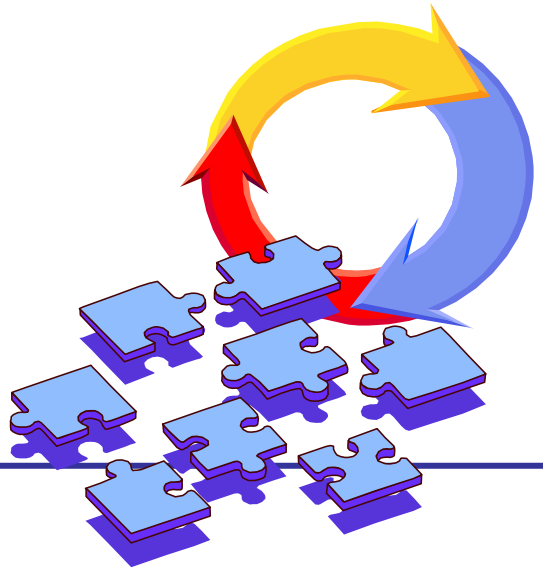


Informatica

Introduzione alla programmazione

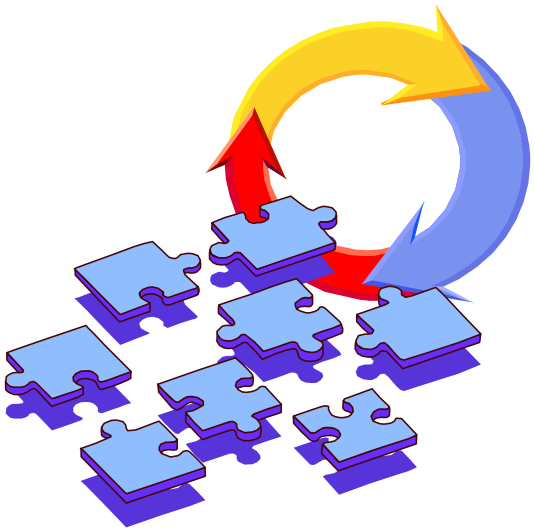
Vibo Valentia, 24 ottobre 2005

Ercole Colonese

e.colonese@virgilio.it

La programmazione

- Analisi e programmazione
- Algoritmi
- Diagrammi a blocchi
- Analisi strutturata
- Costanti e variabili
- Istruzioni
- Vettori e matrici
- Algoritmi iterativi



La programmazione

Analisi e programmazione

Algoritmi

Diagrammi a blocchi

Analisi strutturata

Costanti e variabili

Istruzioni

Vettori e matrici

Algoritmi iterativi

Analisi e programmazione

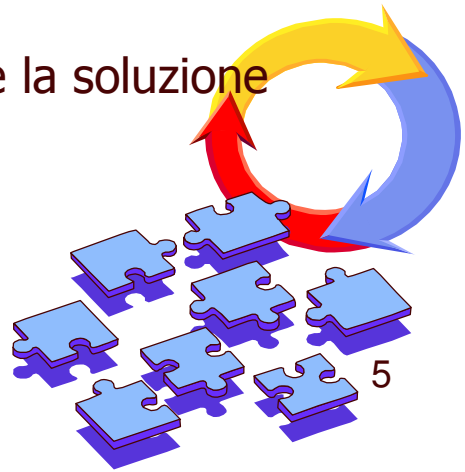
- Tramite un elaboratore si possono risolvere problemi (necessità) di varia natura: *emissione di certificati anagrafici, gestione dei c/c di un istituto di credito, prenotazioni ferroviarie* ...

- Il problema (necessità) deve essere formulato in modo opportuno perché sia possibile utilizzare un elaboratore per la sua soluzione

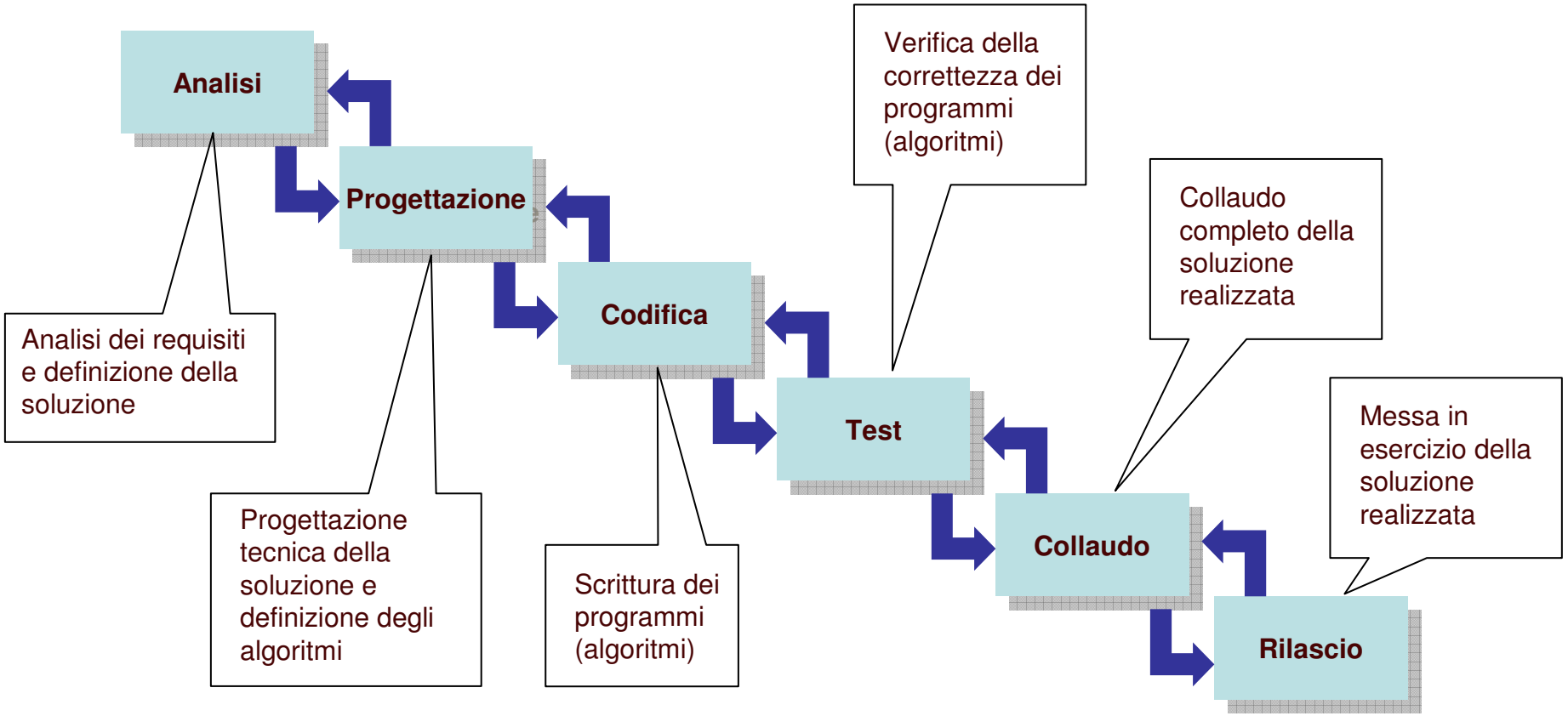
- Per analisi e programmazione si intende l'insieme delle attività preliminari atte a risolvere problemi utilizzando un elaboratore, dalla formulazione del problema fino alla predisposizione della soluzione realizzata
  - Scopo dell'analisi: definire un algoritmo
  - Scopo della programmazione: definire un programma

