsivi

# Analisi e programmazione

# Analisi e programmazione – 1

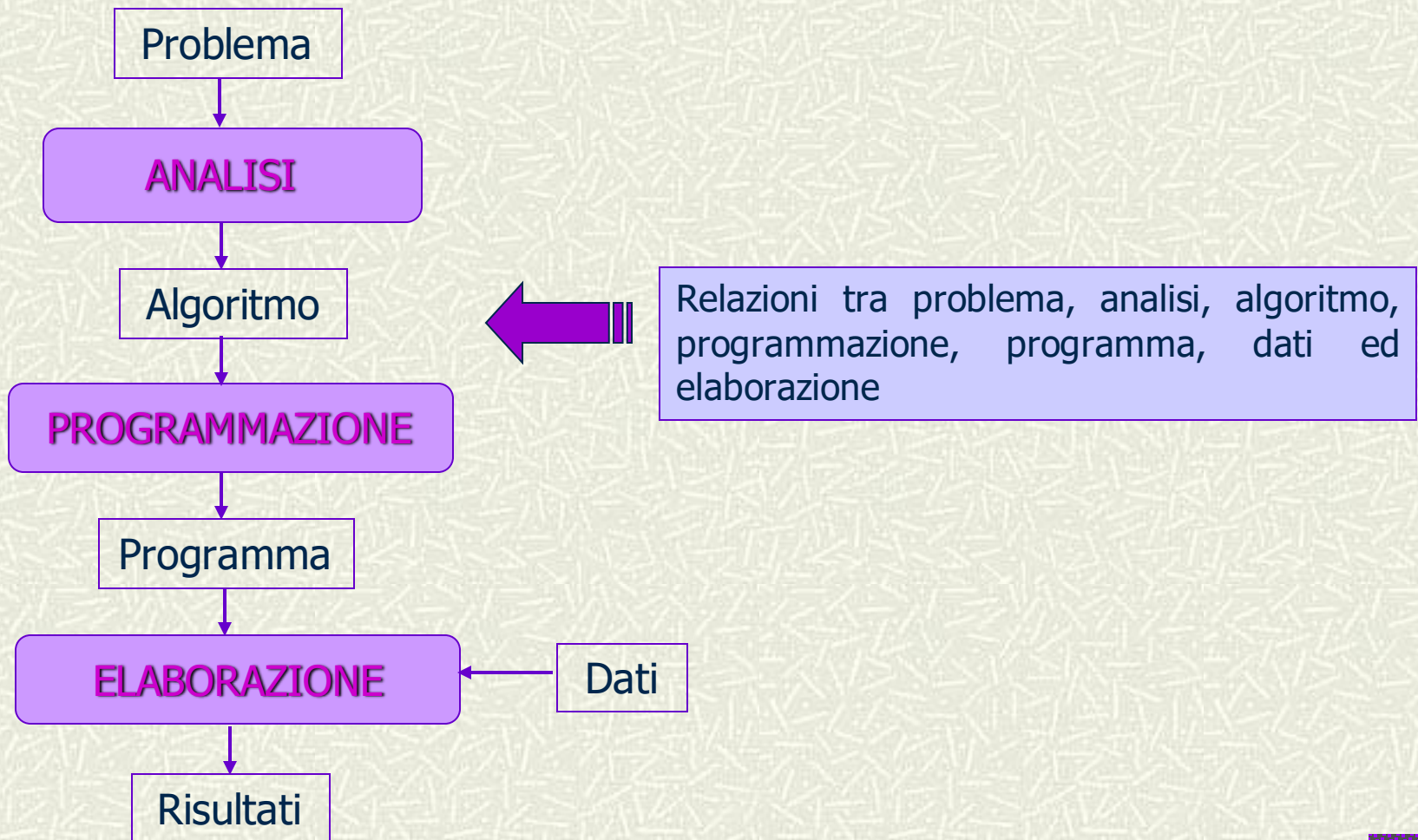
- ✚ Tramite un elaboratore si possono risolvere problemi di varia natura: emissione di certificati anagrafici, gestione dei c/c di un istituto di credito, prenotazioni ferroviarie...
- ✚ Il problema deve essere formulato in modo opportuno, perché sia possibile utilizzare un elaboratore per la sua soluzione
- ✚ 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**  $\Rightarrow$  definire un **algoritmo**
  - ✚ **Scopo della programmazione**  $\Rightarrow$  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

# Le fasi del procedimento di analisi e programmazione

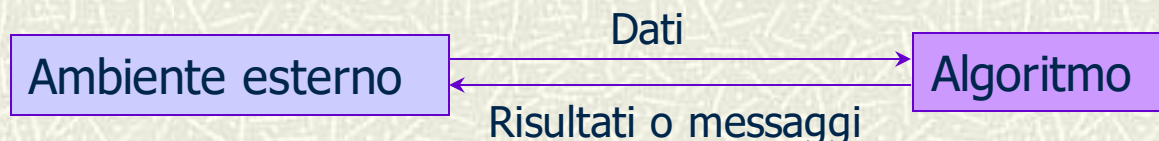


# Gli algoritmi



# 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: 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



# 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"

# 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)  $\neq$  (diverso)
  - $>$  (maggiore)  $<$  (minore)
  - $\geq$  (maggiore o uguale)  $\leq$  (minore o uguale)
- # I predicati che contengono un solo operatore relazionale sono detti **semplici**

