IL CONCETTO DI ELABORAZIONE

- Il concetto astratto di elaborazione e` quello di trasformazione di dati:

$$\mathbf{Y = F(X)}$$

dove:

  - X e` l'insieme di dati *iniziali* o di *ingresso*
  - Y e` l'insieme di dati *finali* o di *uscita*
  - F e` una *regola* che fa corrispondere Y a X

IL CONCETTO DI ELABORAZIONE

- Elaborazione e`:
  - una somma
  - decidere se un numero e` pari
  - la risoluzione di una equazione di secondo grado
  - l'ordinamento di un elenco di nomi
  - la ricerca di una parola in un testo
  - la redazione e stampa di una lettera
  - ecc.

LA TRASFORMAZIONE "**F**"

- Soltanto in casi semplici si puo` ottenere mediante un'unica azione elaborativa (passo di elaborazione)
- In genere, e` composta da una sequenza piu` o meno lunga di elaborazioni intermedie
- La sequenza di azioni elaborative che definisce la trasformazione F e` detta **algoritmo**

IL CONCETTO DI ALGORITMO

- *"Sequenza finita di azioni elaborative che risolve automaticamente un problema indipendentemente da come esse sono espresse o comunicate al calcolatore che le esegue"*
  - opera su dati
  - e` un concetto astratto
  - deve essere comprensibile al suo esecutore
- un'azione elaborativa e` un'attivita` che produce una semplice trasformazione dei dati di un problema, trasportandoli da una parte ad un'altra del sistema di calcolo oppure calcolandone alcuni come *funzioni* di altri, dove il concetto di *funzione* e` il piu` vasto possibile

PROPRIETA` DEGLI ALGORITMI

- L'algoritmo e` caratterizzato da:
  - **garanzia di terminazione**: composto da un numero finito di passi elementari; le operazioni sono eseguite un numero finito di volte
  - **non-ambiguita`**: i risultati non variano in funzione della macchina/persona che esegue l'algoritmo (deterministico)
  - **realizzabilita`:** deve essere eseguibile con le risorse a disposizione

PROPRIETA` DEGLI ALGORITMI

- E' auspicabile che un algoritmo sia:
  - **corretto**

    perviene alla soluzione del compito senza difettare di alcun passo fondamentale
  - **efficiente**

    perviene alla soluzione del problema nel modo piu` veloce possibile e/o usando la minima quantita` di risorse fisiche (compatibilmente con la sua correttezza)

ALGORITMI e CALCOLATORI

- Legame fondamentale tra algoritmi e calcolatori elettronici:

  ☞ calcolatori come "esecutori di algoritmi"

- Gli algoritmi vengono descritti tramite **programmi**

  ☞ in base alle istruzioni contenute in un programma, un computer produce *dati* in output partendo dai *dati* ricevuti in input

  → Manipolazione di *dati*

IL CONCETTO DI PROGRAMMA

- *"Sequenza dettagliata di istruzioni elementari impartite ad un calcolatore per svolgere un compito"*
- Le istruzioni sono:
  - univoche (non ambigue)
  - ordinate (precisate in un certo ordine secondo il flusso dell'algoritmo)
  - scritte in un'opportuno linguaggio comprensibile al calcolatore (tra i molti che sono stati ideati allo scopo)

RISOLUZIONE DI PROBLEMI MEDIANTE CALCOLATORE

1 Comprensione del problema

2 Definizione della sequenza di passi che portano alla soluzione del problema ➔ **Algoritmo**
   - individuare i possibili ingressi
   - precisare le uscite
   - definire completamente e dettagliatamente la sequenza dei passi che portano alla soluzione
     (e` conveniente suddividere il problema in piccoli sottoproblemi)

3 Traduzione in istruzioni comprensibili al calcolatore (programmazione) ➔ **Programma**

4 Verifica e convalidazione

LINGUAGGI DI PROGRAMMAZIONE

- Consentono di scrivere gli algoritmi sotto forma di programmi che possano essere compresi dal calcolatore, che ne e` l'esecutore.
- Progettati per colmare il gap tra linguaggio macchina e linguaggio naturale (umano)
  - troppo ostico il **linguaggio macchina** (insieme dei comandi che la macchina e` in grado di eseguire)
  - troppo ricco, complesso e impreciso per una macchina il **linguaggio naturale**
- La fatica di tradurre il programma nel linguaggio della macchina e` affidata alla macchina stessa