Ciclo di sviluppo a fasi

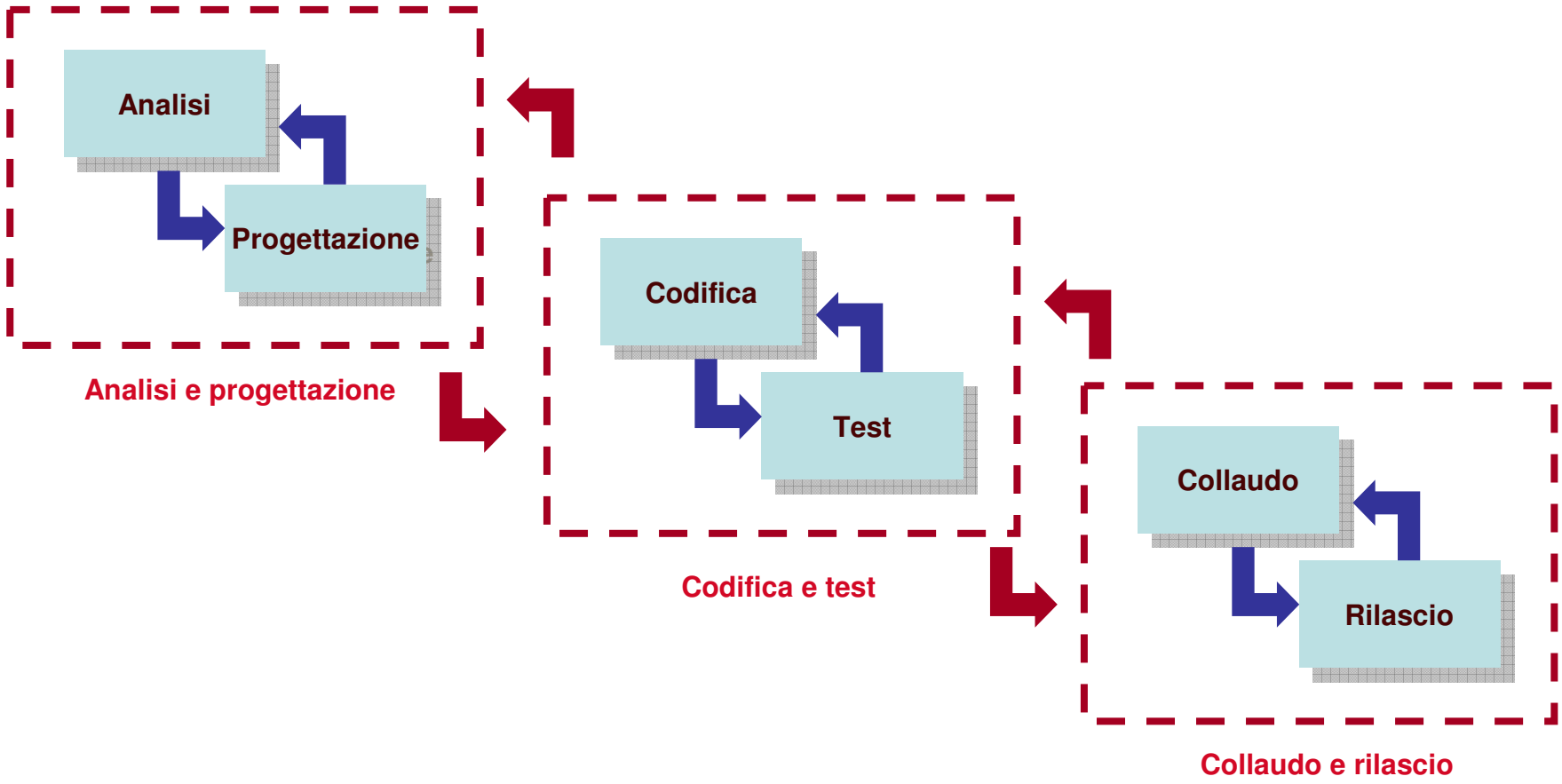
- La realizzazione di una soluzione informatica segue un ciclo di sviluppo composto di fasi
  - Analisi e progettazione: raccogliere i requisiti, analizzarli, valutarli, definire la soluzione funzionale, progettare la soluzione tecnica ed il dettaglio delle componenti
  - Programmazione e test: scrivere i programmi secondo la progettazione ed eseguire i test ai veri livelli (unitari, funzionali, d'integrazione, di sistema)
  - Collaudo e rilascio: eseguire il collaudo finale e rilasciare la soluzione nell'ambiente di produzione



