



# Sviluppo di programmi

---

- Per la costruzione di un programma conviene:
  1. condurre un'analisi del problema da risolvere
  2. elaborare un algoritmo della soluzione rappresentato in un linguaggio adatto alla comprensione degli esseri umani ma abbastanza vicino ai linguaggi di programmazione
  3. produrre un programma scritto nel linguaggio di programmazione scelto



# Sviluppo di programmi

---

- FASE 1: Dare un nome al problema partendo dall'analisi del problema
- FASE 2: Scrivere la specifica funzionale
- FASE 3: Scrittura dell'algoritmo
  - FASE 3.1: Introduzione dei contenitori di dati (variabili) necessarie e delle relative operazioni elementari
  - FASE 3.2: Specifica di un diagramma di flusso ( o di uno pseudo codice) che descrive in modo preciso e non ambiguo la sequenza di operazioni da eseguire
- FASE 4: Traduzione del diagramma di flusso ( o dello pseudo codice) in un programma in un linguaggio di programmazione



# Sviluppo di programmi

---

- FASE 1: Dare un nome al problema partendo dall'analisi del problema
- FASE 2: Scrivere la specifica funzionale
- FASE 3: Scrittura dell'algoritmo
  - FASE 3.1: Introduzione dei contenitori di dati (variabili) necessarie e delle relative operazioni elementari
  - FASE 3.2: Specifica di un diagramma di flusso ( o di uno pseudo codice) che descrive in modo preciso e non ambiguo la sequenza di operazioni da eseguire
- FASE 4: Traduzione del diagramma di flusso ( o dello pseudo codice) in un programma in un linguaggio di programmazione



## FASE 1: problema e sua analisi

---

- L'analisi di un problema è la primissima cosa da fare quando si è di fronte alla richiesta di risolverlo
- Bisogna capire bene il problema prima di pensare di risolverlo!
- L'analisi del problema deve fornire come prodotto:
  - un nome ed una breve descrizione di cosa si vuol fare
  - un elenco di requisiti, ovvero, di richieste, vincoli che il programma deve soddisfare



## Esempio di Analisi del problema: $ax^2+bx+c$

---

- Problema: CALCOLO RADICI
- Descrizione: calcolare le soluzioni reali di una equazione di secondo grado
- Requisiti: l'equazione può avere nessuna soluzione reale, due soluzioni coincidenti, o due soluzioni distinte. Si devono, in questi casi, evidenziare il messaggio
  - nessuna radice
  - radici coincidenti  $r$  (dove  $r$  è il valore delle radici)
  - radici distinte  $r_1$  ed  $r_2$  (dove  $r_1$  ed  $r_2$  sono il valore delle radici)



# Sviluppo di programmi

---

- FASE 1: Dare un nome al problema partendo dall'analisi del problema
- **FASE 2: Scrivere la specifica funzionale**
- FASE 3: Scrittura dell'algoritmo
  - FASE 3.1: Introduzione dei contenitori di dati (variabili) necessarie e delle relative operazioni elementari
  - FASE 3.2: Specifica di un diagramma di flusso ( o di uno pseudo codice) che descrive in modo preciso e non ambiguo la sequenza di operazioni da eseguire
- FASE 4: Traduzione del diagramma di flusso ( o dello pseudo codice) in un programma in un linguaggio di programmazione



## FASE 2: specifica funzionale

---

- Una specifica funzionale descrive:
  - quali sono i dati iniziale, ovvero i dati di input (ingresso) da elaborare
  - qual è il risultato atteso in funzione dei dati in ingresso (output, uscita dell'algoritmo)



## Esempio di specifica funzionale

---

### CALCOLO RADICI: specifica funzionale

- INPUT: i numeri reali  $a, b, c$  che sono i coefficienti dell'equazione da risolvere
- OUTPUT: i messaggi
  - "nessuna radice reale"
  - " $x_1 = x_2 = r$ " se l'equazione ha due radici reali coincidenti pari ad  $r$
  - " $x_1 = r_1, x_2 = r_2$ " se l'equazione ha due radici reali distinte pari ad  $r_1$  ed  $r_2$



# Sviluppo di programmi

---

- FASE 1: Dare un nome al problema partendo dall'analisi del problema
- FASE 2: Scrivere la specifica funzionale
- **FASE 3: Scrittura dell'algoritmo**
  - FASE 3.1: Introduzione dei contenitori di dati (variabili) necessarie e delle relative operazioni elementari
  - FASE 3.2: Specifica di un diagramma di flusso ( o di uno pseudo codice) che descrive in modo preciso e non ambiguo la sequenza di operazioni da eseguire
- FASE 4: Traduzione del diagramma di flusso ( o dello pseudo codice) in un programma in un linguaggio di programmazione