LINGUAGGI DI PROGRAMMAZIONE

- Linguaggio assembler
- Linguaggi di alto livello (*imperativi*)

LINGUAGGIO ASSEMBLER

- Versione simbolica del codice macchina
- Operazioni e indirizzi di memoria indicate con nomi mnemonici invece che con sequenze di numeri binari
- Scritti in file testo
- Vengono tradotti in linguaggio macchina da appositi programmi detti *assemblatori*

LINGUAGGI D'ALTO LIVELLO

- Consentono direttamente ed efficientemente l'uso della macchina virtuale sottostante
- Le istruzioni sono essenzialmente assegnamenti da dare a locazioni di memoria
- piu` comprensibili
- piu` adatti a codificare gli algoritmi
- portabili
- possibilita` di riferirsi agli elementi del programma (celle di memoria, istruzioni, valori costanti) in maniera simbolica, cioe` attribuendo loro un nome (**identificatore**)
- possibilita` di esprimere le istruzioni e il controllo della sequenza della loro esecuzione in un modo piu` vicino al linguaggio naturale che al linguaggio macchina

LINGUAGGI D'ALTO LIVELLO

Alcuni esempi:

- **FORTRAN** (*FORmula TRANslator*, 1954-57)
- **COBOL** (*COmmon Business Oriented Language*, 1960)
- **Pascal** (1971)
- **BASIC** (*Beginner's All purpose Symbolic Instruction Code*, 1963)
- **C** (1974)

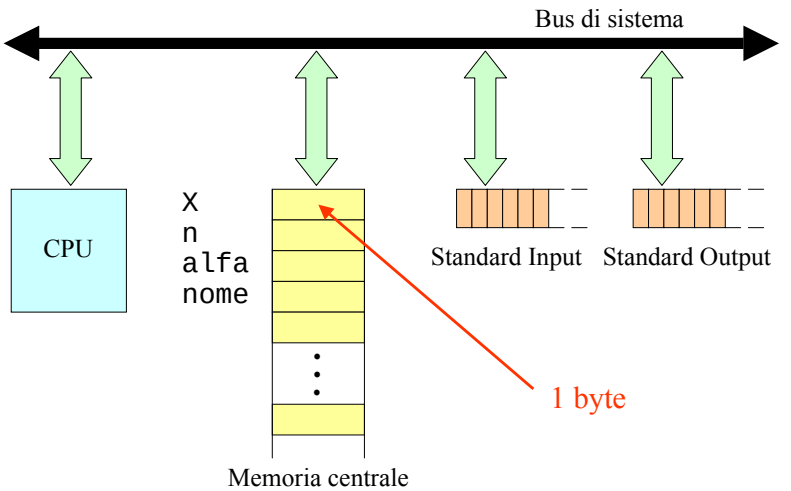
Nota: nel seguito verra` preso come riferimento il linguaggio "**C**"

COMPONENTI DI UN PROGRAMMA

- Dal punto di vista funzionale, un programma si compone di:
  - **variabili**

    memorizzano i valori usati durante l'esecuzione
  - **istruzioni**
    - "frasi del linguaggio di programmazione"
    - specificano le operazioni da effettuare sulle variabili
  - **commenti**
    - usati per documentare o descrivere un programma
    - non iniziano alcuna elaborazione

LA MACCHINA ASTRATTA



VARIABILI E TIPI

- Il concetto di **variabile** rappresenta un'astrazione della nozione di celle di memoria
- Caratterizzate da:
  - un **nome (identificatore simbolico)**

    successione di lettere e cifre, in cui al primo posto vi e` una lettera (non incluse nel "dizionario" del linguaggio)

