Diagrammi di flusso

oppure *Flowcharts*

o anche *Diagrammi a blocchi*

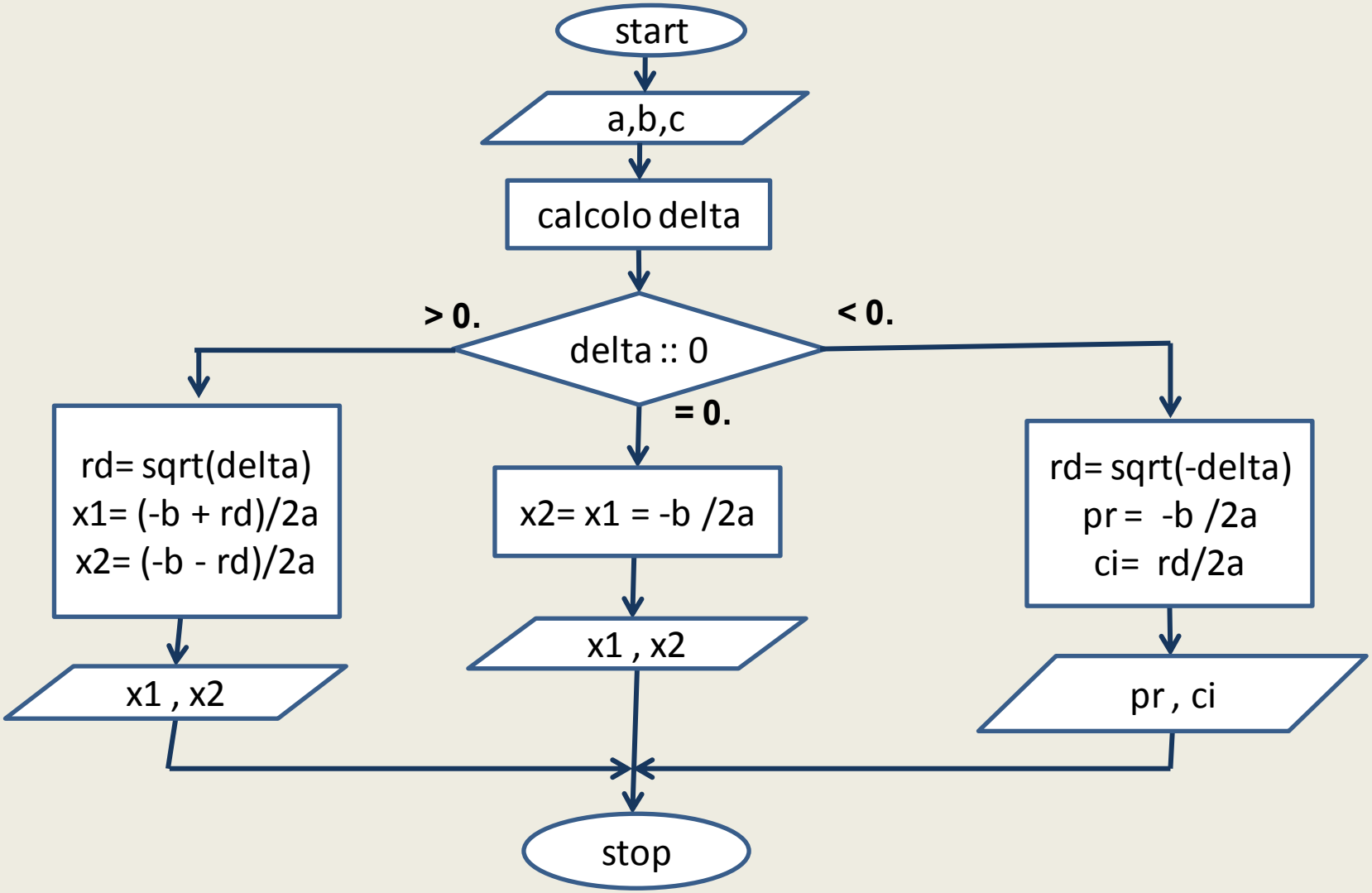
Il “linguaggio” dei **diagrammi di flusso** (**flow chart** in inglese) è un linguaggio grafico specifico per **descrivere processi** .