# Proposizioni e predicati – 3

- ✚ Dato un predicato  $p$ , il predicato  $\text{not } p$ , 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**  $p \text{ and } q$  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**  $p \text{ or } q$  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

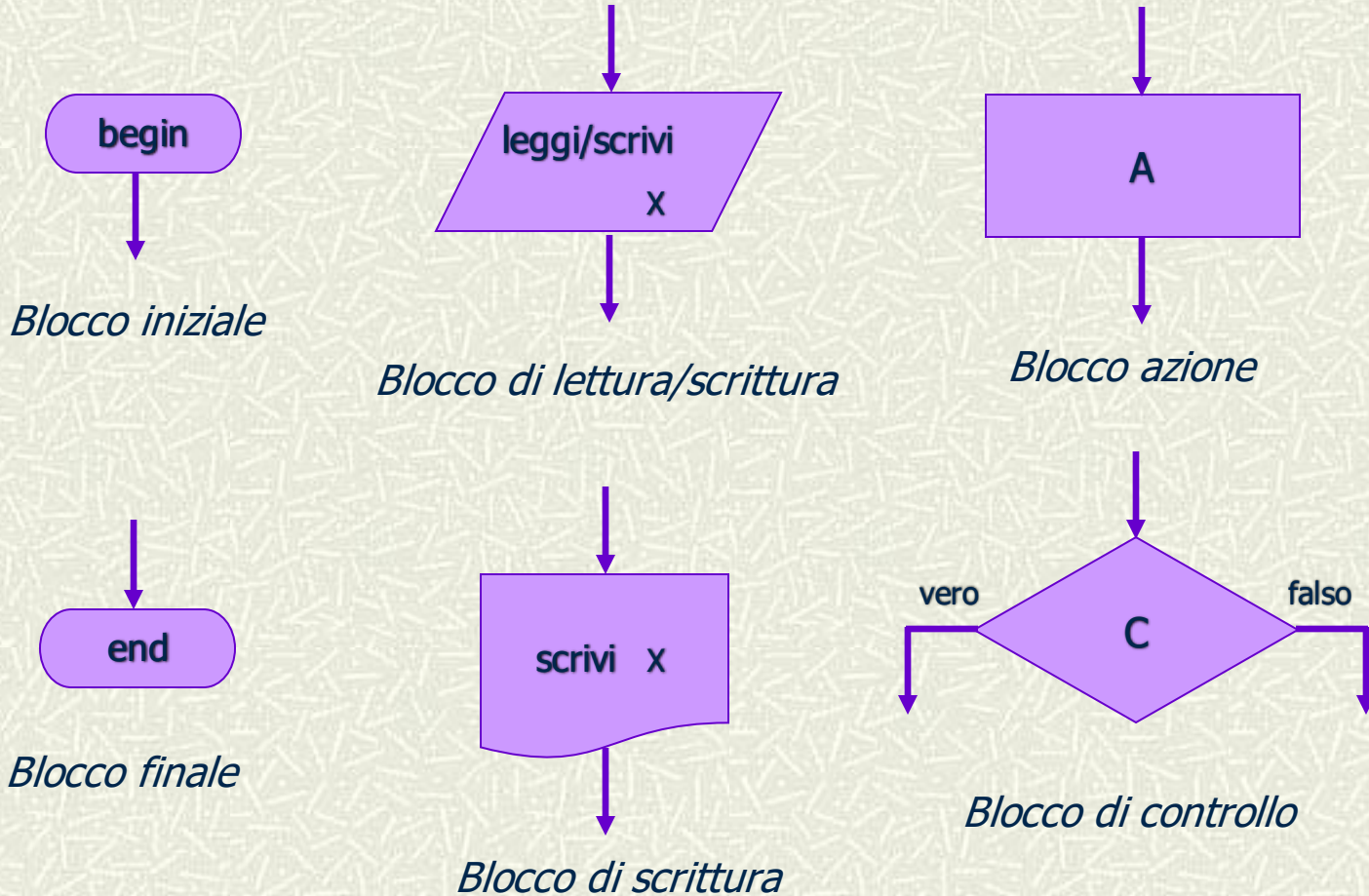
è vero quando il valore di base è maggiore del valore di altezza, o quando il valore di base è maggiore di 100, o quando entrambe le condizioni sono verificate



# 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

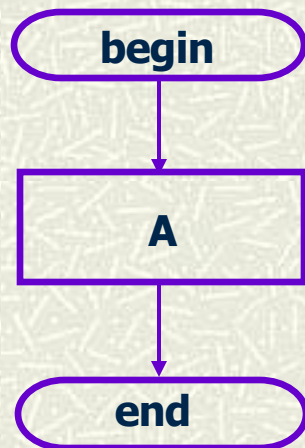
- L'insieme dei blocchi elementari che descrivono un algoritmo deve soddisfare le seguenti condizioni:
  - ciascun blocco di azione o di lettura/scrittura ha una sola freccia entrante ed una sola freccia uscente
  - ciascun blocco di controllo ha una sola freccia entrante e due frecce uscenti
  - ciascuna 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

