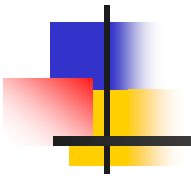
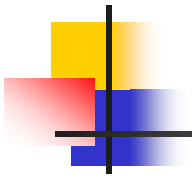


# Programmazione Strutturata

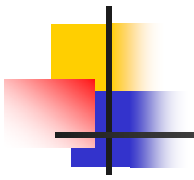


# Programmazione strutturata

- La programmazione strutturata nasce come proposta per regolamentare e standardizzare le metodologie di programmazione (Dijkstra, 1965)

- **Obiettivo:**
  - rendere più facile la lettura dei programmi
    (e quindi la loro modifica e manutenzione)

- **Idea di base:**
  - La parte esecutiva di un programma viene vista come un comando (complesso o *strutturato*) ottenuto componendo **istruzioni elementari**, mediante alcune regole di composizione (**strutture di controllo**)



# Programmazione strutturata

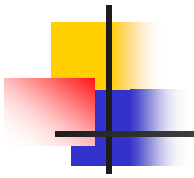
**Concetti chiave:**

Utilizzo solamente delle tre *strutture di controllo* seguenti:

- concatenazione o composizione **BLOCCO**
- struttura condizionale **SELEZIONE**
- struttura di ripetizione o iterazione **CICLO**

Abolizione di **salti incondizionati** (***goto***) nel flusso di controllo

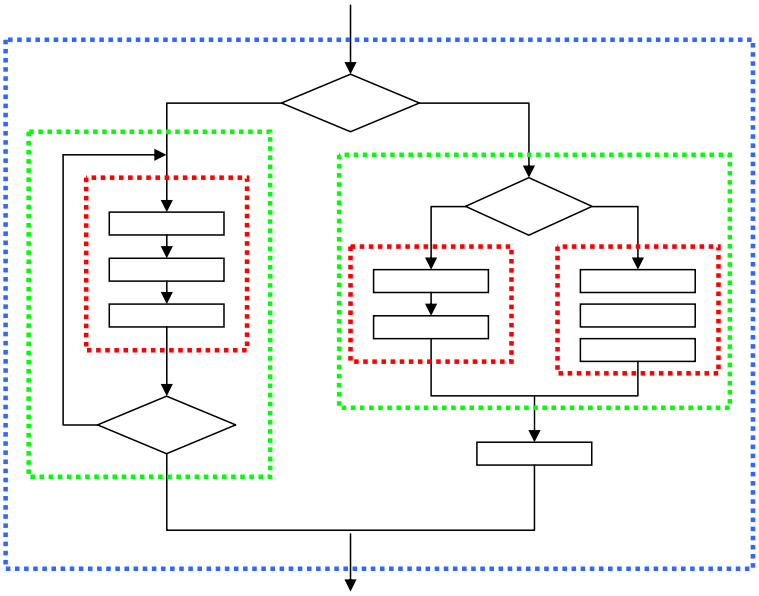




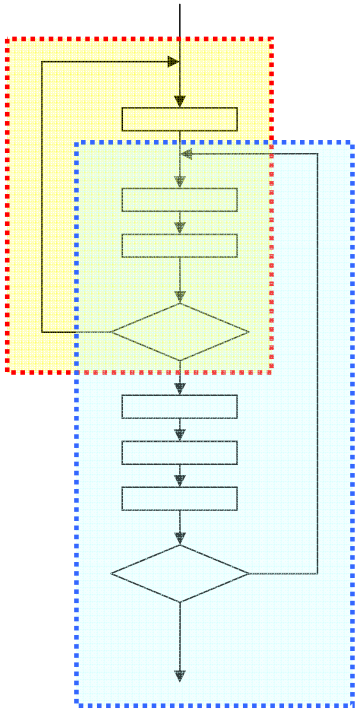
# Schemi a blocchi strutturati

- Le regole della programmazione strutturata impongono quindi delle severe restrizioni nella costruzione dei diagrammi di flusso
  - Ci si basa su poche strutture di base con **un solo ingresso** e **una sola uscita**
  - Le tre strutture (come vedremo) possono essere **concatenate** una di seguito all'altra o **nidificate** una dentro l'altra
  - Non possono però essere ***intrecciate*** o ***accavallate***

