*Dipartimento di Ingegneria dell'Informazione*
Università degli Studi di Parma

# Fondamenti di Informatica

*Laurea in*
*Ingegneria Civile e Ingegneria per l'ambiente e il territorio*

Algoritmi e Programmazione (in C)

*Stefano Cagnoni e Monica Mordonini*

## Il problema di fondo

- **Descrizione di un problema**

☑**individuazione di una soluzione**
  - Quale è il giusto punto di partenza? Cioè, di quali dati abbiamo bisogno ?
  - Quali metodologie o tecniche utilizzare?
  - In quale ordine eseguire le operazioni consentite da tali tecniche ?



## Algoritmo

- Dall'arabo al-Khuwarizmi, a sua volta dal greco arithmós
- Un algoritmo è un metodo generale che risolve in un tempo finito e con una sequenza finita di passi qualsiasi istanza di un dato problema di elaborazione.



## Algoritmo

- E' possibile "trovare" algoritmi anche per la risoluzione di problemi non strettamente informatici

Esempi:
  - Spiegare un percorso stradale
  - Istruzioni per il montaggio di un mobile
  - Istruzioni per la realizzazione di una torta

- Un algoritmo può non essere l'unica soluzione al problema



## Codifica di un algoritmo

- Fase di descrizione (scrittura) di un algoritmo attraverso un insieme ordinato di codici (istruzioni), appartenenti a un qualche linguaggio di programmazione, che specificano le azioni da compiere
- Il prodotto della codifica è un *programma*



## Programma

- Testo scritto in accordo alla sintassi e alla semantica di un linguaggio di programmazione
- Un programma può non essere un algoritmo (basta che la sequenza di mosse non sia finita cioè che il programma non termini)...
- ... e tuttavia può essere molto utile (es. gestione semafori)
- Un programma rappresenta l'insieme delle istruzioni che descrivono un processo espresse in un qualche linguaggio
- Un **processo** trasforma un insieme di dati iniziali nei risultati finali mediante una successione di **azioni elementari**

## Esecuzione

- L'esecuzione delle azioni nell'ordine specificato dall'algoritmo consente di ottenere i risultati che risolvono il problema a partire dai dati in ingresso
- **Problema**: ☑ algoritmo ☑ programma

*Metodo risolutivo* *Codifica in un linguaggio di programmazione*



## Algoritmo

- Un algoritmo deve avere le seguenti **proprietà**:
  - **Finitezza**: composto da un numero finito di passi elementari.
  - **Non ambiguità** (determinismo): i risultati non variano in funzione della macchina/persona che esegue l'algoritmo.
  - **Realizzabilità:** deve essere eseguibile con le risorse a disposizione.
  - **Efficienza** (auspicabile): eseguire il numero minimo di operazioni



## Algoritmo

- Per definire un algoritmo è necessario:
  - Condurre un'attenta analisi del problema ed eventualmente suddividere il problema in sottoproblemi più piccoli.
  - Individuare i possibili ingressi e precisare le uscite (definizione dei dati).
  - Definire completamente e dettagliatamente la sequenza dei passi che portano alla soluzione.



## Il crivello di Eratostene

- Si vogliono trovare tutti i numeri primi compresi fra 2 e *n* (in modo efficiente).
  1. Si costruisca una sequenza ordinata dei numeri fra 2 e *n*.
  2. Si estragga il primo numero dalla sequenza. E' necessariamente un numero primo.
  3. Si eliminino dalla sequenza tutti i multipli del numero estratto al passo 2).
  4. Se la sequenza non è vuota si torna la passo 2) altrimenti si termina.



## Il crivello di Eratostene

Esempio
Sequenza iniziale (da 2 a 20):
2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,19,20
- 2 è primo; lo elimino con tutti i suoi multipli:

3,5,7,9,11,13,15,17,19
- 3 è primo; lo elimino con tutti i suoi multipli:

5,7,11,13,17,19
- 5 è primo; lo elimino con tutti i suoi multipli:

…
- 19 è primo, la sequenza e' vuota, **termino**.



