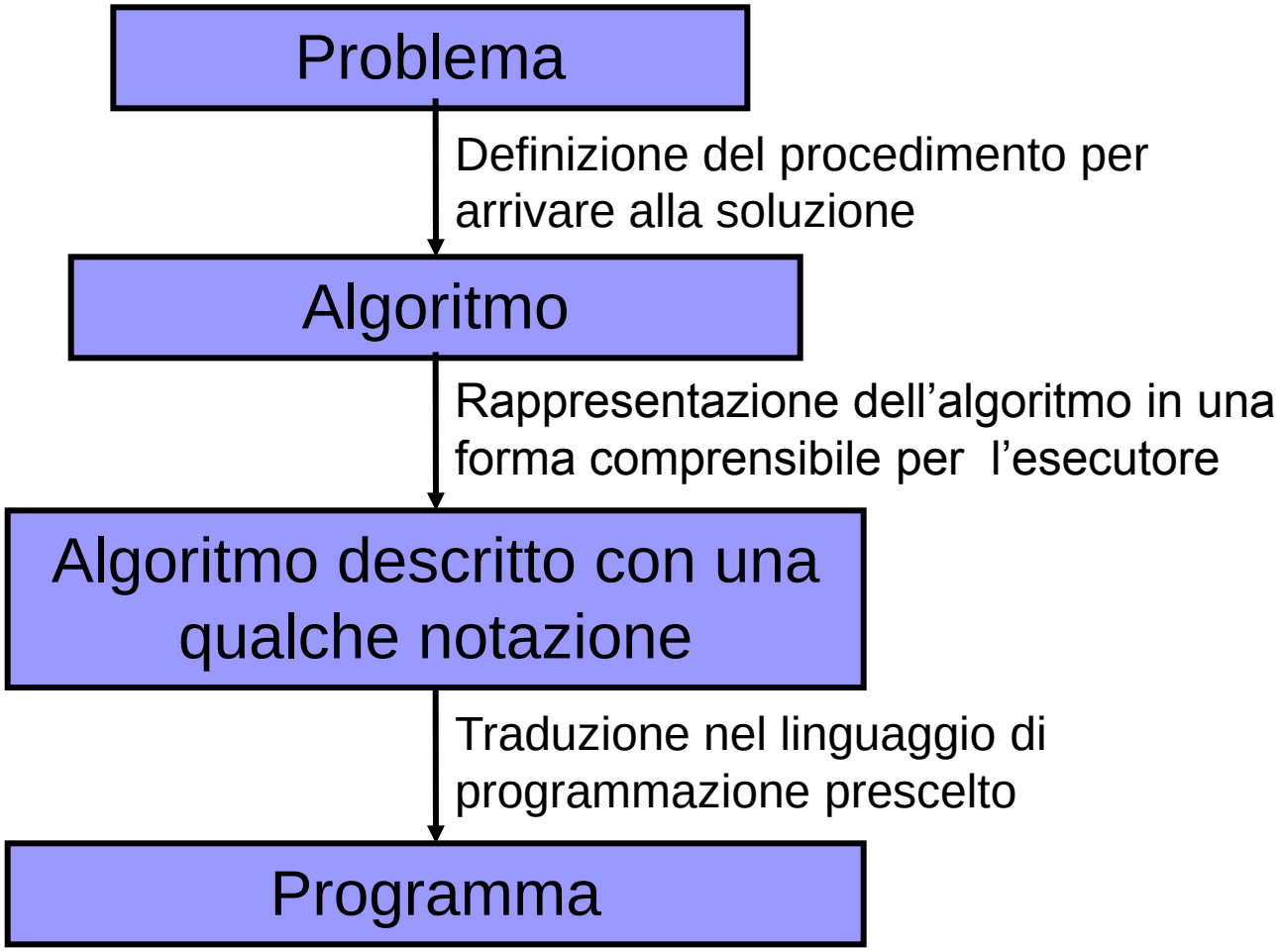
Informatica

Terzo anno

Prof. A. Longheu

Rappresentazione degli algoritmi



Linguaggio naturale – notazione lineare

- Nella rappresentazione di un algoritmo si possono riconoscere tre classi di istruzioni:
- istruzioni di ingresso e uscita (dati e risultati del problema)
- istruzioni di assegnamento (P = A con il significato assegna il nome logico P al valore A)
- istruzioni di controllo: sono quelle istruzioni che modificano la sequenza dell'esecuzione
  - alterazione incondizionata
  - alterazione condizionata

Esempio

- Calcolare il prodotto di due numeri interi positivi A e B supponendo che l'esecutore conosca solo le operazioni di somma, sottrazione e confronto fra numeri.