# Schemi a blocchi strutturati



**Corretto**



**Sbagliato:**
È un tipico esempio di
“spaghetti programming”



# Un dubbio...

- Ma queste restrizioni, cioè l’utilizzo di solamente sequenza, selezione e iterazione senza l’uso dei salti, dà la stessa “potenza di calcolo” (espressività) oppure si perde qualcosa?

**Teorema di Bohm- Jacopini (1966)**

*(in versione semplificata)*

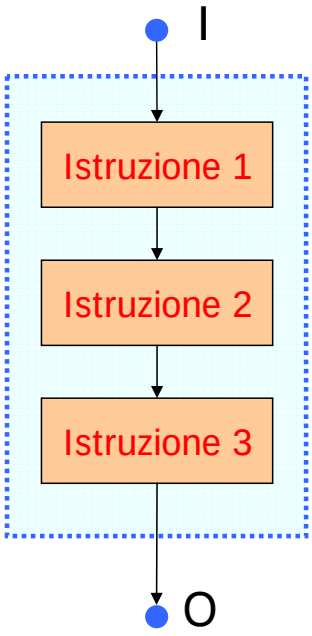
Le strutture di sequenza, selezione e iterazione sono sufficienti ad esprimere un qualsiasi algoritmo

Ossia:

L’uso di queste sole strutture non limita il potere espressivo

# Struttura di composizione (Blocco)

Diagramma a blocchi




Pseudo-codice

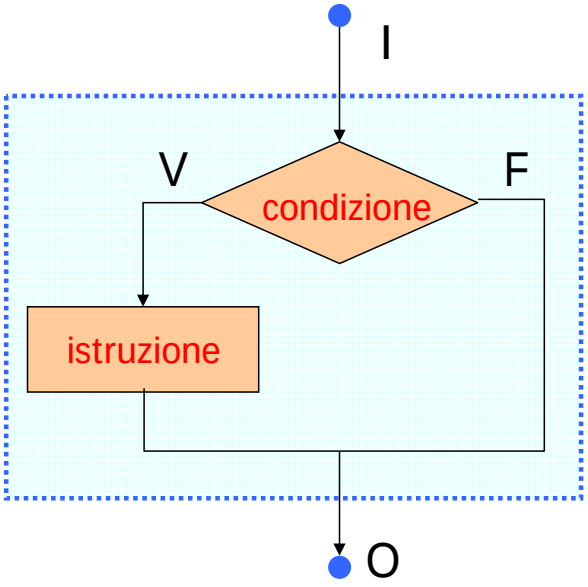
```
{
    Istruzione 1
    Istruzione 2
    Istruzione 3
}
```

Le tre istruzioni vengono eseguite sequenzialmente.
Il concetto di blocco permette inoltre di considerare la sequenza di più istruzioni come un'unica istruzione (composta)