Ciclo a cascata (waterfall)



Ciclo iterativo



Algoritmo, Programma e Linguaggio di programmazione

- 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: insieme di istruzioni formali che traduce l'algoritmo ed è direttamente comprensibile, pertanto eseguibile, da parte di un elaboratore

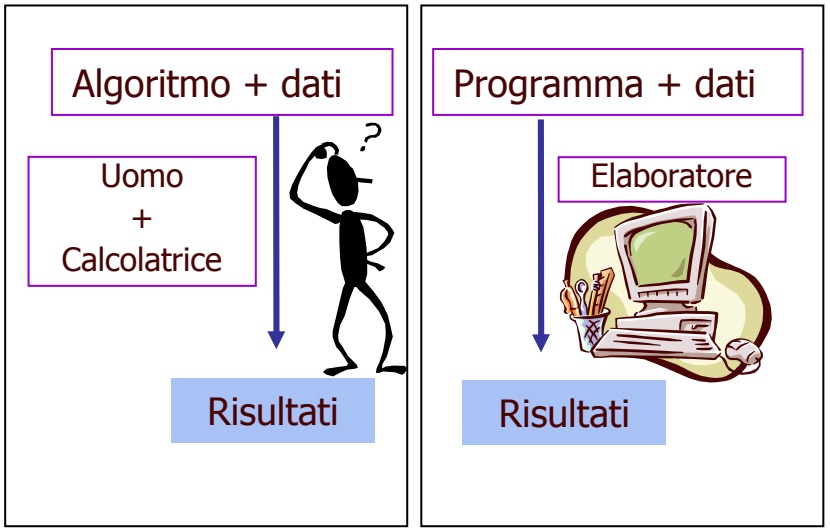
- Linguaggio di programmazione: linguaggio rigoroso che permette la formalizzazione di un algoritmo in un programma

Analisi e programmazione: Esempio

- 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

La programmazione

Analisi e programmazione

Algoritmi

Diagrammi a blocchi

Analisi strutturata

Costanti e variabili

Istruzioni

Vettori e matrici

Algoritmi iterativi

Algoritmi: definizione

- 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

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**: i passi successivi da eseguire devono essere definiti in modo univoco; 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"

La programmazione

Analisi e programmazione

Algoritmi

Diagrammi a blocchi

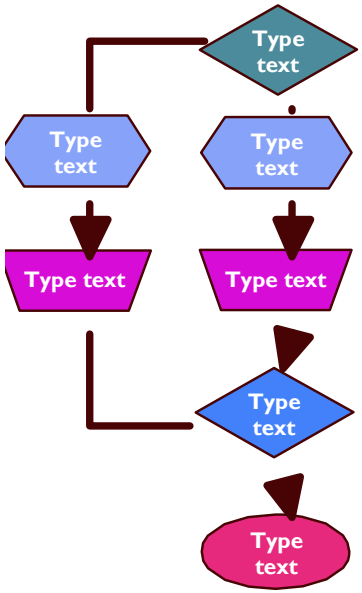
Analisi strutturata

Costanti e variabili

Istruzioni

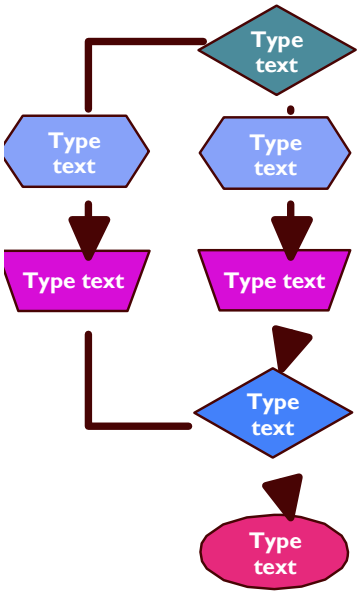
Vettori e matrici

Algoritmi iterativi



Cosa sono i diagrammi a blocchi

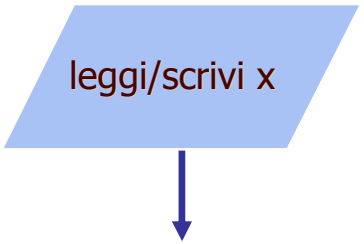
- Il linguaggio dei diagrammi a blocchi è un possibile formalismo per la descrizione di algoritmi
- Il diagramma a blocchi, o *flowchart*, è una rappresentazione grafica dell'algoritmo
- Un diagramma a blocchi descrive il flusso delle operazioni da eseguire per realizzare la trasformazione, definita nell'algoritmo, dai dati iniziali ai risultati
- Ogni istruzione dell'algoritmo è 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