#	Istruzione
1	leggi A e B
2	P = A
3	se B è uguale 1 andare all'istruzione 7
4	P = P+ A
5	B = B - 1
6	andare alla 3
7	scrivi P
8	fine

Tabella di Traccia

- L'algoritmo può essere testato attraverso l'utilizzo di una tabella di Traccia.
- Esempio per l'algoritmo precedente eseguito per i valori di ingresso 7 e 3

Istruzione	A	B	P

N.istruzione	Istruzione
1	leggi A e B
2	P = A
3	se B è uguale 1 vai a 7
4	P = P+ A
5	B = B - 1
6	andare alla 3
7	scrivi P
8	fine

N.istruzione	Istruzione
1	leggi A e B
2	P = A
3	se B è uguale 1 andare all'istruzione 7
4	P = P+ A
5	B = B - 1
6	andare alla 3
7	scrivi P
8	fine

- L'algoritmo può essere testato attraverso l'utilizzo di una tabella di Traccia.
- Esempio per l'algoritmo precedente eseguito per i valori di ingresso 7 e 3

Istruzione	A	B	P
Inizio			
Leggi A, B	7	3	
P = A			7
Se B è uguale 1			
P = P +A			14
B = B -1		2	
Se B è uguale 1			
P = P +A			21
B = B -1		1	
Se B è uguale 1			
Scrivi P			

Rappresentazione degli algoritmi

Rappresentazione grafica

Diagrammi a blocchi / Flow Chart

Rappresentazione testuale

Notazione Lineare Strutturata / PseudoCodice

Algoritmi: operazioni base

- Le operazioni primarie sono:
  - Trasferimento di informazioni (istruzioni di I/O)
    lettura dati, scrittura risultati, visualizzazione dati intermedi
  - Esecuzione di calcoli (valutazione espressioni)
  - Istruzioni di assegnamento
  - Strutture di controllo (che modificano il flusso sequenziale di esecuzione delle operazioni)

Simboli convenzionali

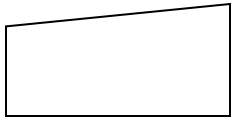
ingresso/uscita

Ciclo

documento

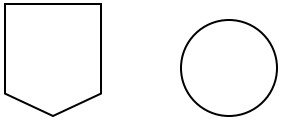
Elaborazione

inizio/fine

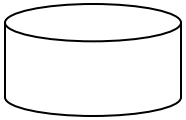


input manuale

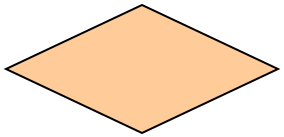
elab. predefinita



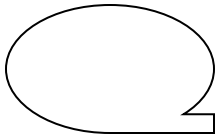
connessioni



disco



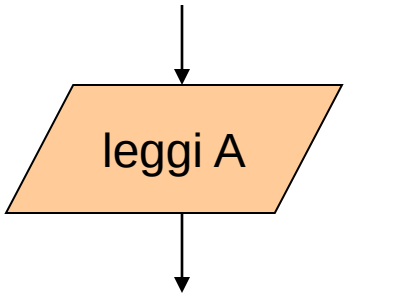
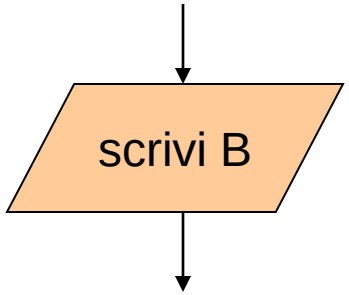
decisione



mem. sequenziale

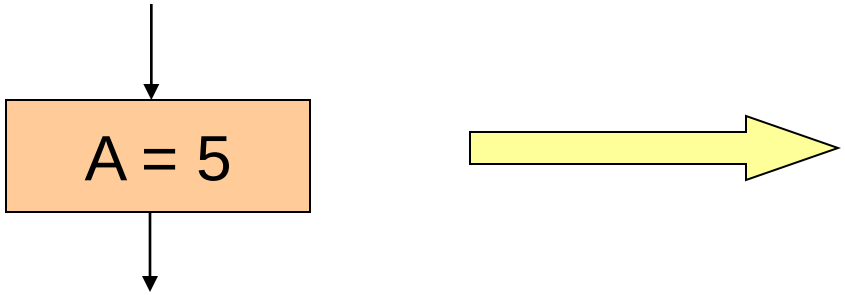
Istruzioni di I/O

- lettura di dati in input
- scrittura dei risultati in output

Diagramma a blocchi	Pseudo-codice
	leggi A
	scrivi B

Istruzione di assegnamento

- Concetto di variabile
  - Identificata da un'etichetta / identificatore simbolico
  - Può essere comodo pensare alla variabile come ad un contenitore in cui possiamo memorizzare o reperire dei dati utilizzati durante il calcolo
  - L'istruzione di assegnamento permette di assegnare un valore ad una variabile