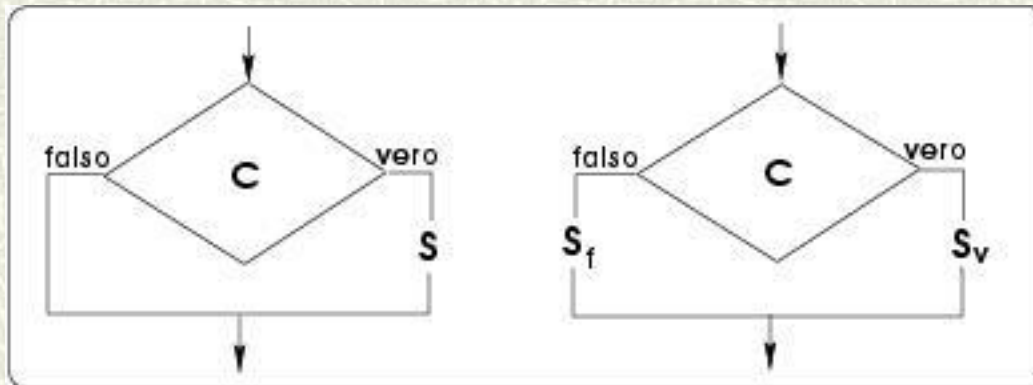


$S_1, S_2, \dots, S_n$  schemi di flusso strutturati



# Analisi strutturata – 3

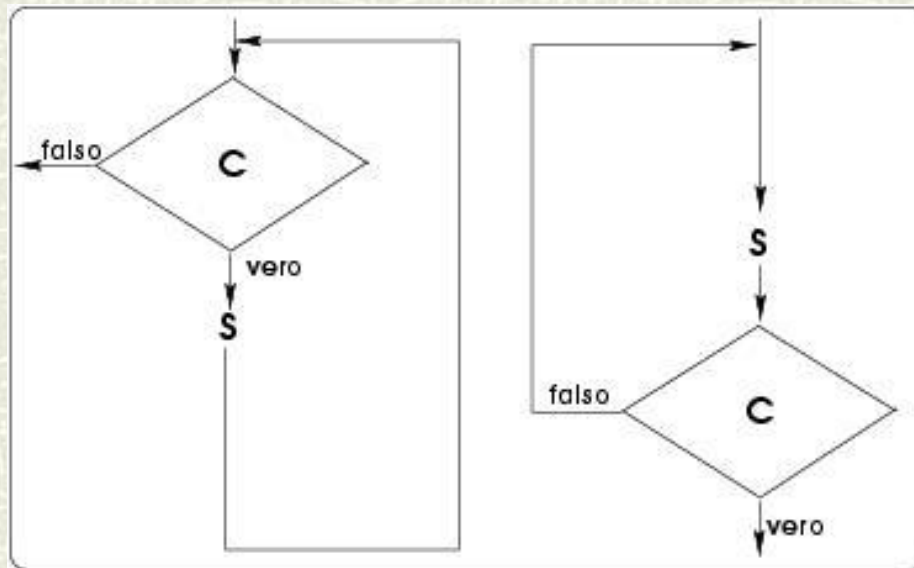
2) ...è uno schema di selezione



- 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

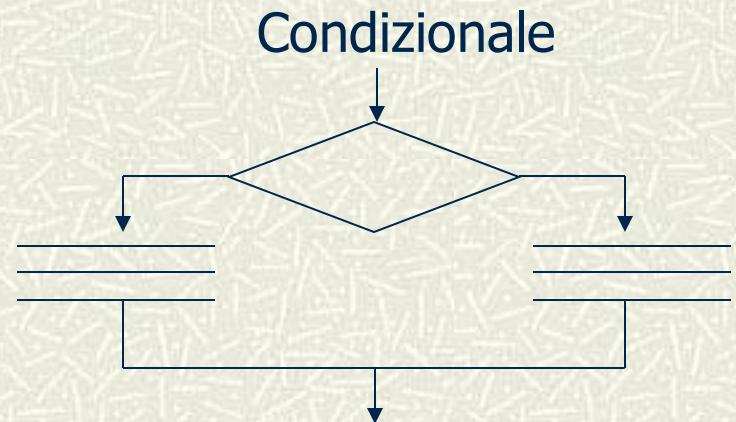
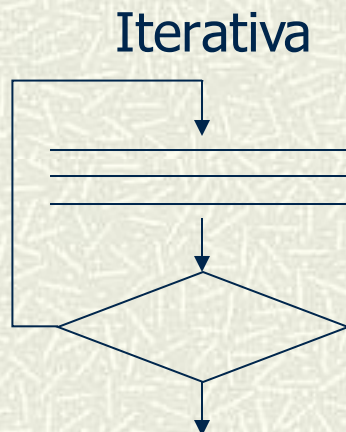
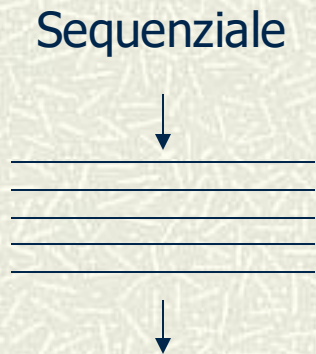
- # Gli schemi di flusso sono **aperti** quando consentono una sola esecuzione di una sequenza di blocchi elementari, sono **chiusi** quando permettono più di un'esecuzione della sequenza di blocchi
- # Gli schemi di sequenza e di selezione sono aperti, lo schema di iterazione è chiuso
- # **Ogni diagramma 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:

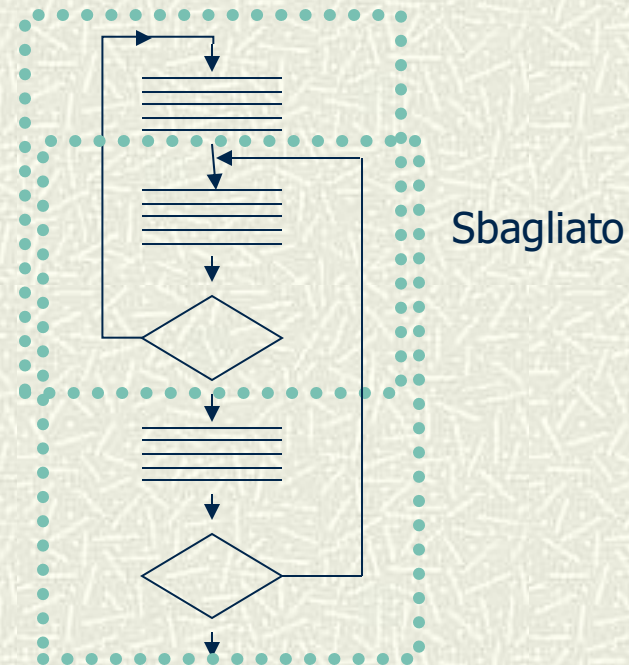
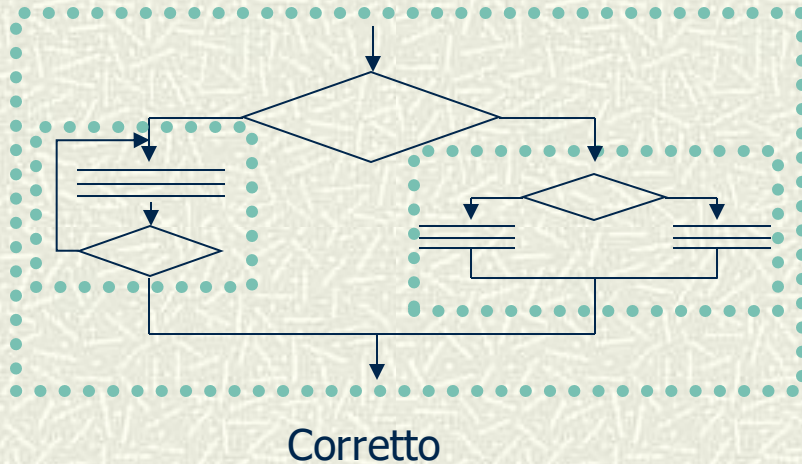


# Analisi strutturata – 7

- # 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 sconsigliato delle istruzioni *go to* e alla definizione delle linee guida della programmazione strutturata, sviluppate negli anni `70

# Analisi strutturata – 8

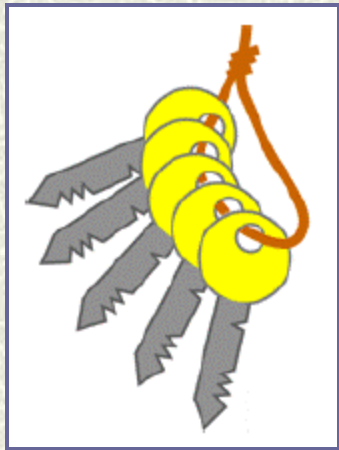
- In un diagramma strutturato non apparirà mai una istruzione di salto incondizionato
- I tre schemi fondamentali possono essere *concatenati*, uno di seguito all'altro, o *nidificati*, uno dentro l'altro; non possono in nessun caso essere "intrecciati" o "accavallati"