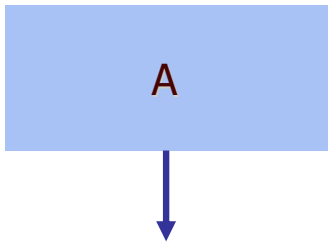
I tipi di diagrammi a blocchi

Blocchi elementari

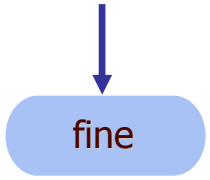
Blocco iniziale



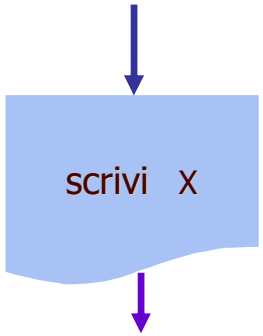
Blocco di lettura/scrittura



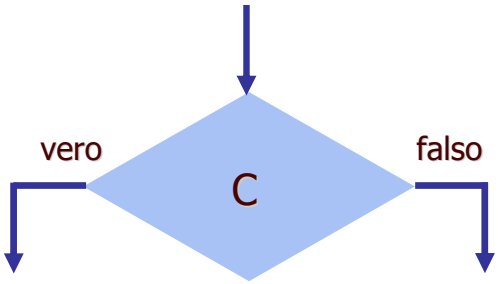
Blocco azione



Blocco finale



Blocco di scrittura



Blocco di controllo

Costituzione elementare dei diagrammi a blocchi

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

Condizioni dei diagrammi a blocchi

- L'insieme dei blocchi elementari che costituiscono 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 inserisce in un'altra freccia
  - ciascun blocco è raggiungibile dal blocco iniziale
  - il blocco finale è raggiungibile da qualsiasi altro blocco

La programmazione

Analisi e programmazione

Algoritmi

Diagrammi a blocchi

Analisi strutturata

Costanti e variabili

Istruzioni

Vettori e matrici

Algoritmi iterativi

Analisi strutturata

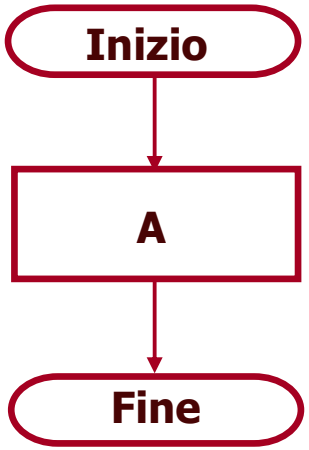
- I programmatori inesperti tendono ad "aggrovigliare" il programma introducendo numerosi salti privi di regole (*spaghetti program*)
- L' analisi strutturata favorisce, viceversa, la descrizione di algoritmi facilmente documentabili e comprensibili
- I blocchi di un diagramma a blocchi strutturato sono 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

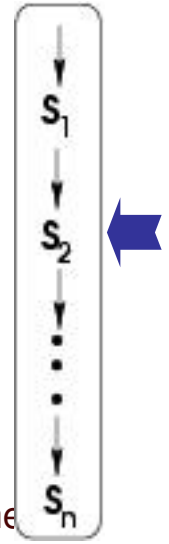
Che cos'è l'analisi strutturata

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

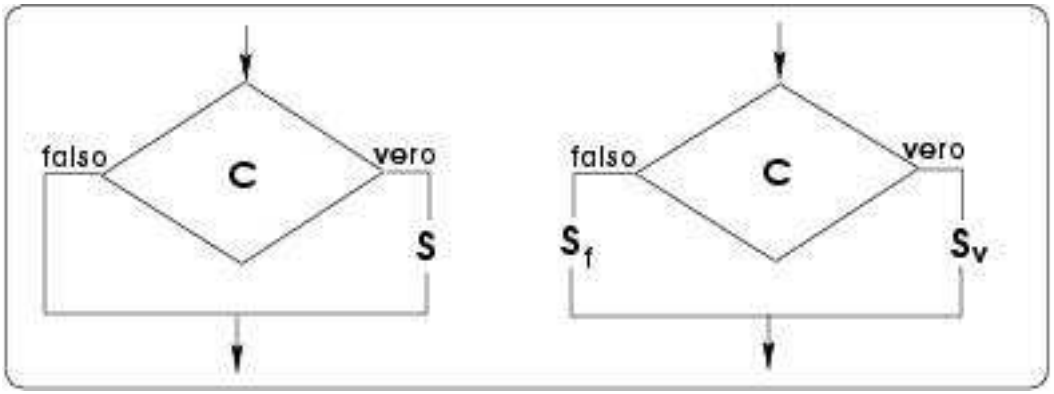




S_1, S_2,…, S_n sono schemi di flusso strutturati

Uno schema strutturato

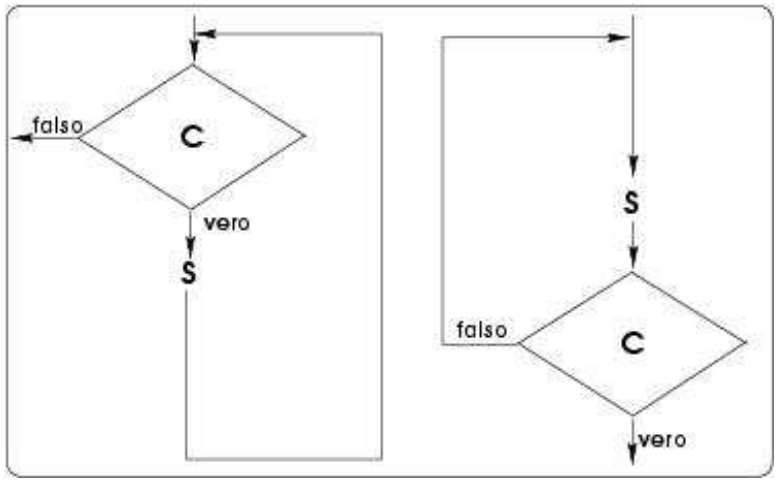
2) … è uno schema di selezione



- 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 $\mathbf{S_v}$ o $\mathbf{S_f}$, a seconda del valore di verità della condizione

Uno schema di iterazione

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

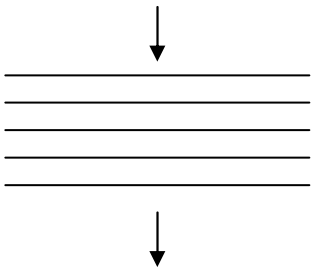
Schemi di sequenza aperti e chiusi

- Gli schemi di flusso sono aperti quando consentono una sola esecuzione di una sequenza di blocchi elementari, sono chiusi quando permettono più di un'iterazione 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