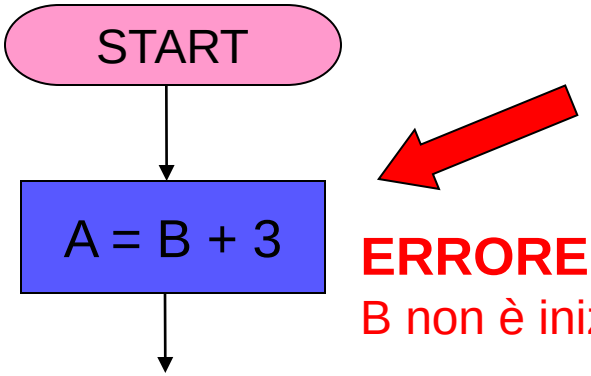


*Alla variabile **A** viene assegnato il valore **5***

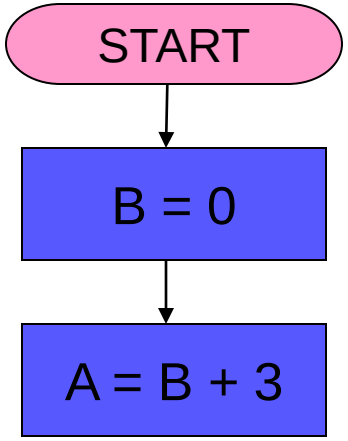
Istruzione di assegnamento

Inizializzazione

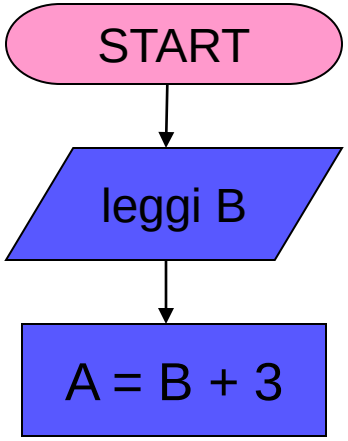
- All'inizio di un algoritmo una variabile non ha alcun valore
- Non ha quindi senso utilizzarla in una espressione (**è un errore!**)
- Essa deve essere inizializzata:
  - O esplicitamente mediante un assegnamento
  - Oppure mediante una operazione di lettura



ERRORE:
B non è inizializzata



CORRETTO



CORRETTO

Istruzione di assegnamento

- E' stato usato il simbolo = per indicare l'istruzione di assegnamento
- Alcuni testi/autori indicano invece il simbolo ← (*per evitare confusione con l'operatore di uguaglianza*)

Dato il seguente frammento di codice:

A = 5

A = A + 1 ⟸ Cosa fa quell'istruzione?

L'esecutore esegue i seguenti passi:
- prima valuta l'espressione a destra, cioè calcola il valore di A+1 che vale 6
- dopo assegna tale valore alla variabile a sinistra, cioè ad A
- alla fine dell'esecuzione di quella istruzione quindi, A vale 6

Operare con le variabili

- a=b, che si puo' immaginare come a ←b, si legge "prendi il contenuto della variabile b e memorizzalo in a (cancellando l'eventuale valore precedente di a)
- a=b quindi NON EQUIVALE a b=a, a differenza della matematica; l'istruzione si legge sempre **da destra verso sinistra**
- Esempio **scambio di variabili**
- La sequenza a=b, b=a NON OPERA LO SCAMBIO, infatti se a=10 e b=7, l'istruzione a=b prende 7 e sovrascrive il 10, avendo quindi 7 sia in a che in b; la successiva b=a riprende il 7 di a e lo memorizza in b. Lo scambio allora richiede una terza variabile: c=a, a=b, b=c

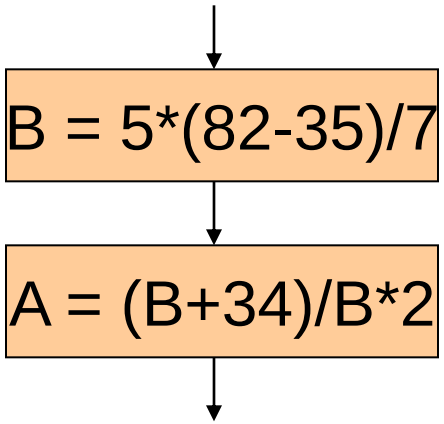
Operare con le variabili

- Esempio di **incremento e decremento**
  - a=a+1, che in matematica sarebbe sbagliato (semplificando a da ambo i membri verrebbe 0=1), si legge, partendo da destra (quindi da "a+1") "prendi il valore attuale di a, gli sommi 1, e il risultato lo rimetti dentro a (cancellando il valore attuale)
  - a=a+1 quindi si può leggere come $a_{futuro} \leftarrow a_{presente} + 1$
  - a=a+3 ad esempio incrementa il valore di a di 3
  - a=a+7 incrementa a di 7
- In generale, esistono un insieme di **operatori** per lavorare con le variabili (+,-,*,/….)

Valutazione espressioni

- L'esecutore è in grado di valutare espressioni aritmetiche

Diagramma a blocchi



Pseudo-codice

```
B = 5*(82-35)/7
A = (B+34)/B*2
```

Rappresentazione degli algoritmi

I diagrammi di flusso possono presentarsi:

- Ad un **livello generale**
- Ad un **livello particolare**

a seconda del livello di dettaglio con cui specificano le operazioni da compiere.

E' opportuno procedere per fasi successive:

- Un diagramma globale (di massima) per focalizzare le operazioni essenziali da compiere
- Un diagramma di flusso più particolareggiato, che tenga conto di operazioni più semplici e più elementari

Rappresentazione degli algoritmi

- Esempi di DAB -> \esempi_dab

Limiti dei diagrammi a blocchi

- I DaB sono generalmente illeggibili, non si riesce a seguire l'algoritmo soprattutto quando superano le dimensioni di un semplice esercizio didattico. La lettura andrebbe fatta un po' dall'alto un po' dal basso senza un ordine preciso
- I DaB sono facilmente esposti ad errori logici, bastano pochi accavallamenti di cicli per perdere il filo del controllo. Ciò nasce dalle correzioni consentite da un indisciplinato uso delle frecce che causa un proliferare di errori logici che si accentua con la complessità del problema e l'inesperienza del risolutore
- Scarsa praticità dovuta alla natura grafica bidimensionale
- difficoltà di riconoscimento della struttura di controllo

Soluzione

- La soluzione a questi problemi consiste nell'imporre una ***disciplina di composizione*** che eviti cattive strutturazioni degli algoritmi
- L'idea base è:
  - **un algoritmo deve essere letto dall'alto al basso secondo un ordine sequenziale di esecuzione**
- Cio non significa che non possono esistere dei cicli o dei test, ma che questi siano strutturati in modo da poter essere considerati come un unico blocco operativo con un unico ingresso e una sola uscita.
- Si possono distringuere
  - blocchi semplici
  - blocchi composti