## FASE 3: scrittura dell'algoritmo

---

- Descrizione dell'algoritmo, ovvero, dei passi da eseguire per giungere al risultato di uscita partendo dai dati in input
- La prima stesura non necessariamente deve essere completa di ogni dettaglio. in genere, si procede per raffinamenti successivi



## Esempio di stesura preliminare di algoritmi

---

CALCOLO RADICI: stesura preliminare dell'algoritmo

- risolvo il problema calcolando il discriminante (delta) dell'equazione
  - analizzo i vari casi di delta
    - $< 0$
    - $= 0$
    - $> 0$
  - costruisco per ognuno di questi casi il messaggio da produrre in output
- 
- Successivamente si definiscono le variabili (contenitori di dati) coinvolte e dettaglio la specifica dell'algoritmo usando un diagramma di flusso o lo pseudo codice.



# Sviluppo di programmi

---

- FASE 1: Dare un nome al problema partendo dall'analisi del problema
- FASE 2: Scrivere la specifica funzionale
- **FASE 3: Scrittura dell'algoritmo**
  - FASE 3.1: Introduzione delle variabili (contenitori di dati) necessarie e delle relative operazioni elementari
  - FASE 3.2: Specifica di un diagramma di flusso ( o di uno pseudo codice) che descrive in modo preciso e non ambiguo la sequenza di operazioni da eseguire
- FASE 4: Traduzione del diagramma di flusso ( o dello pseudo codice) in un programma in un linguaggio di programmazione



## FASE 3.1: definizione delle variabili

---

- Ogni algoritmo deve tenere traccia (memorizzare) i valori dei dati in input, dei risultati in output, e dei valori intermedi calcolati durante l'esecuzione dei passi
- A questo fine si usano delle **variabili** (contenitori di dati). Ogni variabile ha un nome associato al dato (**valore**) memorizzato.
- Una variabile è una astrazione (semplificazione) della nozione di area di memoria (una o più celle della RAM, un registro generale) contenente i bit rappresentanti i dati
- I dati contenuti in una variabile hanno un **tipo** (si dice che una variabile è di un certo tipo) che caratterizza un insieme di elementi e le operazioni che sono possibili su di essi



## FASE 3.1: definizione delle variabili

---

- Ogni variabile memorizza il valore di un dato
- Ogni dato appartiene ad un tipo di dato
- Un tipo di dato è: un insieme di elementi, rappresentabili in modo finito, dotato di operazioni primitive su di esso
- **ESEMPIO:** il tipo di dato intero
  - è l'insieme dei numeri interi, sequenze di cifre, con segno
  - dotato di operazioni primitive ed effettivamente eseguibili come somma (+), differenza (-), prodotto (\*), divisione intera (/), resto.



## FASE 3.1: definizione delle variabili

---

- Le variabili da usare per memorizzare i valori dei dati intermedi dipendono dall'algoritmo
- All'inizio della stesura di un algoritmo difficilmente si è già a conoscenza di quante e quali variabili si dovranno usare
- Ad ogni raffinamento dell'algoritmo si possono aggiungere o togliere variabili fino ad arrivare all'uso corretto di ognuna



## Esempio di definizione di variabili

---

### CALCOLO RADICI: variabili usate

- ci vogliono tante variabili quanti sono i dati in input. In questo caso ci vogliono tre variabili che chiamiamo **a,b,c** per i coefficienti dell'equazione
- ci vogliono tante variabili quanti sono i dati in uscita. In questo caso, servono due variabili **r1,r2** per il valore delle radici dell'equazione
- ci vogliono alcune variabili per tenere conto di dati intermedi da calcolare e modificare per ottenere la soluzione. In questo caso, potremmo usare una variabile per memorizzare il valore del discriminante. La variabile la chiamiamo **delta**
- Il tipo di tutte queste variabili è float (numeri reali).