# Struttura condizionale *(a una via)*

Diagramma a blocchi



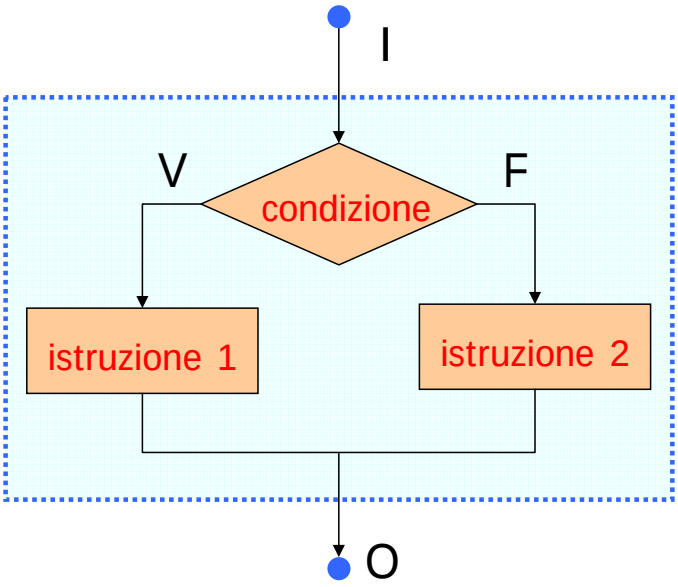

Pseudo-codice

```
if condizione
    istruzione
```

Se la **condizione** è vera allora viene eseguita **istruzione** altrimenti no

# Struttura condizionale *(a due vie)*

Diagramma a blocchi




Pseudo-codice

```
if condizione
      istruzione 1
else
      istruzione 2
```

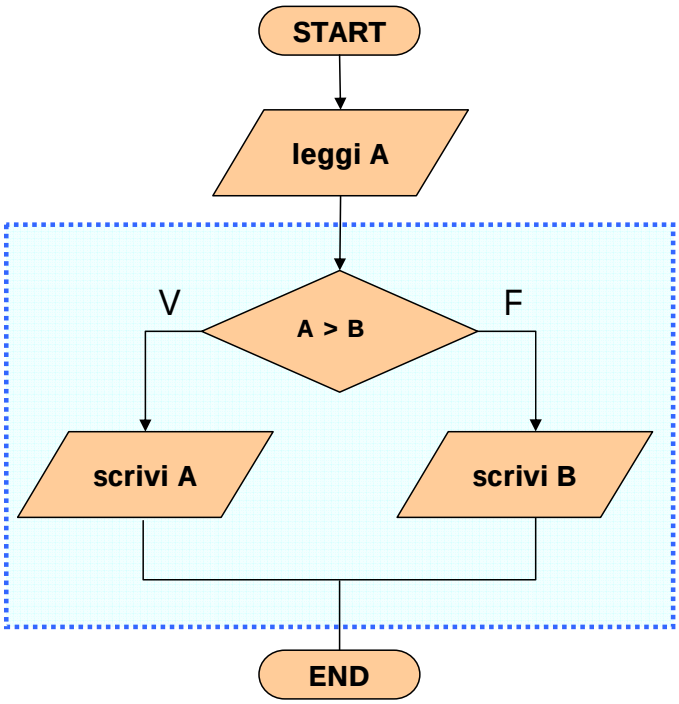
Se la **condizione** è vera allora viene eseguita **istruzione 1** altrimenti
viene eseguita **istruzione 2**



# Strutture condizionali *(osservazioni ed esempi)*

**Es: determina e stampa il maggiore tra due numeri letti da input**

```
leggi A,B;
if (A > B)
      scrivi A;
else
      scrivi B;
```



# Strutture condizionali *(osservazioni ed esempi)*

- <istruzione1> e <istruzione2> sono ciascuna una *singola istruzione*
- Qualora in un ramo occorra specificare più istruzioni, si deve quindi utilizzare un *blocco*

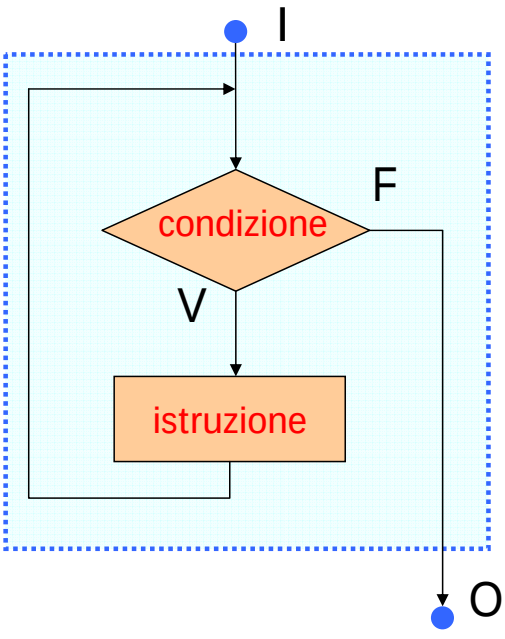
**Es: scrivi in ordine crescente due numeri letti da input**

```
leggi A, B;
if (A < B){
        scrivi A;
        scrivi B;
}
else {
        scrivi B;
        scrivi A;
}
```