Tutti con un unico punto di ingresso e un unico punto di uscita

Soluzione: 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:

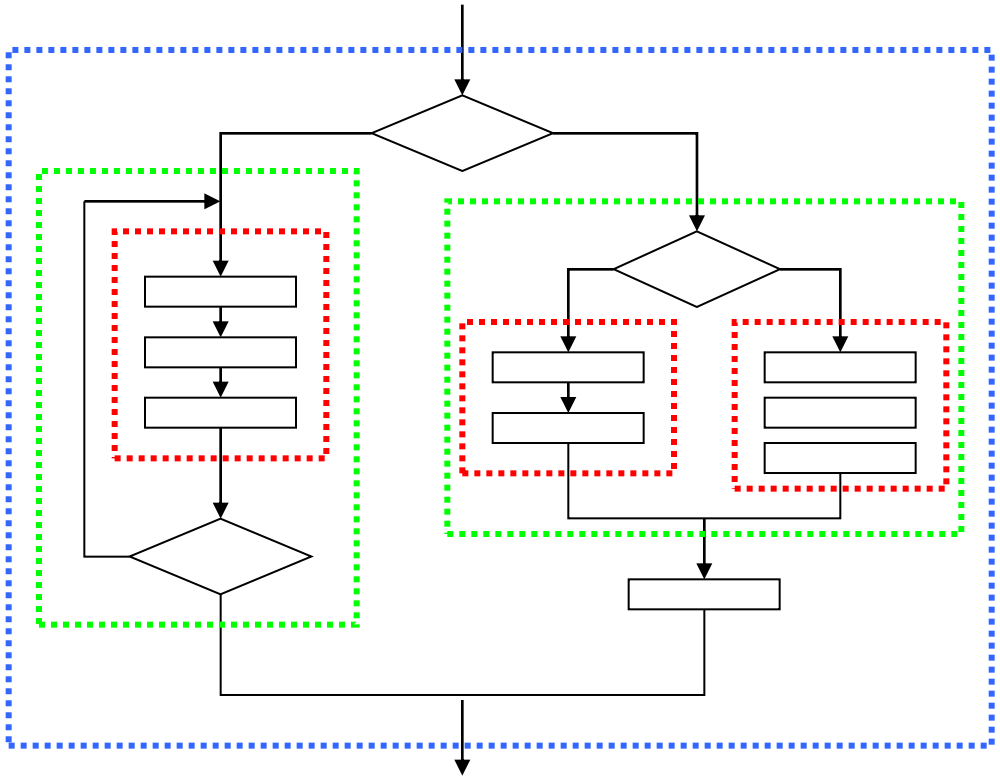
- concatenazione o composizione **BLOCCO**
- struttura condizionale **SELEZIONE**
- struttura di ripetizione o iterazione **CICLO**

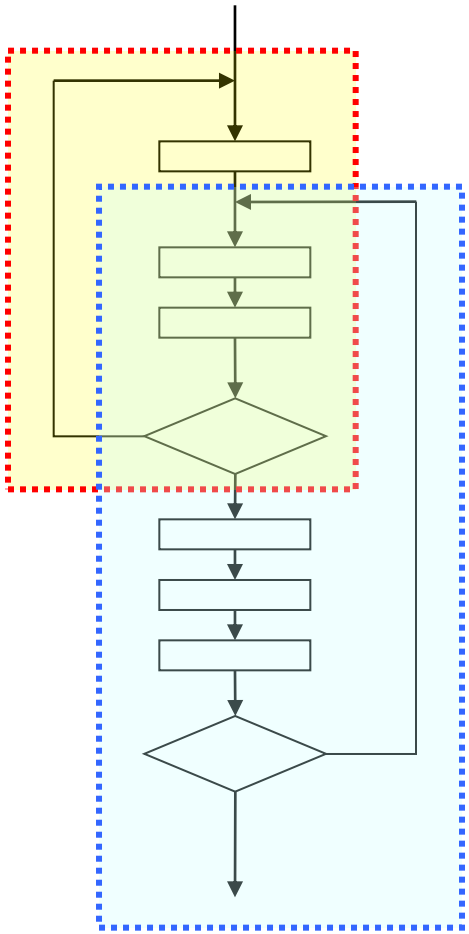
Abolizione di **salti incondizionati** (***goto***) nel flusso

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)

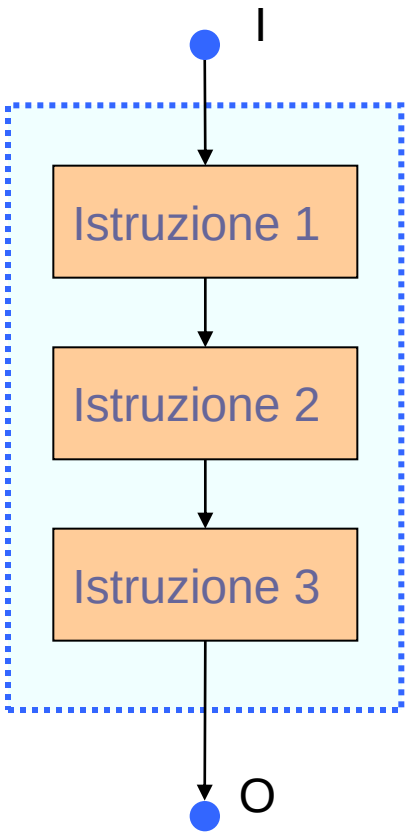
Le strutture di sequenza, selezione e iterazione sono sufficienti ad esprimere un qualsiasi algoritmo