## Il crivello di Eratostene

- Algoritmo
  - Finito
  - Non ambiguo
  - Realizzabile
  - Efficiente (si spera)

## Algoritmo di Euclide

- Calcola il massimo comune divisore fra due interi positivi *m* ed *n* (supponiamo che *n* = *m*, altrimenti li possiamo scambiare).
- L'algoritmo si compone di 3 passi ripetuti ciclicamente, fino ad una condizione di arresto



## Algoritmo di Euclide

- Passo E1 {ricerca del resto} : Sia *m* = *n*. Si divide m per n. Sia r il resto. Allora 0 = *r* < *n*.
- Passo E2 {*r* = 0?} : Se *r* = 0 l'algoritmo è terminato e *n* è il risultato.
- Passo E3 {scambio} : Se *r* ? 0 si operano gli scambi *m* ? *n* e *n* ? *r*. Si torna al passo E1.



## Algoritmo di Euclide: esempio

Si trovi l' MCD fra 30 e 45 (i due numeri vanno scambiati)

| m | n | r |
|---|---|---|
| 45 | 30 | 15 |
| 30 | 15 | 0 |

Poiché il resto r =0, l'ultimo valore di n è il MCD



## Algoritmo di Euclide: finitezza

- Poiché al passo E1 {ricerca del resto} si ha sempre 0 = *r* < *n*, ad ogni scambio il valore di n decresce.
- Dopo un numero finito di passi si raggiungerà la condizione *r* = 0 e la terminazione dell' algoritmo.



## Algoritmo di Euclide: correttezza

- Ad ogni passo abbiamo che *m* = *q n* + *r* (con *q* intero).
- Se *r* ? 0 ogni divisore di *m*,*n* divide anche *r*=*m*-*qn*. D'altra parte, anche ogni divisore di n,r divide *m* = *qn* + *r*.
- In pratica le coppie m,n e n,r prima e dopo lo scambio *m* ? *n* e *n* ? *r hanno lo stesso MCD.*
- Se *r* = 0 l'algoritmo termina e *n* è il MCD.



## Algoritmo di Euclide

- L'algoritmo descritto non è l'unico per risolvere il problema del MCD
- Tuttavia è il più efficiente se a priori non conosco nulla dei due numeri

## Calcolo MCD: algoritmo 2

- Problema: dati due numeri determinare il loro massimo comune divisore:
  - scomporre in fattori primi i due numeri
  - considerare solo fattori comuni
  - per ogni fattore comune considerare l'esponente più piccolo
  - moltiplicarli tra loro



## Calcolo MCD: algoritmo 3

- Problema: dati due numeri determinare il loro massimo comune divisore
  - costruire insieme divisori primo numero
  - costruire insieme divisori secondo numero
  - costruire intersezione fra i due insiemi
  - individuare nell'intersezione l'elemento più grande



## Algoritmo

- Per realizzare un algoritmo con un calcolatore occorre rappresentare la sequenza di passi che portano alla soluzione con istruzioni appartenenti ad un linguaggio di programmazione.



## Diagrammi di flusso (Flow-Chart)

- I diagrammi di flusso sono un formalismo grafico per descrivere gli algoritmi.
- I diagrammi di flusso scompongono in passi successivi gli algoritmi.
- Un diagramma di flusso è una descrizione più efficace e meno ambigua di una descrizione a parole.



## Diagrammi di flusso

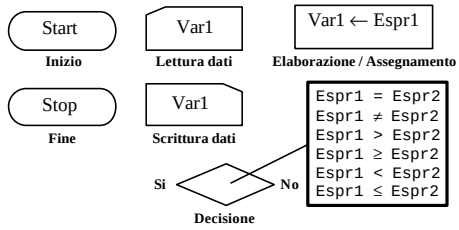
- Operazioni rappresentabili con un diagramma di flusso
  - Ingresso/Uscita dati (rappresentate come **schede**)
  - Operazioni sui dati (rappresentate come **rettangoli**)
    - Trasferimento di informazione (Assegnamenti)
    - Calcolo di espressioni aritmetiche e logiche
  - Assunzione di decisioni (rappresentate come **rombi**)
  - Esecuzione di iterazioni, o cicli (combinazioni di rettangoli e rombi)
- Possono contenere costanti e variabili