Le diverse fasi del processo di elaborazione della informazione, e le differenti condizioni che devono essere rispettate nel passaggio da una fase alla successiva, vengono rappresentati da simboli grafici detti **blocchi elementari**. I blocchi sono collegati tra loro tramite **frecce** che indicano la cronologia, cioè l’ordine in cui le azioni rappresentate dai blocchi debbono susseguirsi.

I diagrammi di flusso trovano applicazione anche in ambiti diversi (in campo industriale, in campo economico) ma storicamente sono sempre stati usati in informatica per **descrivere algoritmi di calcolo**

Un altro linguaggio (non grafico) comunemente usato per la descrizione di algoritmi è il cosiddetto *pseudocodice* o *pascal-like*

ESEMPIO : questo diagramma di flusso descrive il procedimento per la risoluzione di un' equazione algebrica di secondo grado.



Algoritmo

procedimento per la risoluzione di un problema descritto come sequenza di azioni elementari definite in maniera non ambigua e tali da portare alla risoluzione del problema in un numero finito di passi.

--- Ogni istruzione (ordine di eseguire una azione) deve essere comprensibile a chi eseguirà l'algoritmo in maniera non ambigua (deve appartenere ad un insieme di comandi predefinito)

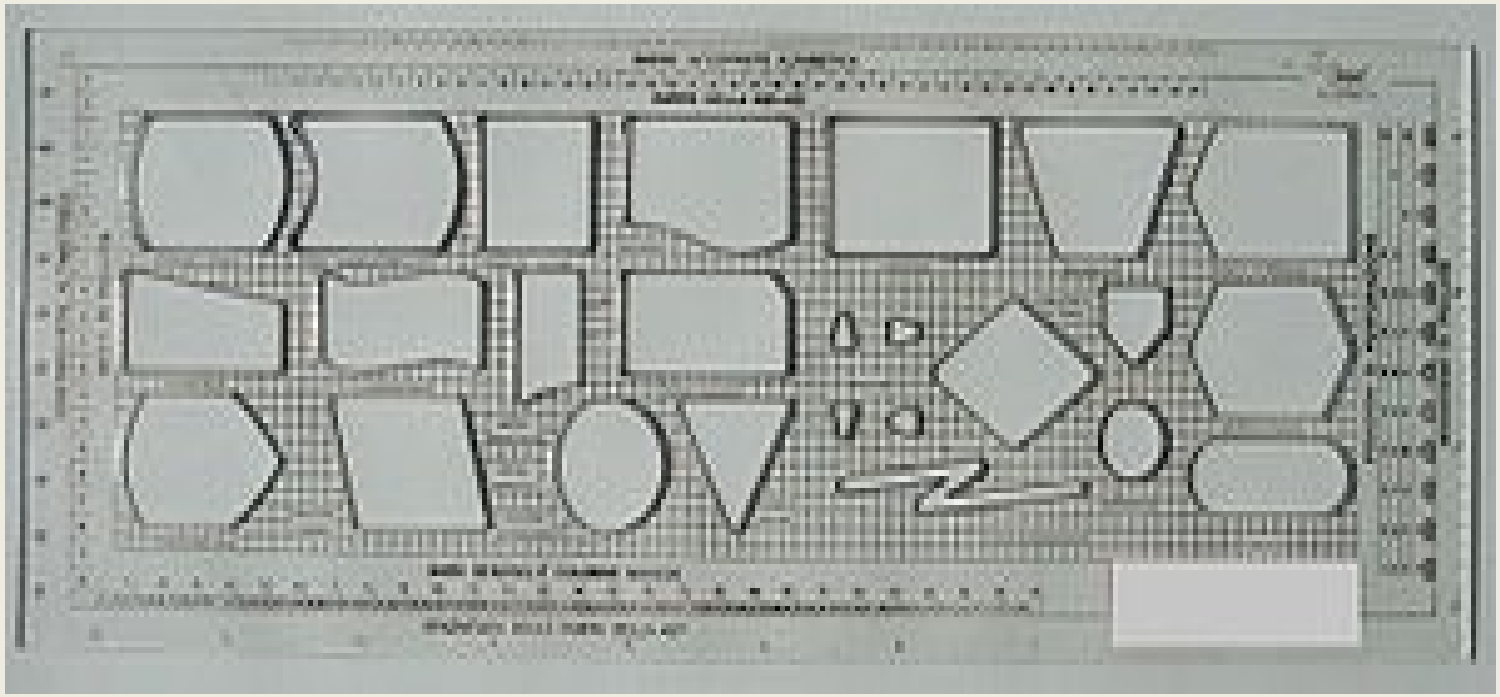
--- Ogni istruzione deve poter essere eseguita individualmente con lo strumento disponibile