Ossia:

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

Struttura di composizione (Blocco)

Diagramma a blocchi



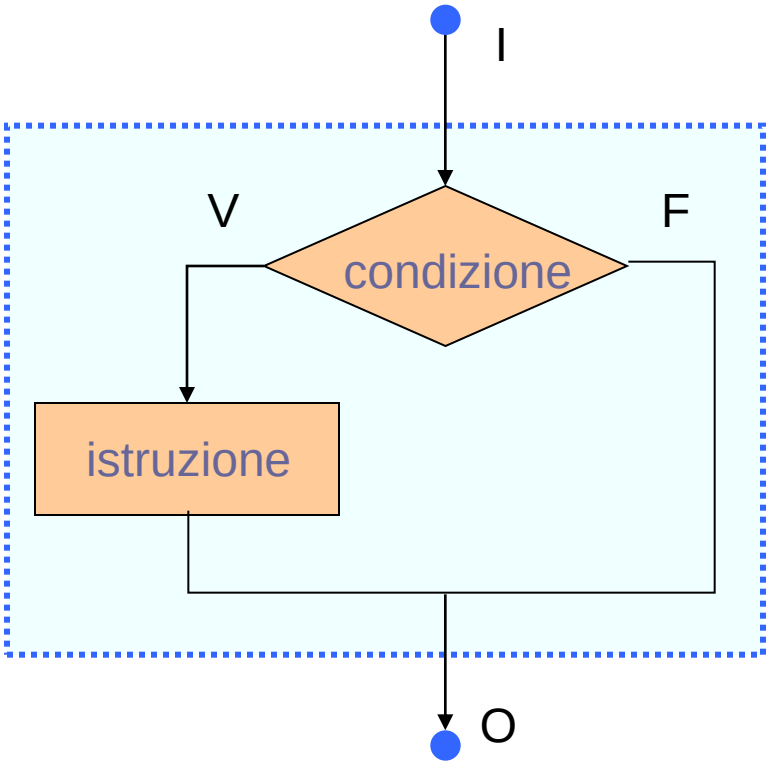
Pseudo-codice

```
{
    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



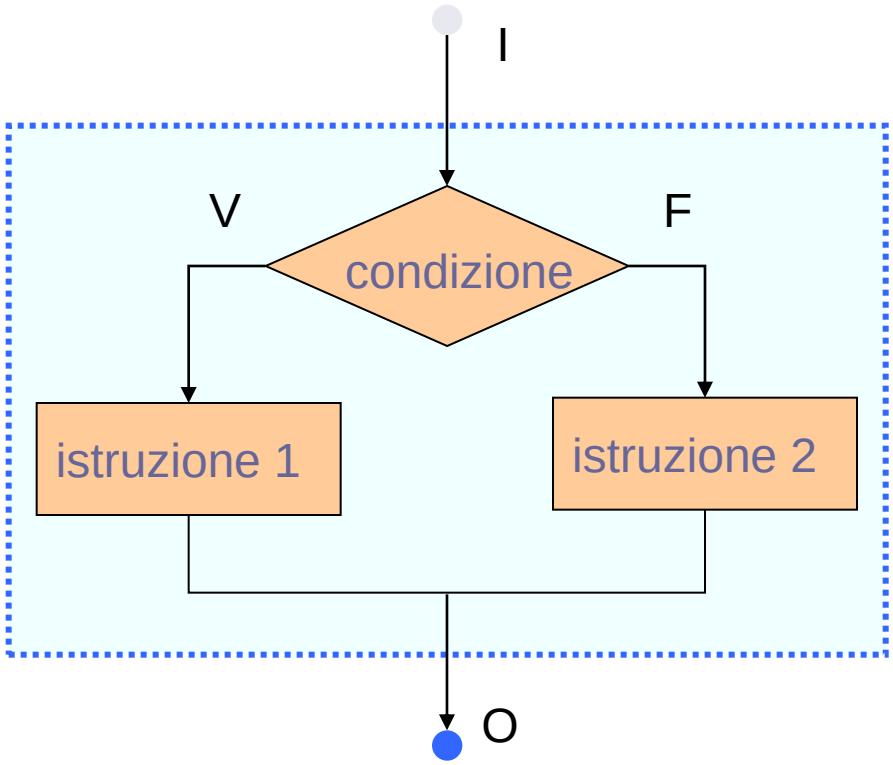
Pseudo-codice