# Esempio

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



# Esercizi

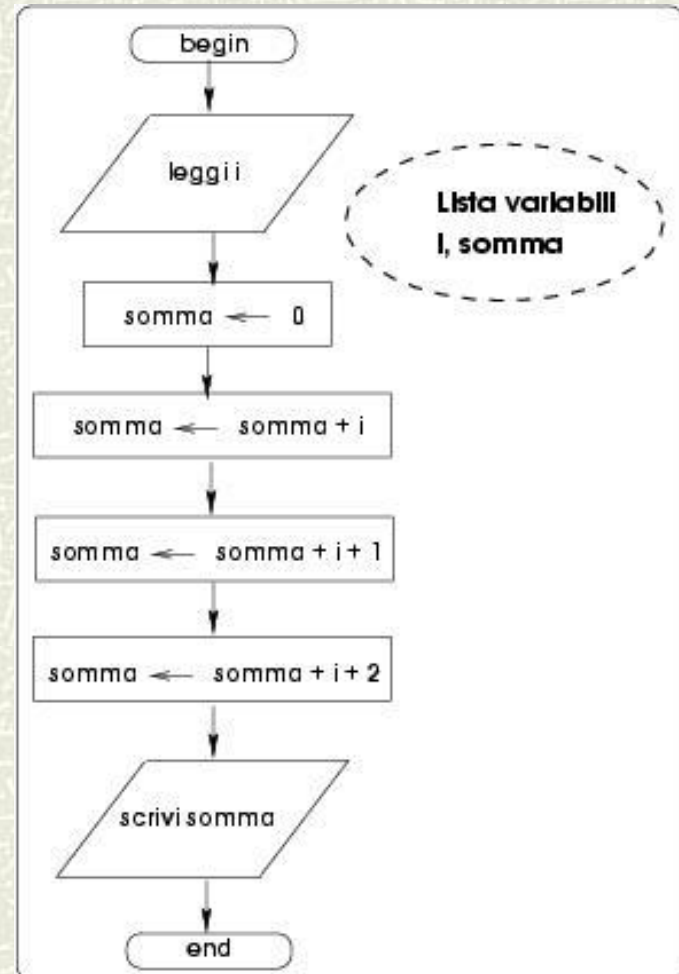
- # Scrivere un algoritmo, e rappresentarlo tramite diagramma a blocchi, per la soluzione dei seguenti problemi:
  - calcolare l'area del triangolo
  - trovare il max di due numeri
  - moltiplicare due numeri (usando solo l'operazione di somma)
- # Formalizzare, tramite diagramma a blocchi, l'algoritmo 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:**

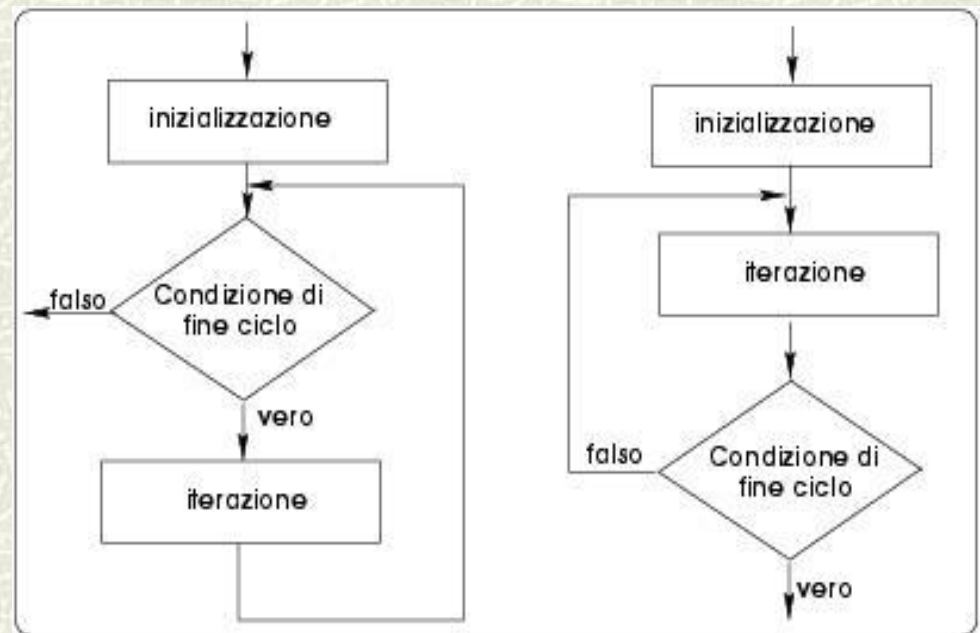
- ◆ La variabile **somma** è un contenitore di somme parziali, finché non si ottiene la somma totale richiesta
- ◆ La soluzione del problema viene raggiunta eseguendo azioni simili per un numero opportuno di volte