    `a, x, tmp, var1, Var1, VAR1, f1_x`
  - **valore**

    che possono assumere attraverso le istruzioni del programma

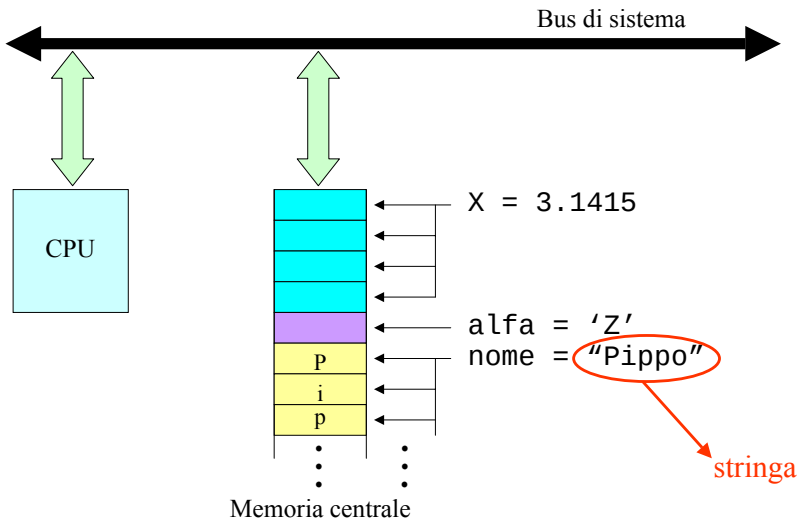
VARIABILI E TIPI

- Lo spazio di memoria necessario per rappresentare una variabile dipende dal **tipo** della variabile
- Esempi:

Tipo	Spazio in memoria
carattere	1 byte
intero	2 byte
reale	4 byte

- Il **tipo** specifica
  - l'insieme dei valori che la variabile puo` assumere
  - l'insieme di operazioni che possono essere applicate

LA MACCHINA ASTRATTA



ISTRUZIONI

- Dichiarazioni
- Istruzioni di assegnamento
- Istruzioni di ingresso uscita (I/O)
- Istruzioni composte

DICHIARAZIONI

- Particolare tipo di istruzioni
- Elencano le variabili che dovranno essere usate
- Stabiliscono il loro tipo ed, eventualmente, il loro valore iniziale
- Servono a far riservare una posizione di memoria del calcolatore per una variabile di cui viene fornito il nome
- Vengono inoltre fornite informazioni sulla natura delle variabili

ISTRUZIONE DI ASSEGNAMENTO

- Utilizzata per *assegnare* a una variabile il valore di un'espressione
- Consiste nel simbolo `=` preceduto dall'identificatore di una cella di memoria e seguito da un'**espressione** che definisce un valore
- La sua esecuzione comporta:
  - la valutazione dell'espressione eseguita sostituendo agli identificatori di variabile i valori delle celle corrispondenti al momento della valutazione
  - la memorizzazione del risultato nella variabile indicata a sinistra del simbolo `=`

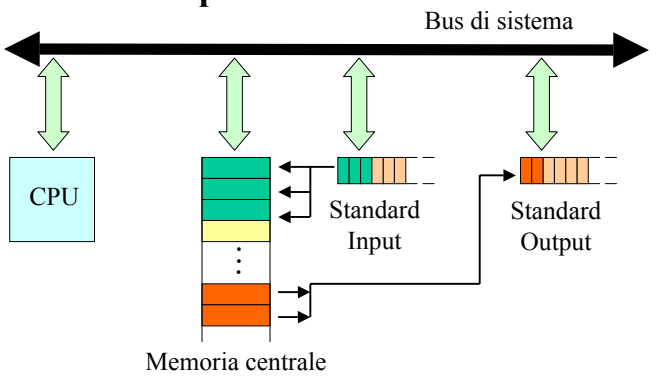
ESPRESSIONE

- L'**espressione** puo` essere costituita da valori costanti, identificatori di variabili o una loro combinazione ottenuta mediante operatori aritmetici (`+`, `-`, `*`, `/`, ecc.)
- Esempi:

```
x=23;
w='a';
y=z;
alfa=x+y;
r3=(alfa*43-xgg)*(delta-32*ig);
x=x+1;
```


ISTRUZIONI DI I/O

- Consentono di leggere o di scrivere il valore di una variabile dallo **Standard Input** o sullo **Standard Output**



ISTRUZIONI DI I/O

- **Istruzioni di Input**
  - I valori vengono spesso assegnati alle variabili mediante la loro immissione da tastiera, da file o altra sorgente
  - le istruzioni di Input provvedono ad un appropriato trasferimento dell'informazione
  - I dati devono generalmente apparire nello stesso ordine con cui sono richiesti
  - Tipicamente lo Standard Input e` la tastiera