```
if condizione
    istruzione
```

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

Struttura condizionale *(a due vie)*

Diagramma a blocchi



Pseudo-codice

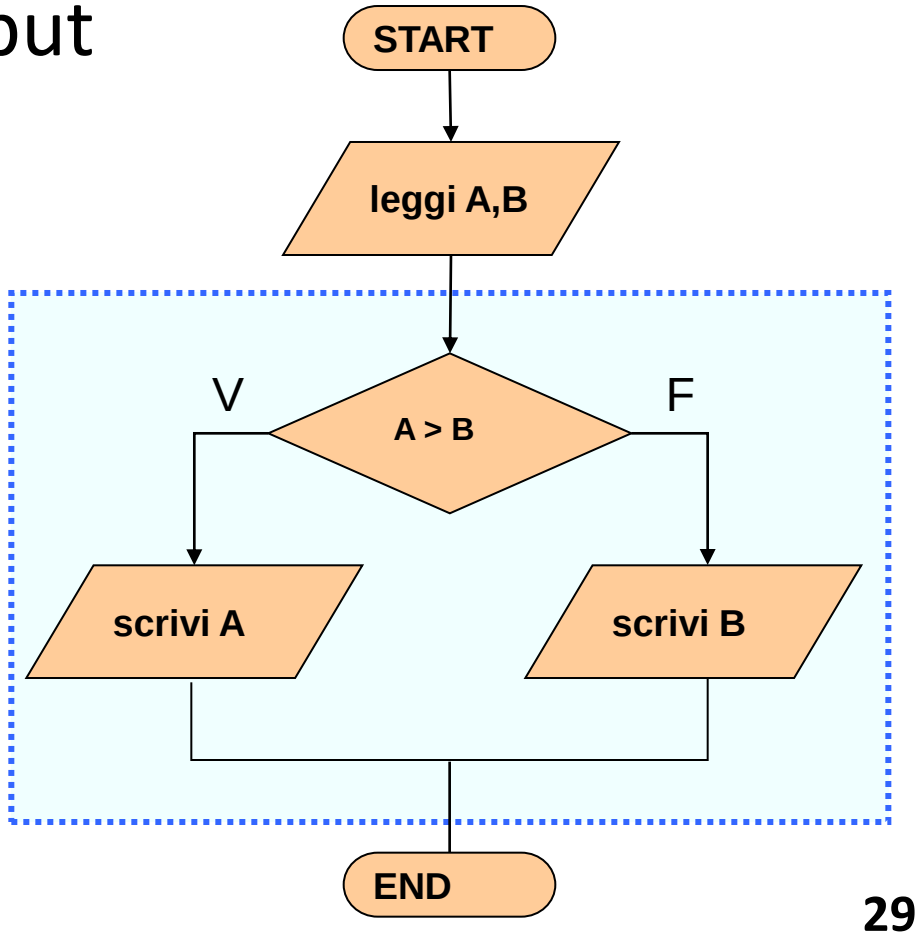
```
if condizione
    istruzione 1
else
    istruzione 2
```

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)*

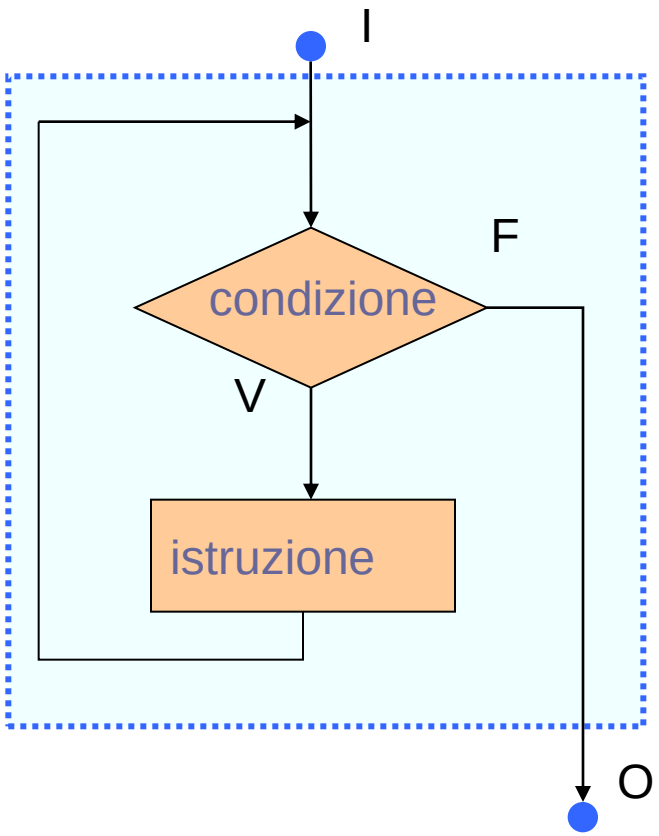
- <istruzione1> e <istruzione2> sono ciascuna una *singola istruzione*
- Qualora in un ramo occorra specificare più istruzioni, si deve quindi utilizzare un *blocco*

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



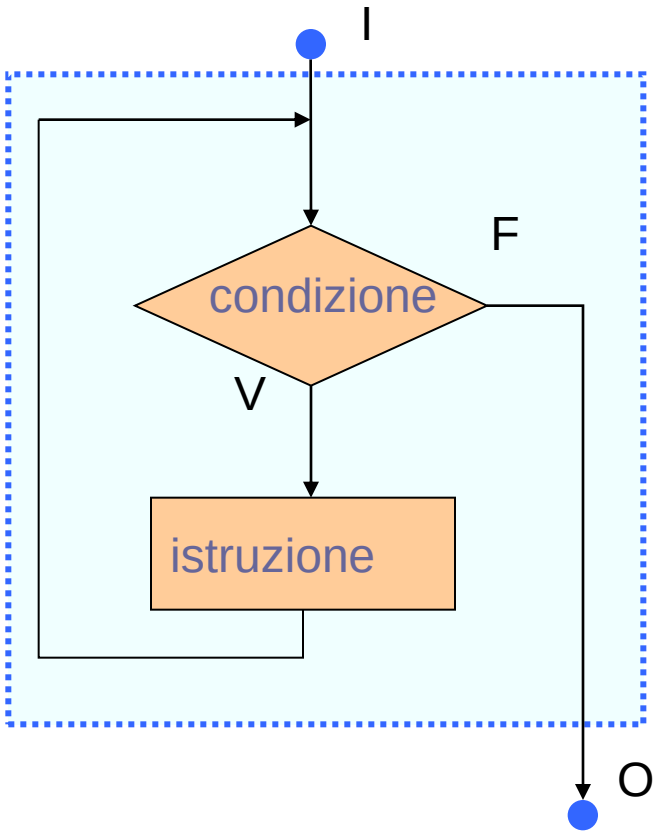
Pseudo-codice