## Diagrammi di flusso

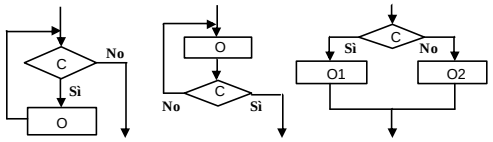
- Un diagramma di flusso è costituito da due tipi di entità:
  - Nodi
    - rappresentano le operazioni e gli stati di inizio e fine dell'algoritmo
  - Archi orientati
    - rappresentano con frecce il 'flusso' dei dati, quindi la sequenza delle operazioni: il risultato prodotto da un nodo è successivamente elaborato dal nodo a cui punta l 'arco uscente dal primo nodo
- Una struttura di questo tipo è detta *grafo (orientato)*

# Tipi di Nodi






# Strutture di Controllo




While - Do

Ripete una stessa operazione O finché la condizione C resta vera

Repeat - Until

Ripete una stessa operazione O finché la condizione C non diventa vera

If - Then - Else

Se C è vera esegue O1, altrimenti esegue O2

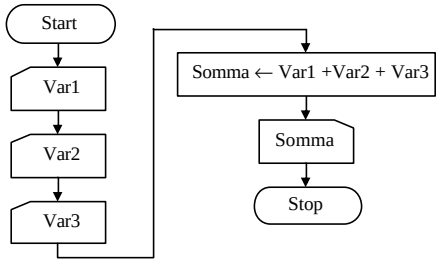


# Programmazione Strutturata

- Si compone di sequenze di azioni, decisioni (if then, if then else) e cicli (while-do, repeat until).
- Ogni diagramma ha esattamente un ingresso ed una uscita.
- Ogni azione può essere
  - una operazione semplice
  - una azione composta da altri diagrammi strutturati

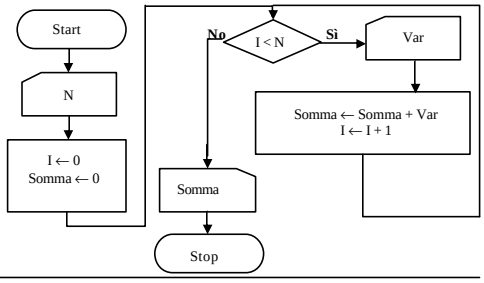


# Esempio: Somma di Tre Numeri






# Esempio: Somma di N Numeri






# Programmazione Top-Down

- La programmazione top-down è una tecnica di programmazione per la realizzazione di programmi con la scomposizione iterativa di un problema in sotto-problemi.

## Tecniche di programmazione

- Programmazione *top-down*:
  - scomposizione iterativa del problema in sottoproblemi
  - i sottoproblemi devono essere indipendenti ed avere interfacce ben definite
  - visibilità dei dettagli di ogni sottoproblema solo all'interno del sottoproblema stesso

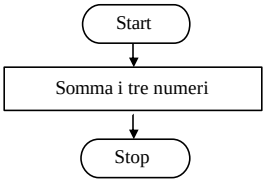


## Programmazione Top-Down

- La programmazione top-down si presta molto bene per la definizione di algoritmi con i diagrammi di flusso:
  - All'inizio il diagramma di flusso è rappresentato da un nodo che rappresenta la soluzione al problema.
  - Questo nodo viene scomposto in una rete di nodi in modo iterativo.
  - La scomposizione termina quando i singoli nodi possono essere rappresentati da semplici sequenze di istruzioni del linguaggio di programmazione scelto.

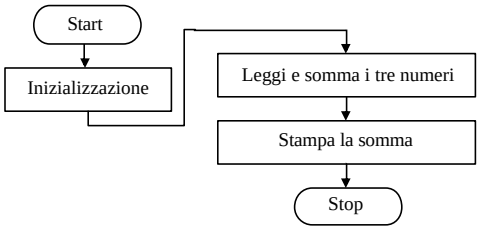


## Esempio: Somma di Tre Numeri






## Esempio: Somma di Tre Numeri






## Esempio: Somma di N Numeri