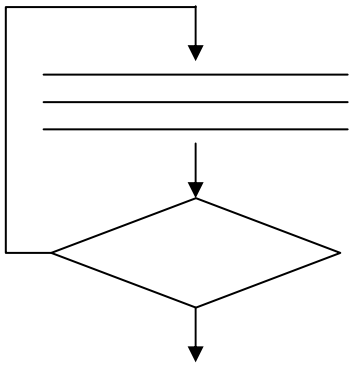
Teorema fondamentale della programmazione

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

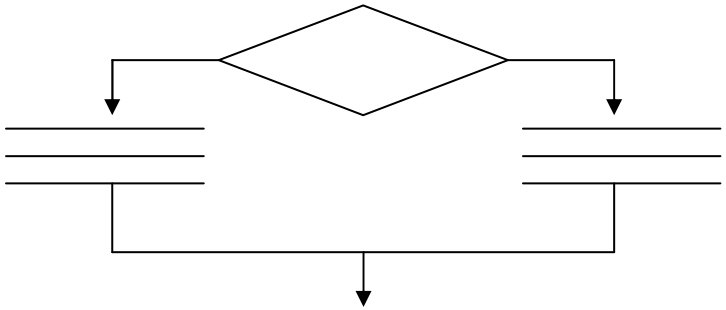
Sequenziale



Iterativa

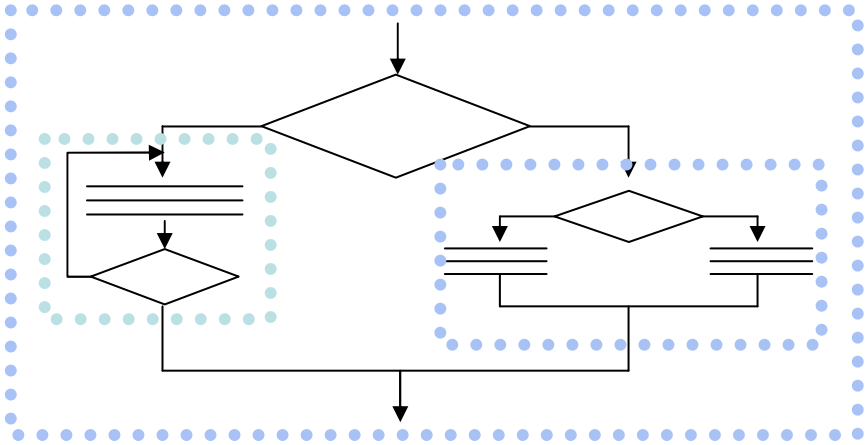


Condizionale

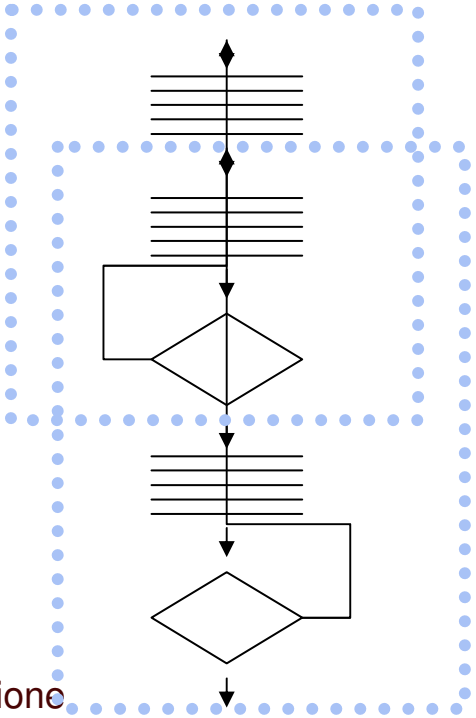


Strutture concatenate e nidificate

- Le tre strutture fondamentali possono essere *concatenate*, una di seguito all'altra, o *nidificate*, una dentro l'altra
- **Non possono** in nessun caso essere "intrecciate" o "accavallate"



Corretto



Sbagliato

La programmazione

Analisi e programmazione

Algoritmi

Diagrammi a blocchi

Analisi strutturata

Costanti e variabili

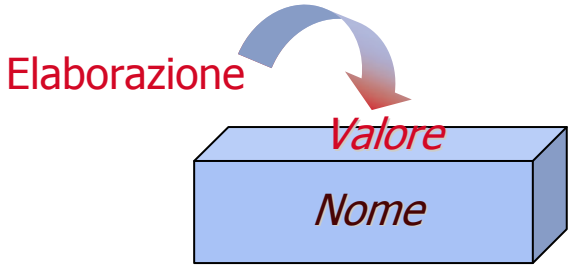
Istruzioni

Vettori e matrici

Algoritmi iterativi

Costanti e variabili

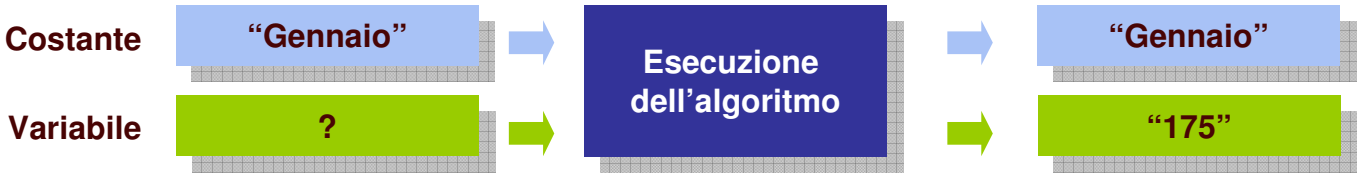
- I dati su cui opera un algoritmo sono costanti e variabili
  - Un dato è costante quando il suo valore non può essere aggiornato durante l'esecuzione dell'algoritmo (es.: una costante può essere un campo che contiene il numero di giorni in una settimana, "7")
  - Un dato è variabile quando il suo valore può essere modificato dall'elaborazione; è costituito da una coppia <*nome*, *valore*>: dove il "nome" rimane costante nel tempo ed il "valore" può dipendere dall'elaborazione (es.: una variabile può essere un campo dove andiamo a mettere il risultato di un'operazione aritmetica)