```
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)*

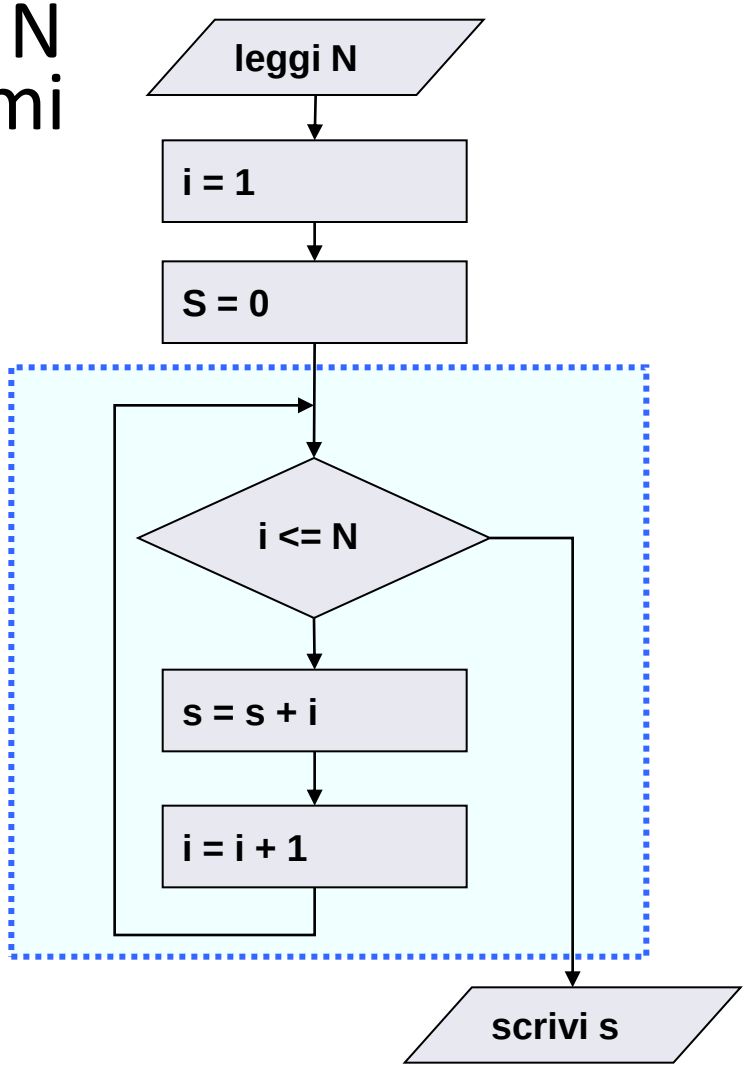
- Prima o poi, *direttamente o indirettamente*, l'istruzione deve *modificare la condizione:* altrimenti → ***CICLO INFINITO***