ISTRUZIONI DI I/O

- **Istruzioni di Output**
  - permettono di stampare a video, o trasferire ad altri supporti di memoria piu` o meno permanenti i valori attuali delle variabili specificate
  - I valori trasferiti in Output sono generalmente
    organizzati in un certo formato (formattati)
    accompagnati da informazioni di identificazione
  - Tipicamente lo Standard Output e` il video

ISTRUZIONI COMPOSTE

- Producono effetti diversi a seconda che siano verificate o meno certe **condizioni** sul valore delle variabili
- Sono di due tipi
  - istruzioni di selezione
  - istruzioni iterative

} **Strutture di controllo del flusso** (dell'algoritmo)

Meccanismi atti a controllare la sequenza delle operazioni da applicare per l'esecuzione di un algoritmo

FLUSSO DEL PROGRAMMA

- Rappresenta l'ordine secondo cui singole istruzioni o gruppi di esse vengono eseguiti, come specificato dall'algoritmo
- Puo` essere:
  - sequenziale
  - condizionale
  - ripetitivo

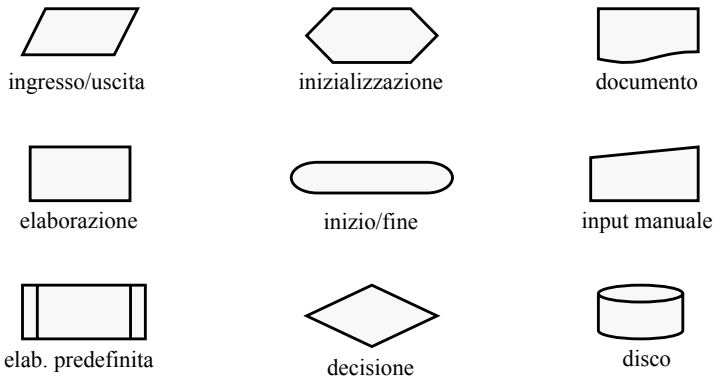
DIAGRAMMI DI FLUSSO (FLOW CHART)

- Rappresentazioni grafiche dei passi elementari dell'algoritmo; visione globale del problema
- Possono costituire uno strumento efficace per descrivere un algoritmo (piu` della descrizione a parole, troppo generica o appesantita da troppi dettagli)
- Le operazioni base sono 4

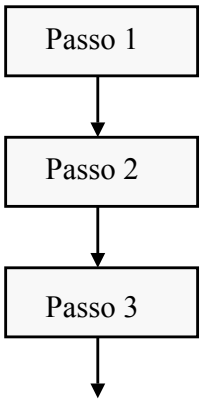
OPERAZIONI BASE

- Le operazioni primarie sono:
  - trasferimento di informazioni
    lettura dati, scrittura risultati, visualizzazione dati intermedi
  - esecuzione di calcoli
  - assunzione di decisioni
  - esecuzione di iterazioni
    ripetizione di sequenze di operazioni
- Sono rappresentate da forme geometriche diverse

SIMBOLI CONVENZIONALI

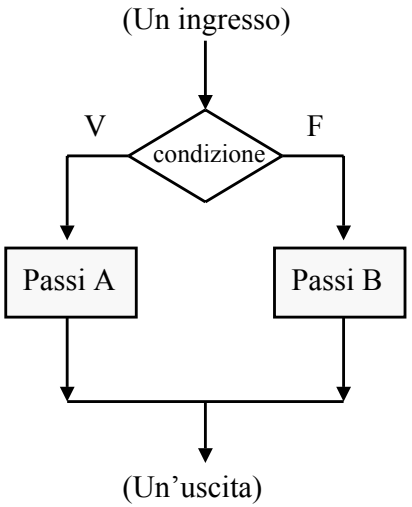


FLUSSO SEQUENZIALE



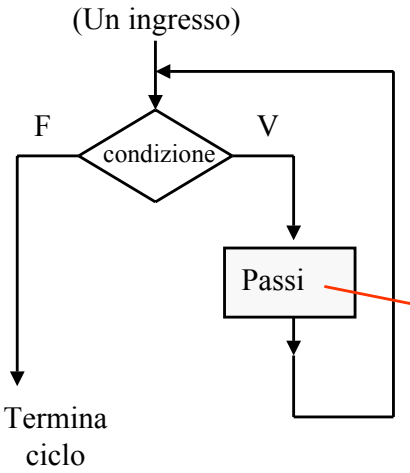
"Le istruzioni sono eseguite in successione ordinata senza che sia necessario una decisione su quale debba essere l'istruzione successiva"