# 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

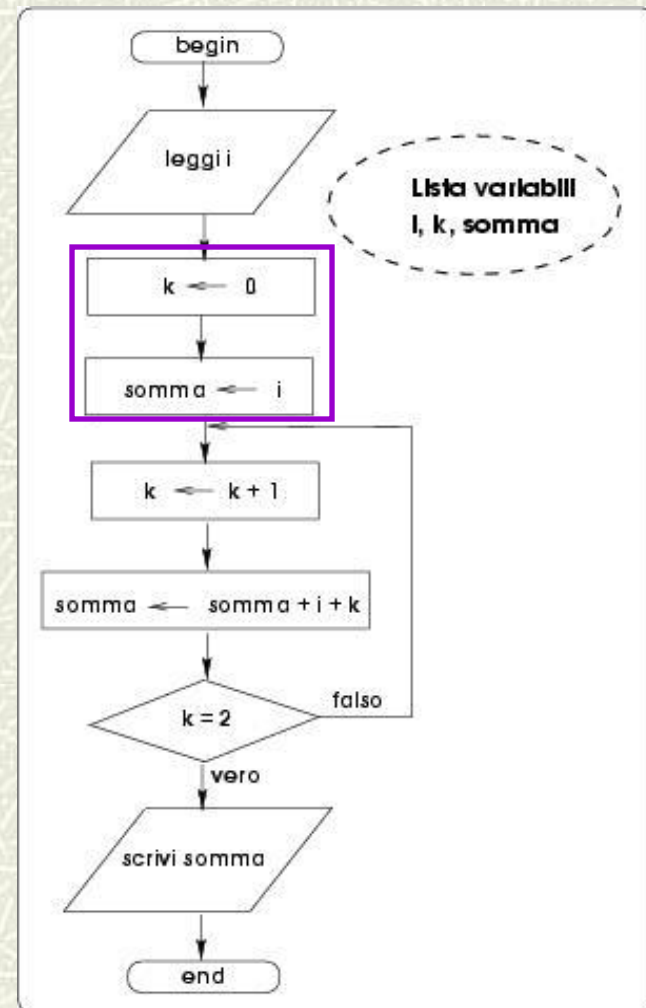
Ciclo con controllo in coda

# 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)

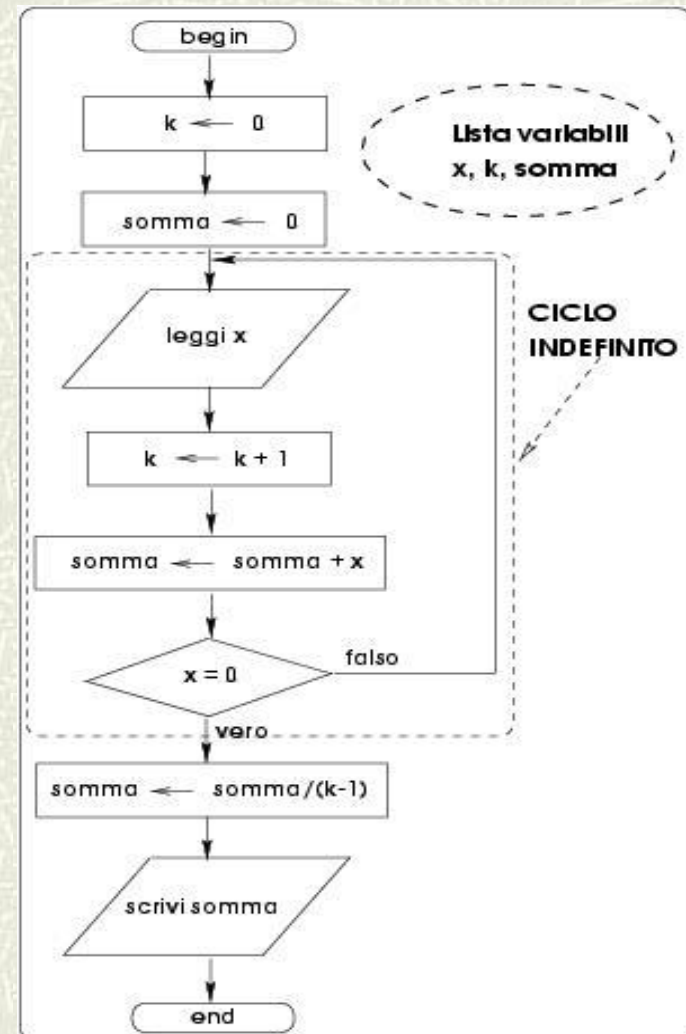


# Gli algoritmi iterativi – 5

- # 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