# Struttura di iterazione *(while)*

Diagramma a blocchi



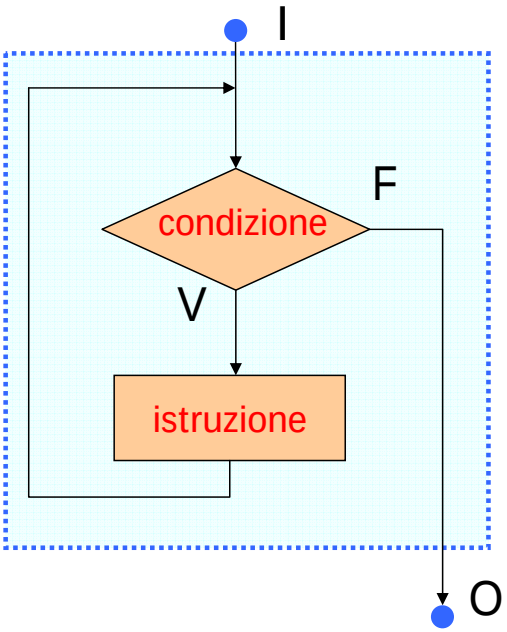

Pseudo-codice

**while** condizione
istruzione

Fintantoché la **condizione** si mantiene vera allora esegui **istruzione**

# Struttura di iterazione *(while)*

**while** condizione
istruzione




- <istruzione> viene ripetuta *per tutto il tempo in cui la condizione rimane* ***vera***
- Se la condizione è già inizialmente falsa, l'iterazione non viene eseguita *neppure una volta*
- In generale, *non è noto* a priori ***quante volte*** l'istruzione sarà ripetuta



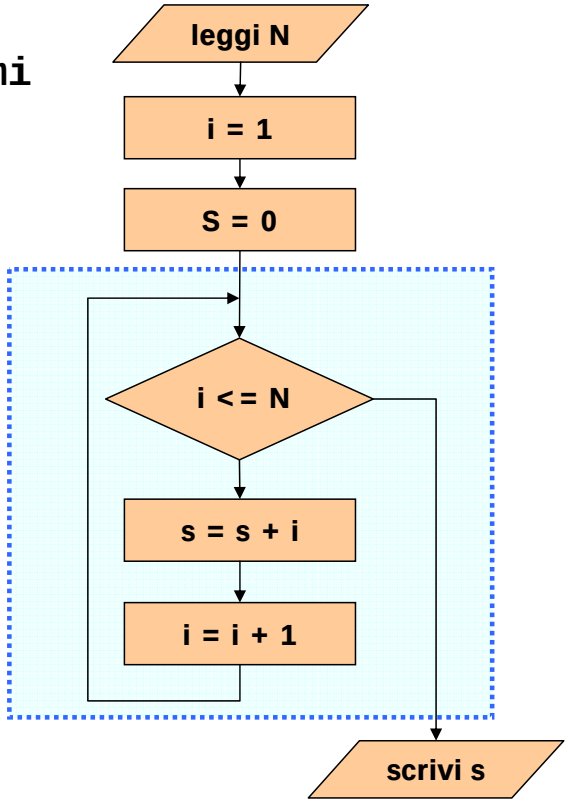
# Struttura di iterazione *(while)*

- Prima o poi, *direttamente o indirettamente*, l'istruzione deve *modificare la condizione:* altrimenti → ***CICLO INFINITO***
- Per questo motivo, quasi sempre *<istruzione>* è in realtà un blocco, che contiene una istruzione in cui si modifica qualche variabile che compare nella condizione