Rappresentazione di una "variabile"

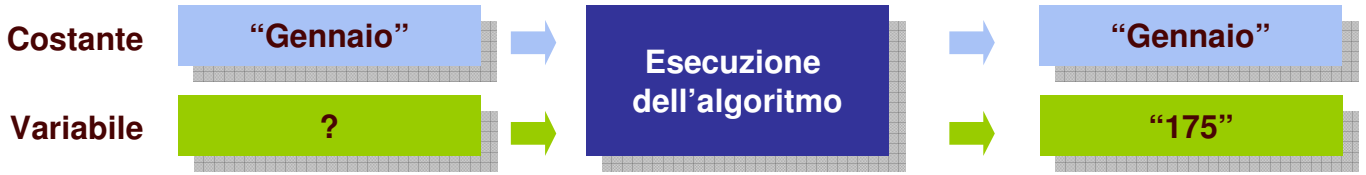
Contenuti delle costanti e delle variabili

- Una costante può contenere un valore che rientri nell'insieme di definizione del campo stesso (es.: un campo definito come "numerico" potrà contenere solo "numeri")
- Anche una variabile potrà contenere solo un valore che appartenga all'insieme di definizione (es.: "numerico" o "alfabetico")
- Una variabile <*nome, valore*> avrà un valore indeterminato all'inizio dell'elaborazione e acquisterà un valore ben specifico durante l'esecuzione dell'algoritmo



Assegnazione dei valori

- L'Assegnazione è un'istruzione che determina quale valore debba assumere una costante o una variabile
- L'assegnazione di un valore ad una "costante" è fatta all'inizio e generalmente non è più modificata durante l'esecuzione dell'algoritmo
- L'assegnazione di un valore ad una "variabile" è fatta durante l'esecuzione dell'algoritmo e dipenderà dal risultato dell'elaborazione compiuta

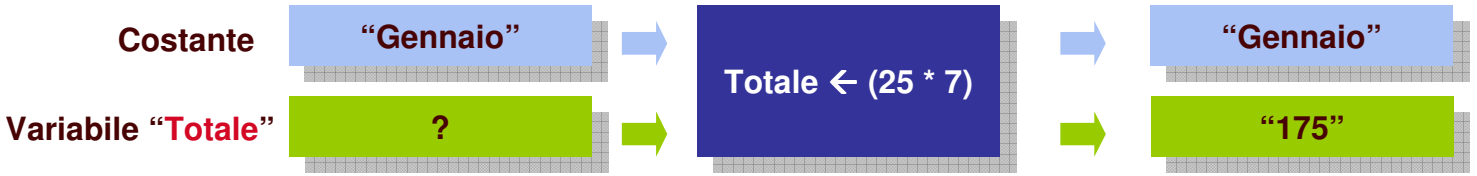


Come si rappresenta l'assegnazione

- L'assegnazione si rappresenta con il simbolo "←":

  nome di variabile ← espressione

  che si legge *assegna alla variabile "nome di variabile" il valore di "espressione"* ; l'espressione a destra di ← è costituita da variabili, costanti e operatori



Esempio di assegnazione

- Esempio:

La programmazione

Analisi e programmazione

Algoritmi

Diagrammi a blocchi

Analisi strutturata

Costanti e variabili

Istruzioni

Vettori e matrici

Algoritmi iterativi

Tipi di istruzioni

Tipi di istruzioni:

- 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
- 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

Esempi di tipi di istruzioni

Esempio: Calcolare la differenza tra due numeri se il primo è maggiore del secondo, altrimenti sommare i due numeri, e mostrare il risultato.

a)	"Inizio"	è un'istruzione di inizio esecuzione
b)	"Acquisire i due numeri a, b"	è un'istruzione di lettura (input)
c)	"Se a>b, calcola R=a-b"	è un'istruzione di controllo (l'istruzione di operativa "R=a-b" viene eseguita solo se a>b)
d)	"Se a>b vai a Stampa"	è un'istruzione di salto
e)	"R=a+b"	è un'istruzione operativa
f)	"Stampa: mostra il valore di R"	è un'istruzione di scrittura (output)
g)	"Fine"	è un'istruzione di fine esecuzione

Proposizioni e predicati

- Una proposizione è un costrutto linguistico del quale si può asserire o negare la veridicità

Esempio:

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 dal valore di alcune variabili

Esempio:

1) "*la variabile età è minore di 30*"
2) "*la variabile base è maggiore della variabile altezza*"

Valutazione di un predicato

- 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 sono espressi 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

Alcune regole

- Dato un predicato **p**, il predicato "**not p**" è detto opposto o negazione logica di p ed 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 (anche se uno dei due è vero)
- 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 uno fra i simboli 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

Alcuni esempi

Esempi:

not (base > altezza)