- 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)*

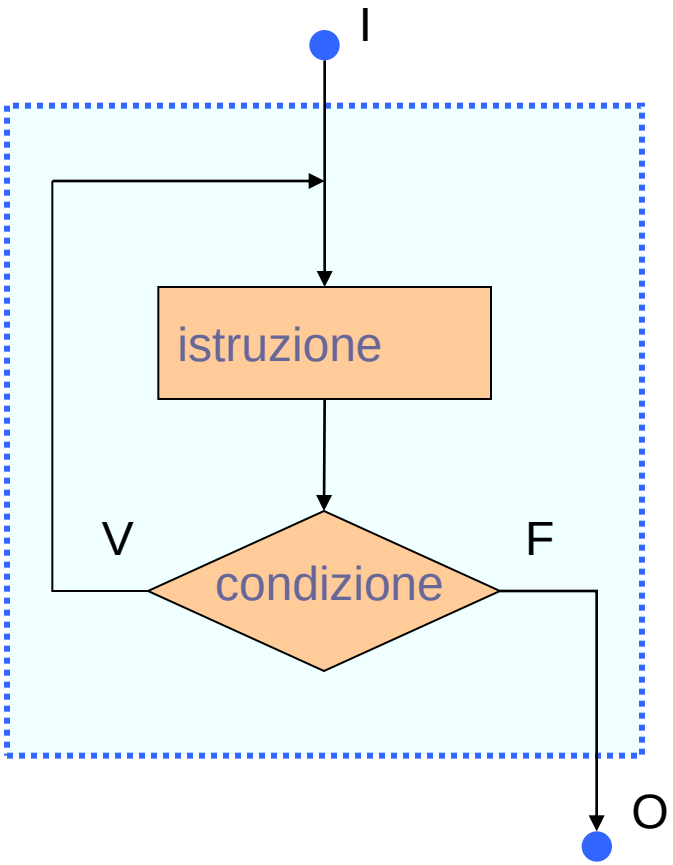
Es: Letto un numero intero N da input, sommare i primi N numeri positivi e scrivere il risultato

```
{
    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



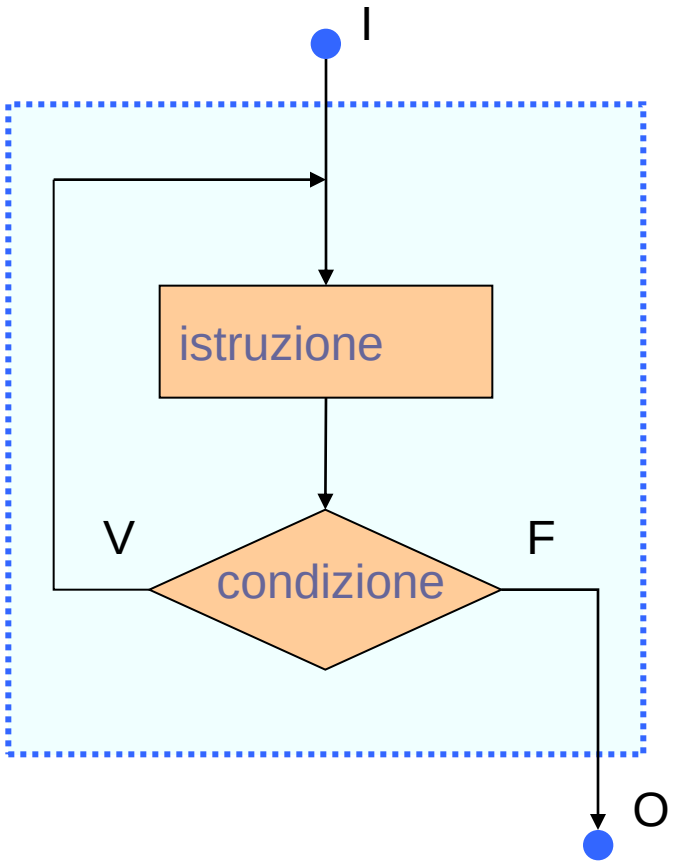
Pseudo-codice

```
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)*

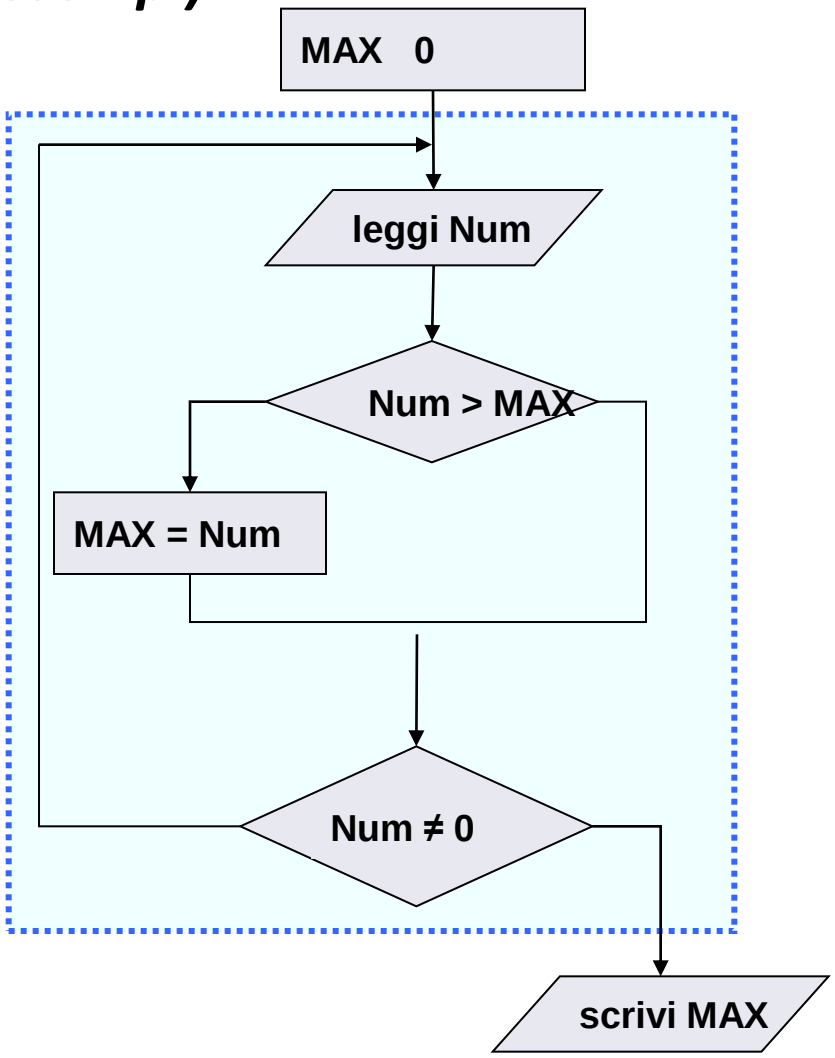
- Analogamente al *while*, per evitare il ciclo infinito, <istruzione> deve modificare prima o poi qualche variabile che compare nella condizione
- Si noti che, come nel caso del *while*, si esce dal ciclo quando la condizione è falsa
- **È adatta** a quei casi in cui, per valutare condizione, è necessario 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;
Leggi Num;
While (Num ≠ 0)
{  if (Num > MAX)
                MAX = Num;
    leggi Num;
}
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*)
  - 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