# Struttura di iterazione *(esempi)*

**Es: Letto un numero intero N da input, sommare i primi N numeri positivi e scrivere il risultato**

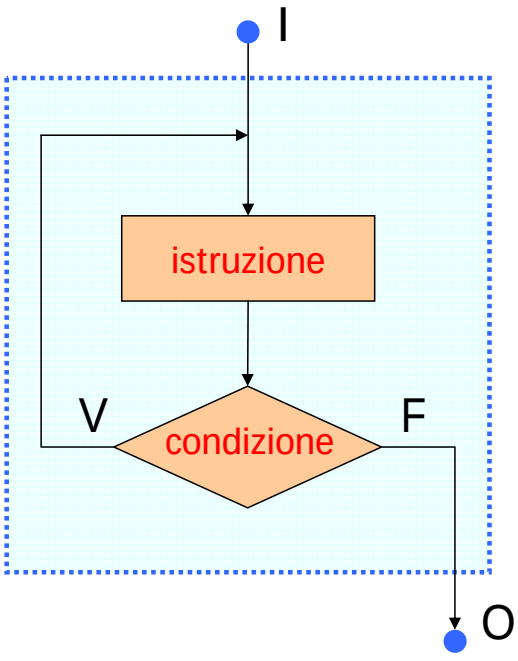
```
leggi N;
i=1;
s=0;
while (i<=N) {
   s = s + i;
   i = i + 1;
}
scrivi s;
```






# Struttura di iterazione *(do-while)*

Diagramma a blocchi



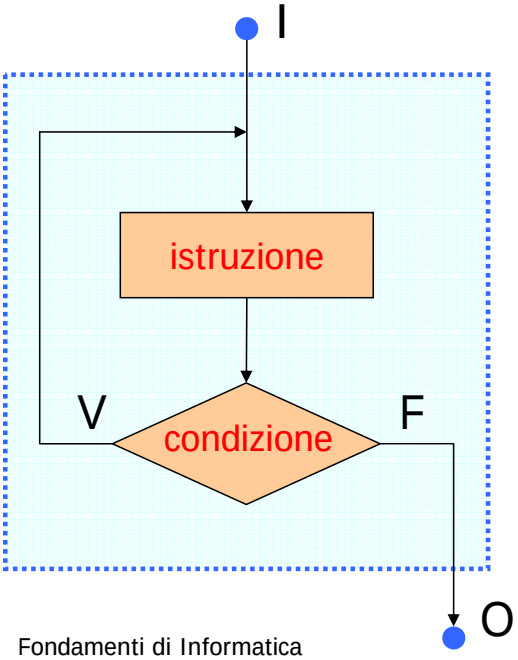

Pseudo-codice

**do**
istruzione
**while** condizione

Esegui **istruzione** fintantoché la **condizione** si mantiene vera

# Struttura di iterazione *(do-while)*

**do**
    istruzione
**while** condizione




- È una variante della precedente: la condizione viene verificata ***dopo*** aver eseguito <istruzione>
- Se la condizione è falsa, l'istruzione **viene comunque eseguita *almeno una volta***