FLUSSO CONDIZIONALE



"Se la condizione e` verificata allora segui la strada A; altrimenti segui la strada B"

FLUSSO RIPETITIVO



"Permette la ripetizione dell'esecuzione di una sequenza di passi di elaborazione ogni volta che una certa condizione e` verificata"

Deve includere un'istruzione di aggiornamento/modifica della condizione per garantire la fine del ciclo

CONDIZIONE

- Espressione il cui valore puo` essere **vero** o **falso**
- E` costruita mediante
  - operatori aritmetici (+, -, *, /)
  - operatori di relazione (==, !=, <, >, <=, >=)

==	uguale	>	maggiore
!=	diverso	<=	minore o uguale
<	minore	>=	maggiore o uguale

  - operatori logici (!, ||, &&)

NOT OR AND

OPERATORI LOGICI

NOT(*A*)

A	
F	**V**
V	**F**

V=vero; F= falso

A* OR *B

A \ *B*	F	V
F	**F**	**V**
V	**V**	**V**

A* AND *B

A \ *B*	F	V
F	**F**	**F**
V	**F**	**V**

Gli operatori OR e AND si applicano a due operandi, l'operatore NOT a uno solo

ESEMPI DI CONDIZIONI

- Esempi di condizioni sono i seguenti:

```
x == 0
(alfa > beta) && (x != 3)
!( (a+b)*3>x || a<c )
```

- La valutazione di un'espressione condizionale e` soggetta a determinate regole di precedenza
- **Per definizione il valore numerico di un'espressione logica o relazionale e` 1 se la relazione e` vera, 0 se e` falsa**

STRUTTURE DATI

- Programma = Algoritmo + **Strutture dati**
- L'elaborazione dell'informazione tramite calcolatore richiede che la stessa venga organizzata in **strutture** che si adattino
  - alle caratteristiche fisiche e alla logica di funzionamento del calcolatore
  - alla natura del problema da risolvere
- I **dati strutturati** rappresentano l'informazione come aggregazione di diverse componenti che raggruppano al loro interno informazioni piu' semplici

STRUTTURE DATI

- Esempi:
  - vettori e matrici
  - elenchi di nomi, di fatture, di archivi, …
- I linguaggi di alto livello permettono di trattare questo tipo di informazione in modo aderente alla sua organizzazione logica attraverso le **variabili strutturate** (= variabili che memorizzano diversi elementi informativi)

VETTORI

- Il vettore e` un insieme di elementi dello stesso tipo, individuabili singolarmente tramite un indice che indica la loro posizione rispetto al primo elemento

v = {v[0], v[1], …, v[i], …, v[N-1]}

elemento i-esimo

indice

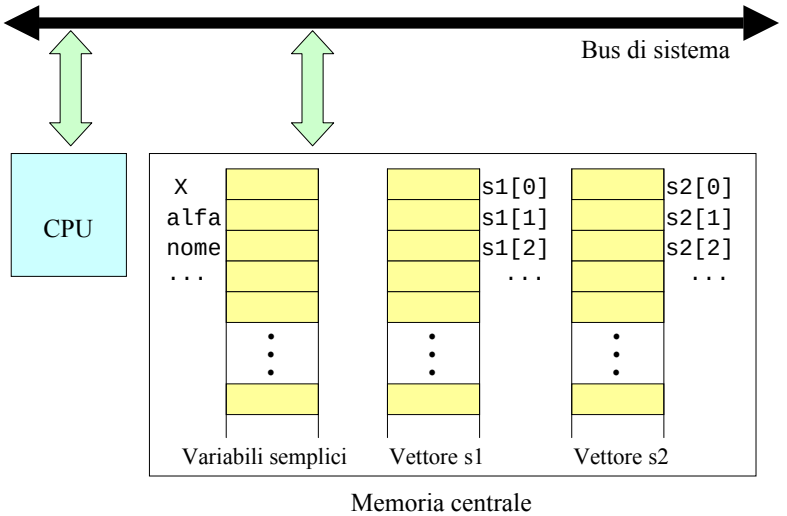
- gli indici sono rappresentati da interi