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

età > 30 and età < 50

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

base > altezza or base > 100

E' 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

La programmazione

Analisi e programmazione

Algoritmi

Diagrammi a blocchi

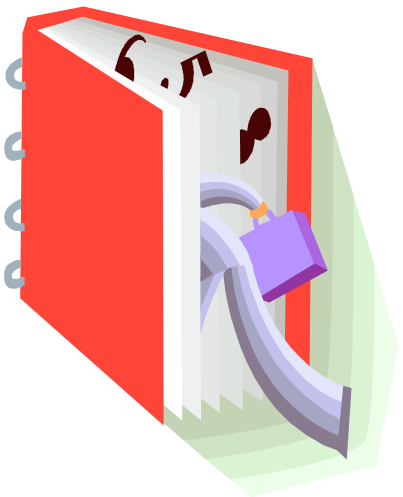
Analisi strutturata

Costanti e variabili

Istruzioni

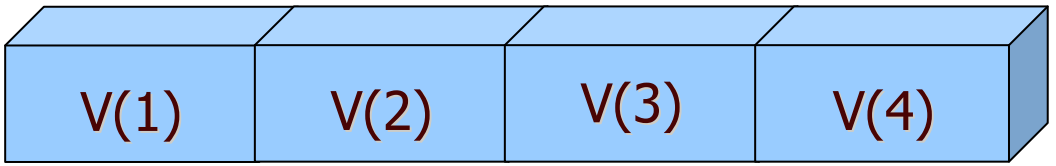
Vettori e matrici

Algoritmi iterativi



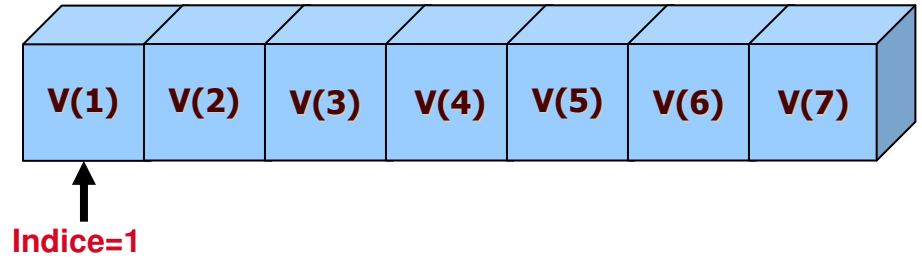
Vettori

- Le variabili, definite come coppie <*nome*, *valore* >, sono dette scalari
- Una coppia <*nome, insieme di valori* > è una variabile vettore e può essere immaginata come un contenitore diviso in scomparti; ciascun scomparto può contenere un valore, detto elemento o componente del vettore
- Ciascuna elemento o componente è individuato dalla sua posizione all'interno del vettore; è rappresentato tramite un numero racchiuso fra parentesi tonde (n): l'indice del vettore
- La dimensione di un vettore è il numero dei suoi elementi



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

Esempio di vettore



Variabile vettoriale S che rappresenta i giorni della settimana

E' costituita dai 7 elementi: S(1), S(2), S(3), S(4), S(5), S(6), S(7)

I valori di ciascun elemento sono:

S(1) = "Lunedì"

S(2) = "Martedì"

S(3) = "Mercoledì"

S(4) = "Giovedì"

S(5) = "Venerdì"

S(6) = "Sabato"

S(7) = "Domenica"

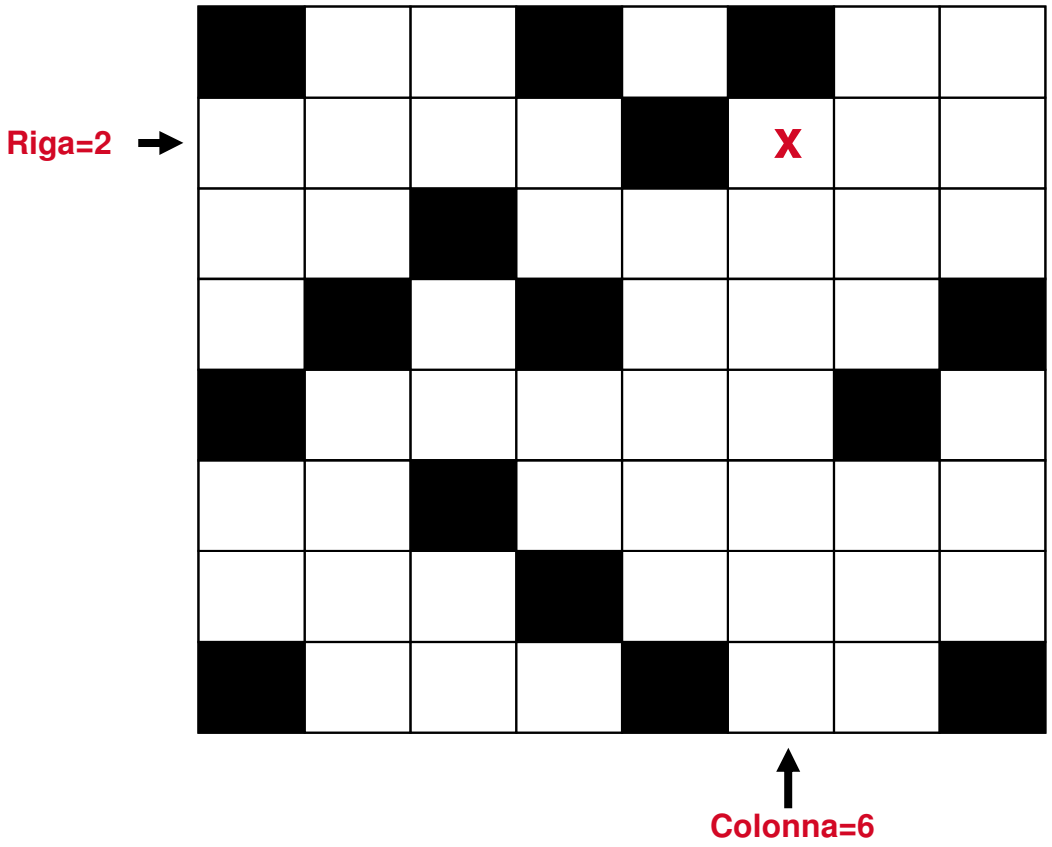
L'**Indice** indica il giorno della settimana