# Struttura di iterazione *(do-while)*

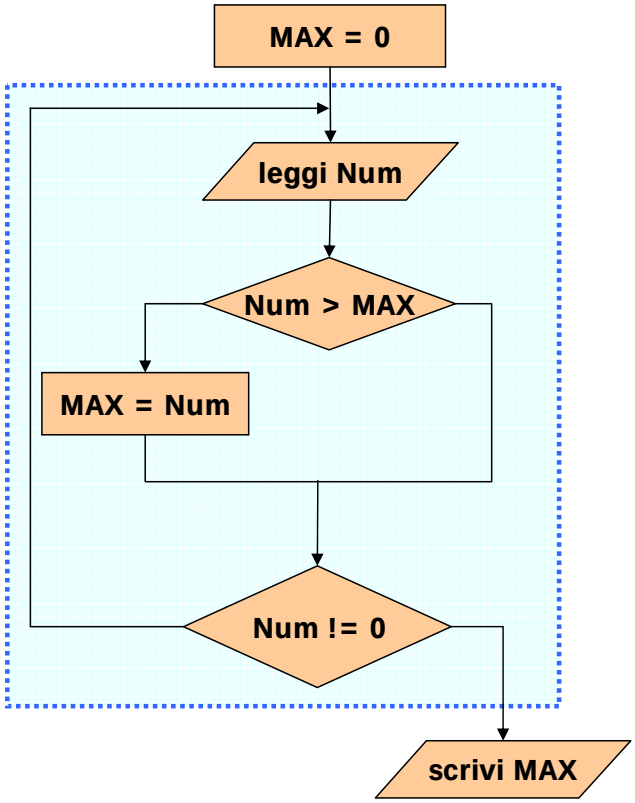
- Analogamente al *while*, per evitare il ciclo infinito, <istruzione> deve modificare prima o poi qualche variabile che compare nella condizione
- Si noti che, come nel caso del *while*, si esce dal ciclo quando la condizione è falsa
- **È adatta** a quei casi in cui, per valutare condizione, è necessario aver già eseguito <istruzione>

  (esempio tipico: controllo di valori di input)
- **Non è adatta** a quei casi in cui il corpo del ciclo può non dover essere *mai eseguito*

# Struttura di iterazione *(esempi)*

**Es:Scrivere il valore massimo di una sequenza di interi positivi (letti da input) chiusa da uno zero**

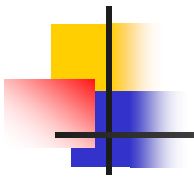
```
MAX = 0;
do {
   leggi Num;
   if (Num > MAX)
         MAX = Num;
} while (Num != 0);
scrivi MAX;
```






# Alcune considerazioni finali

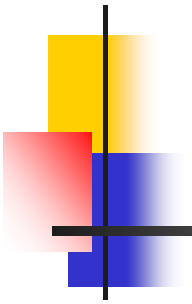
- Tutti i linguaggi imperativi implementano una qualche forma delle strutture presentate
- Naturalmente la forma sintattica può a volte variare leggermente, ma il funzionamento rimane identico
  - uso di diversi marcatori per l'inizio e la fine di un blocco (*begin* e *end* in Pascal)
  - uso della parola chiave *then* nell'istruzione if (in Pascal)
- Inoltre i vari linguaggi introducono altre strutture di controllo, non indispensabili, ma atte a semplificare il lavoro di scrittura del codice
  - istruzione di scelta multipla (*switch-case* in C)
  - istruzione iterativa controllata da contatore (*for*)



# Programmazione strutturata (Conclusioni e Vantaggi)

Vantaggi:

- Leggibilità
- Supporto a metodologie di progetto top-down:
  - Soluzione di problemi complessi attraverso la scomposizione in sotto-problemi, a loro volta scomponibili in sotto-problemi, etc.
  - La soluzione si ottiene componendo le soluzioni dei sottoproblemi attraverso strutture di concatenazione, di selezione e di ripetizione
- Supporto a metodologia bottom-up:
  - La soluzione di problemi avviene aggregando componenti già disponibili mediante concatenazione, selezione e ripetizione
- Maggior facilità di verifica e manutenzione



# Bibliografia

- Documentazione Ing. O.Tomarchio
- Introduzione all'informatica in C, Demichelis, Piccolo, McGraw-Hill.
- C: Corso completo di programmazione 2ed, Apogeo 2000.