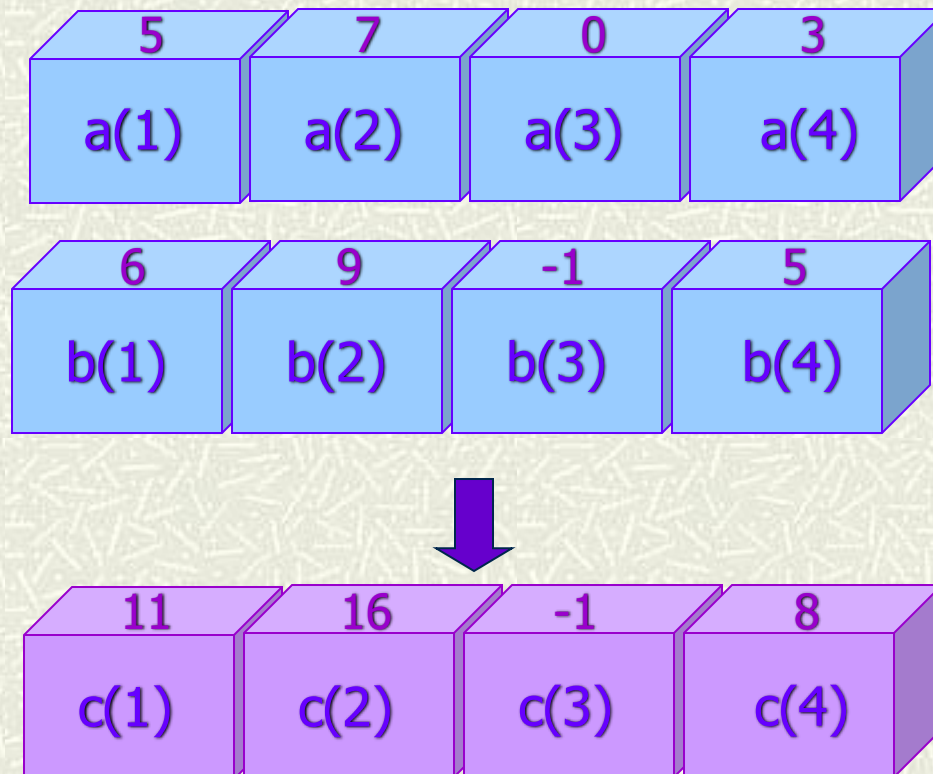
# Gli algoritmi iterativi – 6

- **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

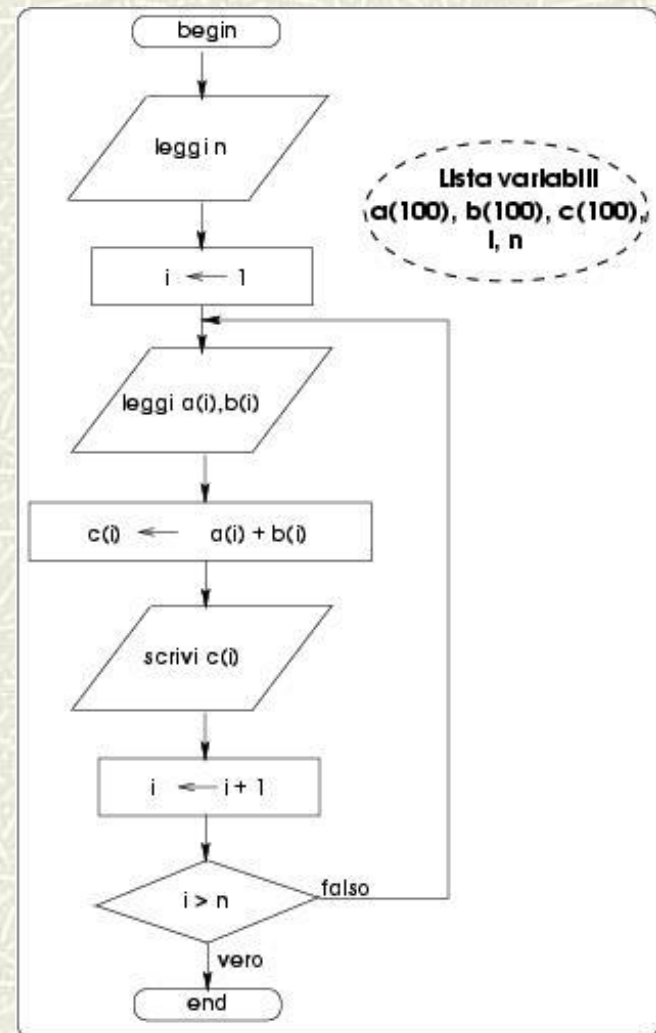
- **Problema:** Calcolare il vettore somma di due vettori di uguale dimensione  $n$