RAPPRESENTAZIONE DI VETTORI

- In locazioni di memoria **contigue**
- Si definisce
  - l'indirizzo della prima locazione: $ind(el_0)$
  - dimensione di ogni elemento: $size(el)$
- Accesso all'elemento i-esimo:

$$ind(el_i)=ind(el_0)+i*size(el)$$

RAPPRESENTAZIONE DI VETTORI



VETTORI A PIU` DIMENSIONI

- Gli elementi sono individuati da *n* indici

$$v[i_1, i_2, \ldots, i_n]$$

- Vettore bidimensionale → **matrice** M[i,j]
  - tabella in cui ogni elemento e` individuabile da una coppia di indici (i,j)
    [i = indice di riga; j = indice di colonna]
  - accesso all'elemento (i,j)-esimo:

$$ind(el_{i,j})=ind(el_{0,0})+(j+i*N)*size(el)$$

N = numero totale di colonne

- Facile l'estensione a piu` di due dimensioni

RAPPRESENTAZIONE DEI VETTORI

- Per memorizzare vettori e matrici vengono riservate zone di memoria calcolando le dimensioni necessarie in base al numero di elementi specificato al momento della loro dichiarazione nel programma
- Tali dimensioni non possono essere modificate durante l'esecuzione
- Le istruzioni per accedere a elementi di queste strutture non possono far riferimento a indici che ne superino la dimensione massima

LIMITI DEI VETTORI

- Il vettore e` una struttura di semplice utilizzo per certe classi di problemi, *ma* ha dei limiti:
  - le dimensioni non possono essere modificate durante l'esecuzione
  - se si deve inserire un ulteriore elemento in posizione intermedia, cio` non e` possibile se non sovrascrivendo un altro elemento o spostando in avanti (ammesso che vi sia spazio) uno a uno gli altri elementi
- Esistono strutture dati piu` complesse che superano questi limiti
  - liste, code e pile, alberi e grafi

PROGRAMMAZIONE TOP-DOWN

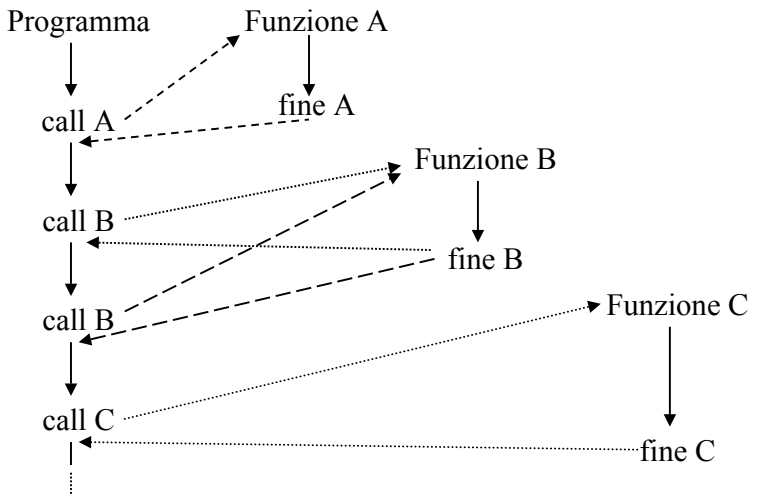
- Scomposizione iterativa del problema in sottoproblemi
- I sottoproblemi devono essere indipendenti ed avere interfacce ben definite
- Visibilita` dei dettagli di ogni sottoproblema

☞ Il programma consiste di piu` parti, che possono essere chiamate *funzioni (procedure, sottoprogrammi, subroutine o moduli)*