# Sviluppo di programmi

---

- FASE 1: Dare un nome al problema partendo dall'analisi del problema
- FASE 2: Scrivere la specifica funzionale
- **FASE 3: Scrittura dell'algoritmo**
  - FASE 3.1: Introduzione delle variabili (contenitori di dati) necessarie e delle relative operazioni elementari
  - FASE 3.2: Specifica di un diagramma di flusso ( o di uno pseudo codice) che descrive in modo preciso e non ambiguo la sequenza di operazioni da eseguire
- FASE 4: Traduzione del diagramma di flusso ( o dello pseudo codice) in un programma in un linguaggio di programmazione



## Descrizione di un algoritmo

---

- Si descrive un algoritmo cercando di sintetizzare il più possibile la sua sequenza di passi;
- Non si utilizza un linguaggio di programmazione specifico, ma è meglio utilizzare qualcosa di più generale;
- Pseudo-codici o Diagrammi di Flusso



## Descrizione di un algoritmo

---

- Un algoritmo descrive due tipi fondamentali di operazioni:
  - calcoli ottenibili tramite le operazioni primitive su tipi di dato (valutazione di espressioni)
  - azioni che consistono nella modifica del valore associato alle variabili



## Descrizione di un algoritmo

---

- Un algoritmo descrive due tipi fondamentali di operazioni:
  - calcoli ottenibili tramite le operazioni primitive su tipi di dato (valutazione di espressioni)
  - azioni che consistono nella modifica del valore associato alle variabili



## Calcoli: espressioni valutabili

---

- L'esecutore dell'algoritmo valuta espressioni dove gli elementi possono essere variabili o valori costanti
  - $3 * X + \sin(Y * \text{sqrt}(z))$
  - $b*b-4*a*c$
  - $h / j$
  - $k / 2.23$
  - 192



## Calcoli: espressioni booleane (logiche)

---

- Ci sono anche particolari tipi di espressioni chiamate booleane o logiche
- Sono espressioni che possono dare origine a due soli possibili valori: VERO o FALSO
- Una proposizione della quale dobbiamo valutare se è vera o falsa
- Per costruire un'espressione logica si possono usare le seguenti relazioni:
  - = (uguaglianza);
  - < (minore); <= (minore o uguale);
  - > (maggiore); >= (maggiore o uguale);
  - <> (diverso);



## Esempi di espressioni booleane (logiche)

---

- A seconda del valore delle variabili l'espressione booleana può essere vera o falsa
  - $b*b-4*a*c = 0$
  - $h / j \lt 2.4$
  - $k / 2.23 \gt p$
  - $192 \leq 12$



## Calcoli: espressioni logiche complesse - NOT

- Possiamo costruire espressioni booleane più complesse grazie ad alcune regole della cosiddetta algebra booleana;
- Se **A** è un'espressione logica, anche **Not A** (la negazione logica di A) lo è

<b>A</b>	<b>Not A</b>
Vero	Falso
Falso	Vero



## Congiunzione (AND)

---

- Se  $A$  e  $B$  sono espressioni logiche, anche  $A$  and  $B$  lo è;

<b>A</b>	<b>B</b>	<b>A and B</b>
Vero	Vero	Vero
Vero	Falso	Falso
Falso	Vero	Falso
Falso	Falso	Falso



## Disgiunzione (OR)

---

- Se  $A$  e  $B$  sono espressioni logiche, anche  $A$  or  $B$  lo è;

<b>A</b>	<b>B</b>	<b>A or B</b>
Vero	Vero	Vero
Vero	Falso	Vero
Falso	Vero	Vero
Falso	Falso	Falso



## Descrizione di un algoritmo

---

- Un algoritmo descrive due tipi fondamentali di operazioni:
  - calcoli ottenibili tramite le operazioni primitive su tipi di dato (valutazione di espressioni)
  - azioni che consistono nella modifica del valore associato alle variabili



## Azioni: modifica valori delle variabili

---

- L'esecutore di un algoritmo deve poter compiere azioni che hanno un effetto sui dati gestiti
- Le azioni più semplici sono **assegnamenti a variabili**
- Un assegnamento ad una variabile si denota con

VARIABILE := ESPRESSIONE

- con il seguente significato:
  1. l'esecutore valuta l'espressione a sinistra del :=
  2. e poi modifica il valore della variabile con il risultato della valutazione dell'espressione



## Esempi di azioni: assegnamenti a variabili

- L'esecutore valuta espressioni dove gli elementi possono essere variabili o valori costanti e se ne assegna il valore ad una variabile
  - `pippo := 3 * X + sin (Y * sqrt(z))`
  - `delta := b*b-4*a*c`
  - `a := h / j`
  - `W := k / 2.23`
  - `M := 192`
  - `T := p`
  - `x := x + 1`
- Una variabile **non può essere "vuota"**: ad una variabile **deve sempre essere associato un valore**
- Il valore contenuto in una variabile è l'ultimo assegnatole e resta inalterato finché una successiva assegnazione non modifica il valore stesso