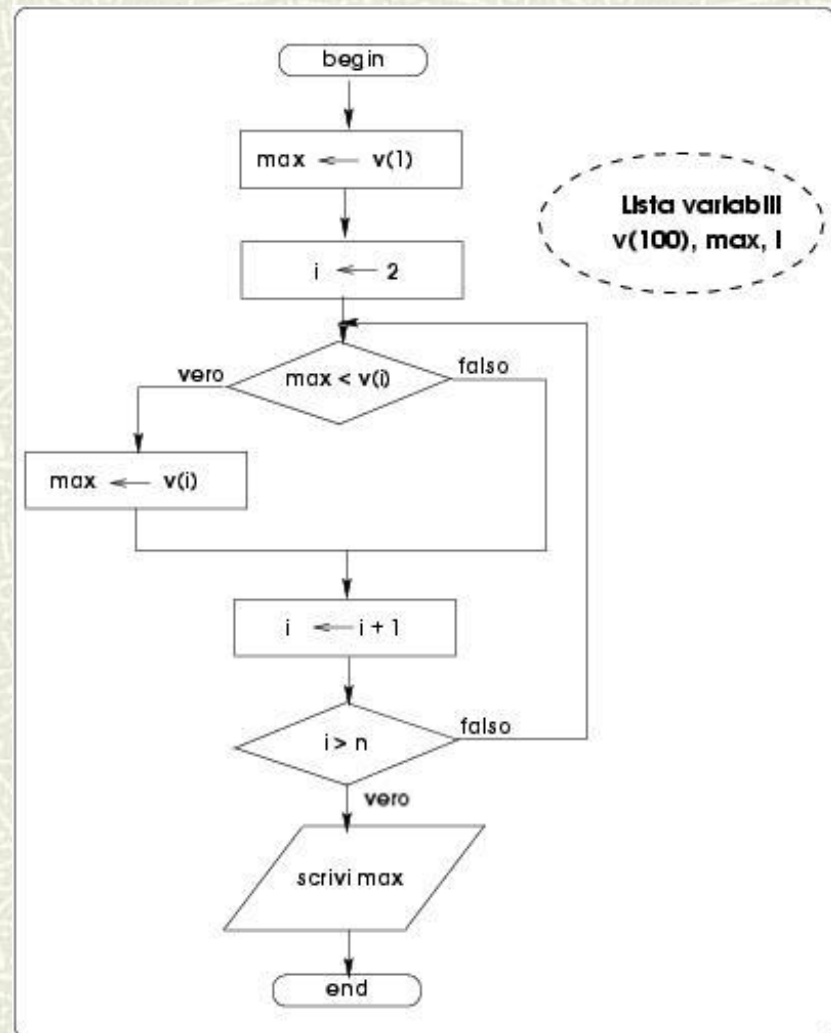
# Gli algoritmi iterativi – 8

- # 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



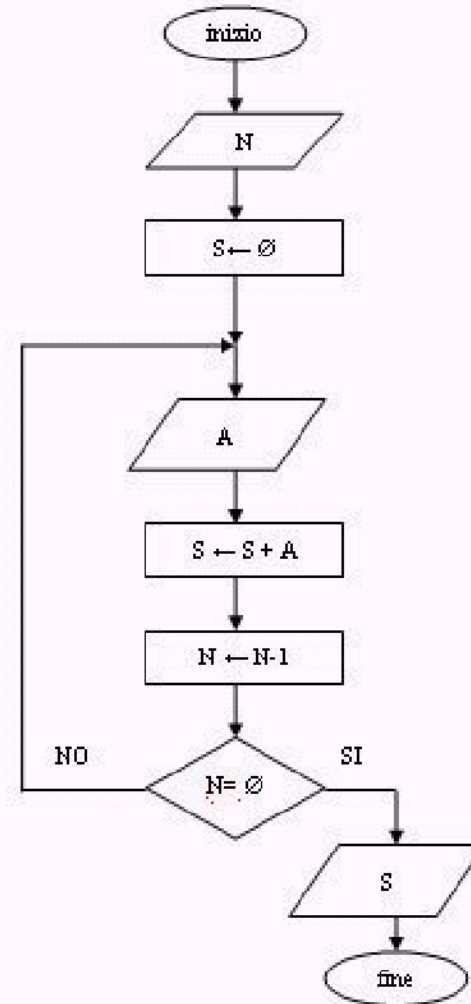
# Ancora esempi...

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

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

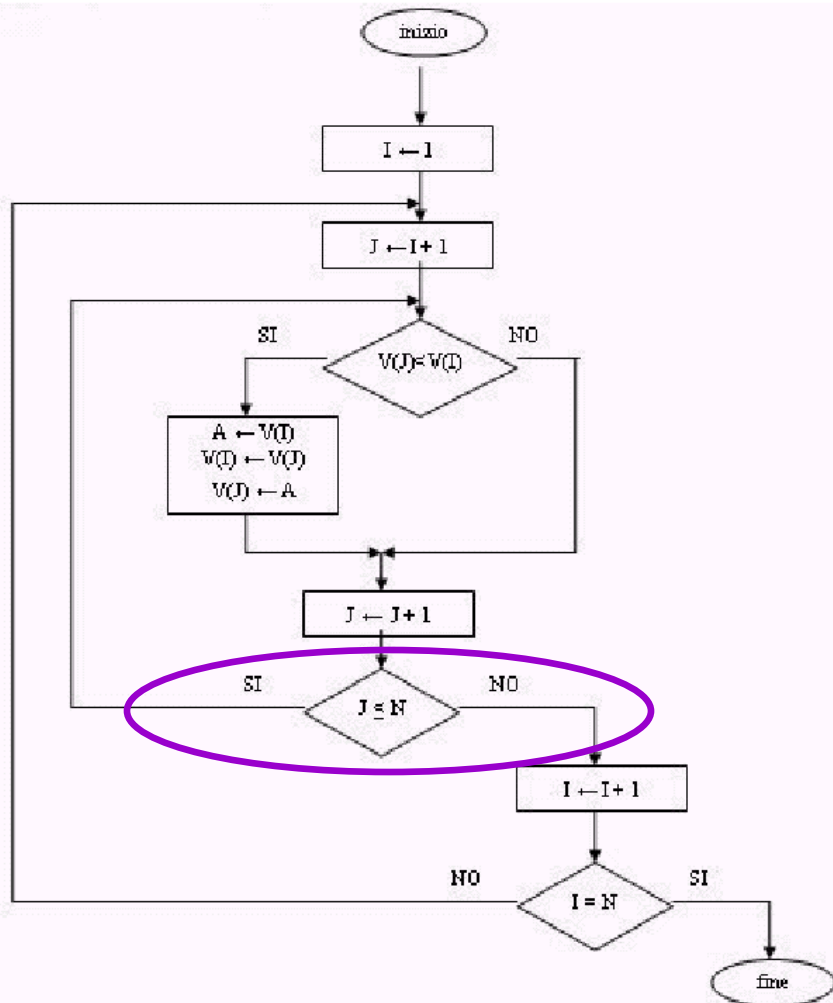
$$S = a_1 + a_2 + \dots + a_n$$

- 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.

# 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

$$n! = n \times (n-1) \times (n-2) \times \dots \times 2 \times 1$$

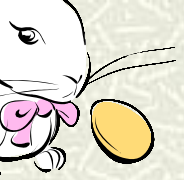
- ✚ Per definizione,  $0! = 1$

```
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

$t=0$

$t=1$

$t=2$

$t=3$

$t=4$

...

$t=N$

...

...

?



# 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: $12 \times 12 = 12 + 12 + \dots + 12 = 144$	$\begin{array}{r} 12 \times \\ \underline{12 =} \\ 24 \\ \underline{12 =} \\ 144 \end{array}$



# 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, 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.