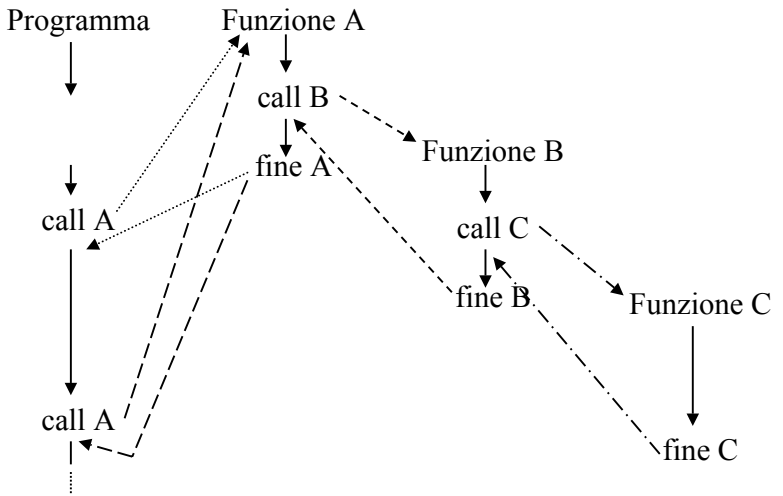
FUNZIONI

- In generale, si tratta di parti del programma che possono essere attivate ("richiamate") piu` volte
- Il meccanismo di chiamata si basa su un'istruzione in linguaggio macchina che prende il nome di *call*
  - memorizza l'indirizzo dell'istruzione del programma a cui si e` arrivati e fa saltare l'esecuzione al punto in cui e` memorizzata la funzione chiamata
  - al termine di questa viene eseguita un'altra istruzione particolare (*return*) che ripristina l'indirizzo dell'istruzione immediatamente successiva alla *call*
  - l'esecuzione riprende da dove era stata interrotta

FUNZIONI



FUNZIONI



COMUNICAZIONE CON LE FUNZIONI

- Attraverso **variabili globali**
  - visibili in tutto il programma
  - contrapposte alle **variabili locali** dichiarate localmente alla funzione e pertanto visibili solo al suo interno
- Attraverso **parametri**, detti **argomenti**, che assumono valori assegnati dal programma chiamante
  - e` possibile riutilizzare una funzione per manipolare dati diversi, contenuti in diverse variabili [ ad es. `ordina(M)`]
  - le funzioni possono, inoltre, restituire al programma chiamante un valore risultato dell'elaborazione [`A=f(x)`]

SOTTOPROGRAMMI INTERNI

- Sono listati assieme al programma principale
- operano prevalentemente su variabili globali
- non e` facile che queste procedure possano essere utili altrove

```
Programma principale:
      . . .
      Richiamo al sottoprogramma 1
      Richiamo al sottoprogramma 2
      . . .
Sottoprogramma 1:
      . . .
End
Sottoprogramma 2:
      . . .
End
Fine del programma principale
```


SOTTOPROGRAMMI ESTERNI

- E` un programma indipendente, compilato e provato separatamente
- non puo` operare direttamente sulle variabili globali
- disponibile per un uso generalizzato → **librerie**

```
Programma principale:
      . . .
      Richiamo al sottoprogramma(A,B,C)
      . . .
Fine del programma principale
Sottoprogramma(X,Y,Z):
      . . .
End
```

argomenti

parametri

FUNZIONI

Riassumendo:

- rappresentano un elemento fondamentale nella programmazione
  - permettono di suddividere il programma in piu` parti (ciascuna corrispondente alla soluzione di un particolare sottoproblema)
- consentono di riutilizzare il codice gia` scritto e anche di richiamare codice scritto da altri
  - tutti i compilatori dispongono di **librerie** di funzioni per le operazioni piu` comuni (input/output, funzioni matematiche, ecc.)

FORMALISMI PER LA CODIFICA DI ALGORITMI

- **Pseudocodifica**:
  - utilizza uno pseudolinguaggio il piu` possibile vicino a quello naturale
  - facilita` di traduzione in un linguaggio di programmazione strutturato
- **Diagramma di flusso** (*flow-chart*):
  - utilizza un numero limitato di simboli grafici connessi da archi orientati (all'interno dei simboli grafici vengono descritte le singole operazioni elementari)
  - evidenzia graficamente il flusso delle istruzioni
  - puo` risultare difficile tradurlo in un linguaggio strutturato

CARATTERISTICHE DEGLI ALGORITMI

- Nel valutare quale algoritmo sia migliore per portare alla soluzione di un certo Pb, occorre considerare
  - il tempo necessario per la sua esecuzione in rapporto alla quantita` di dati in ingresso
  - l'occupazione di memoria
  - la sua facilita` di comprensione
    (facilita` di codifica e correzione)

COMPLESSITA` DEGLI ALGORITMI

- *T(n)* ➔ **tempo di esecuzione** di un programma su un input di dimensione *n*

- *O(f(n))* ➔ **ordine di complessita`**
  - relazione tra l'ordine di grandezza del numero di istruzioni eseguite e il numero di dati in ingresso

  $O(n)$ ➔ crescita lineare delle istruzioni con i dati

  $O(n^2)$ ➔ crescita quadratica

  $O(2^n)$ ➔ crescita esponenziale