Matrici

- La matrice è un'estensione del concetto di vettore
- Una matrice è costituita da un insieme di valori, ciascuno dei quali è individuato per mezzo della sua posizione, espressa da più indici (es.: riga e colonna)
- Una matrice a "N" dimensioni ha "N" indici che determinano la posizione dei singoli elementi
- 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 (i=indice di riga e j=indice di colonna)

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

Esempio di matrice



La programmazione

Analisi e programmazione

Algoritmi

Diagrammi a blocchi

Analisi strutturata

Costanti e variabili

Istruzioni

Vettori e matrici

Algoritmi iterativi

Algoritmi iterativi

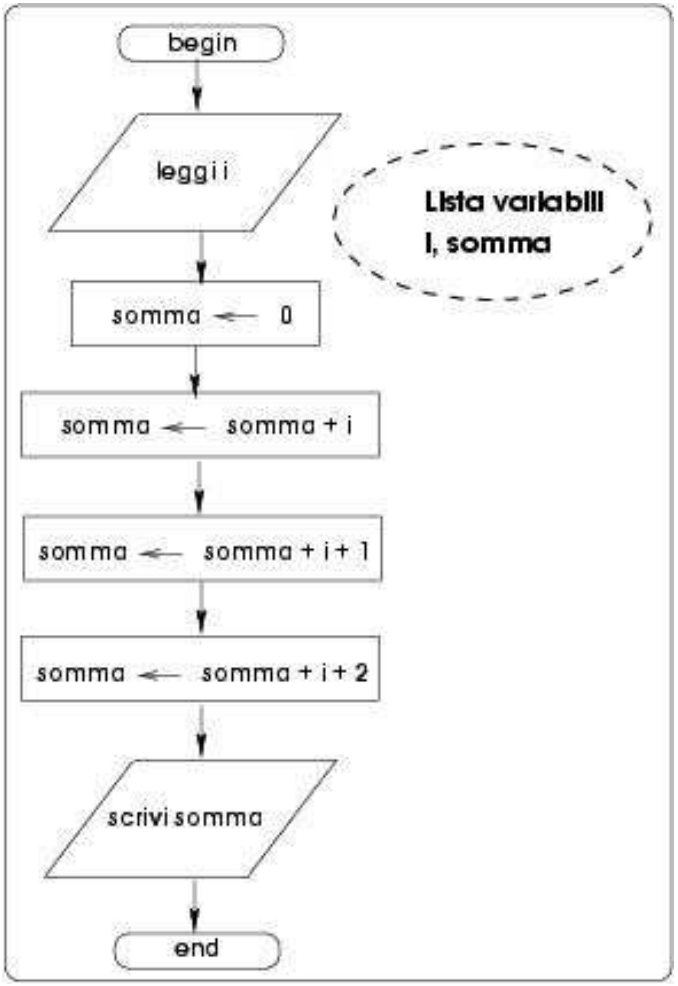
Problema:

Calcolare la somma di tre interi consecutivi a partire da un numero "i" fornito esternamente

Soluzione:

- Definiamo una variabile "somma" come un contenitore di somme parziali, finché non si ottiene la somma totale richiesta

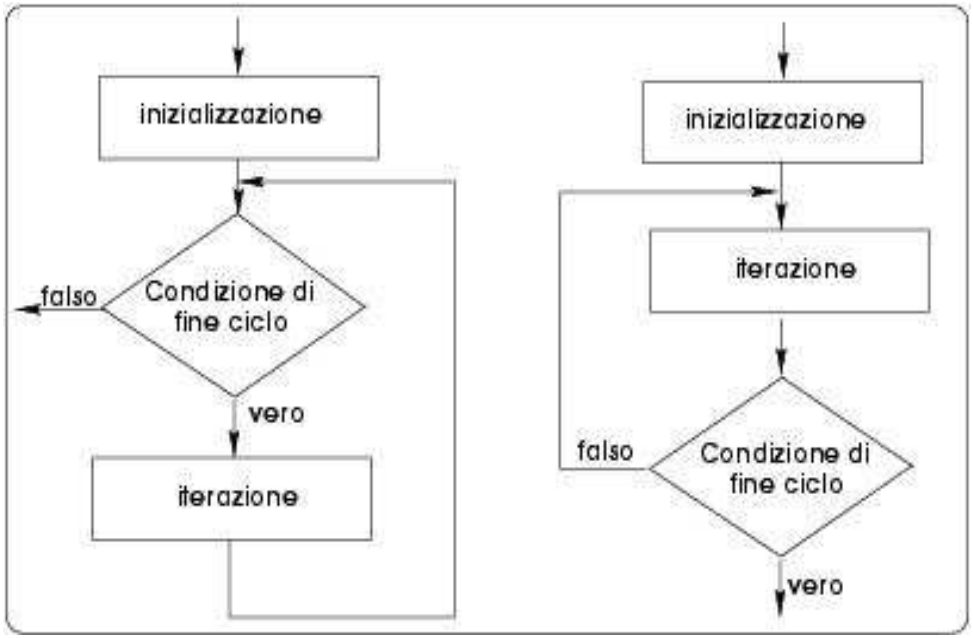
- La soluzione del problema è raggiunta eseguendo azioni simili per un numero opportuno di volte (Iterazione)



Il “ciclo” o “loop”

- Il ciclo o *loop* è uno schema di flusso che descrive, in modo conciso, situazioni in cui “un gruppo di operazioni” è ripetuto più volte

- La condizione di fine ciclo è verificata ogni volta che si esegue un ciclo; se la condizione assume valore “vero“ (oppure “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 di partenza alle variabili che controllano la condizione di fine ciclo



Ciclo con controllo in testa

Ciclo con controllo in coda

Un esempio di “ciclo” o “loop”

Problema: Calcolare la somma di tre interi consecutivi a partire da un numero “i” fornito esternamente

Soluzione:

- La fase di inizializzazione riguarda la somma e l’indice del ciclo
- Il controllo di fine ciclo viene effettuato in coda