--- La sequenza delle istruzioni deve essere finita.

--- Il numero di passi deve essere finito.

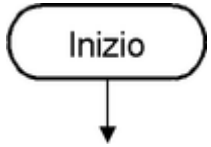
--- Deve essere definito l' input per l'algoritmo e l'output che esso produce

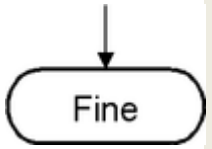
Un reperto "archeologico": il righello per tracciare flowcharts

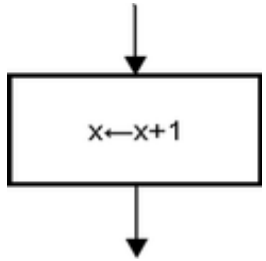


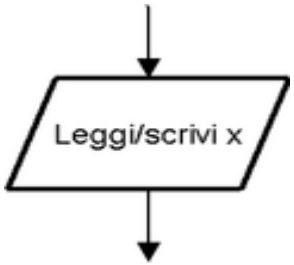
Blocchi elementari

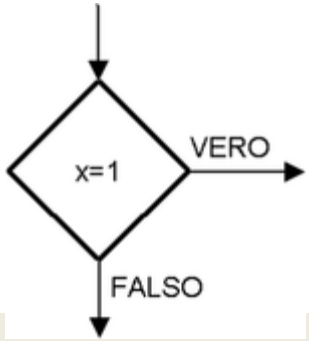
Tipi fondamentali :

1- connettori : es. blocco iniziale 

e blocco finale 

2 - blocco elaborazione (o azione) 

3 - blocco I/O 

4 - blocco di controllo (decisione) 

Una combinazione/successione di blocchi elementari (= diagramma) **descrive un algoritmo** se:

- viene usato un **numero finito** di blocchi
- lo schema inizia con un *blocco iniziale* e termina con un *blocco finale*
- ogni blocco soddisfa alle **condizioni di validità**

Condizioni di validità :

sui blocchi:

1. ogni *blocco azione* e ogni *blocco lettura/scrittura* ha una sola freccia entrante e una sola freccia uscente

2. ogni *blocco di controllo* ha una sola freccia entrante ed almeno due frecce uscenti

sulle frecce:

1. ogni *freccia* o entra in un blocco o confluisce in un'altra freccia

sui percorsi:

1. dal *blocco inizio* deve essere possibile raggiungere ogni blocco

2. da ogni blocco dev'essere possibile raggiungere il *blocco finale*

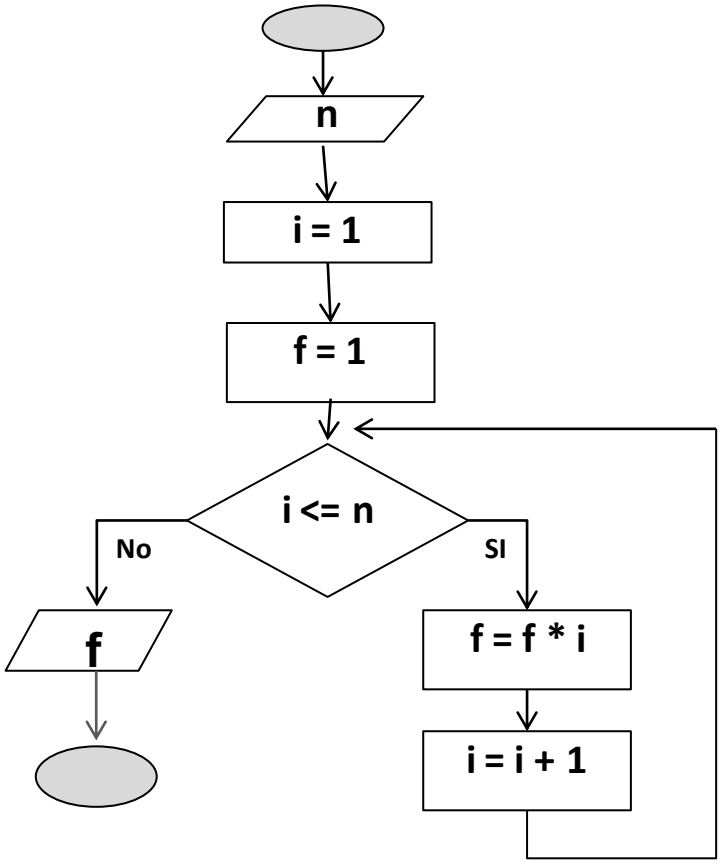
Come “leggere” il diagramma di flusso per eseguire l’ algoritmo

1. si inizia dal blocco ‘*inizio’*;
2. si seguono le frecce;
3. si esegue l’azione indicata in ogni ‘*blocco azione*’, e in ogni ‘*blocco I/O’* che si incontra;
4. si valuta l’espressione logica di ogni ‘*blocco decisione*’ che si incontra e si segue la freccia etichettata con il valore risultante dalla valutazione;
5. ci si arresta quando si incontra il blocco ‘*fine*’.

Esempio : calcolo del fattoriale di un numero intero n

[n! = 1 * 2 * … * n-2 * n-1 * n]

Algoritmo iterativo : **diagramma di flusso** ed esempio di esecuzione per n = 3



Esempio di esecuzione dell'algoritmo.

- Input : n (3) ;
- Elaborazione : i=1 ;
- Elaborazione : f=1 ;
- Decisione : 1 <= 3 ? (SI)
- Elaborazione : f=1*1 quindi f=1;
- Elaborazione : i=1+1 quindi i=2;
- Decisione : 2 <= 3 ? (SI);
- Elaborazione : f=1*2 quindi f=2;
- Elaborazione : i=2+1 quindi i=3;
- Decisione : 3 <= 3 ? (SI) ;
- Elaborazione : f=2*3 quindi f=6;
- Elaborazione : i=3+1 quindi i=4;
- Decisione : 4 <= 3 ? (NO) ;
- Output : f (6)

Fattoriale di n: codifica in Fortran 90 dell'algoritmo nel flowchart

```
PROGRAM fattoriale
!-------------------------------------------------
!Questo programma calcola il fattoriale di un numero
!intero positivo  fornito in input
!N.B.  QUESTO PROGRAMMA USA IL "GO TO" CHE IN
!     GENERALE E' ALTAMENTE SCONSIGLIATO !!
!-------------------------------------------------

!----------------Parte dichiarazioni----------------
!! tutte le variabili utilizzate nel programma
! sono dichiarate esplicitamente

IMPLICIT NONE
! elenco delle variabili con dichiarazione del tipo

INTEGER  :: n, i      ! i è un contatore
INTEGER  :: fatt
! REAL :: FATT  ( provare a -----------------
! Sollecita l'input del dato
WRITE (*,*)
WRITE (*,*) &
'Inserire il valore di cui si vuole il fattoriale'
WRITE (*,*)
READ  (*,*) n
```

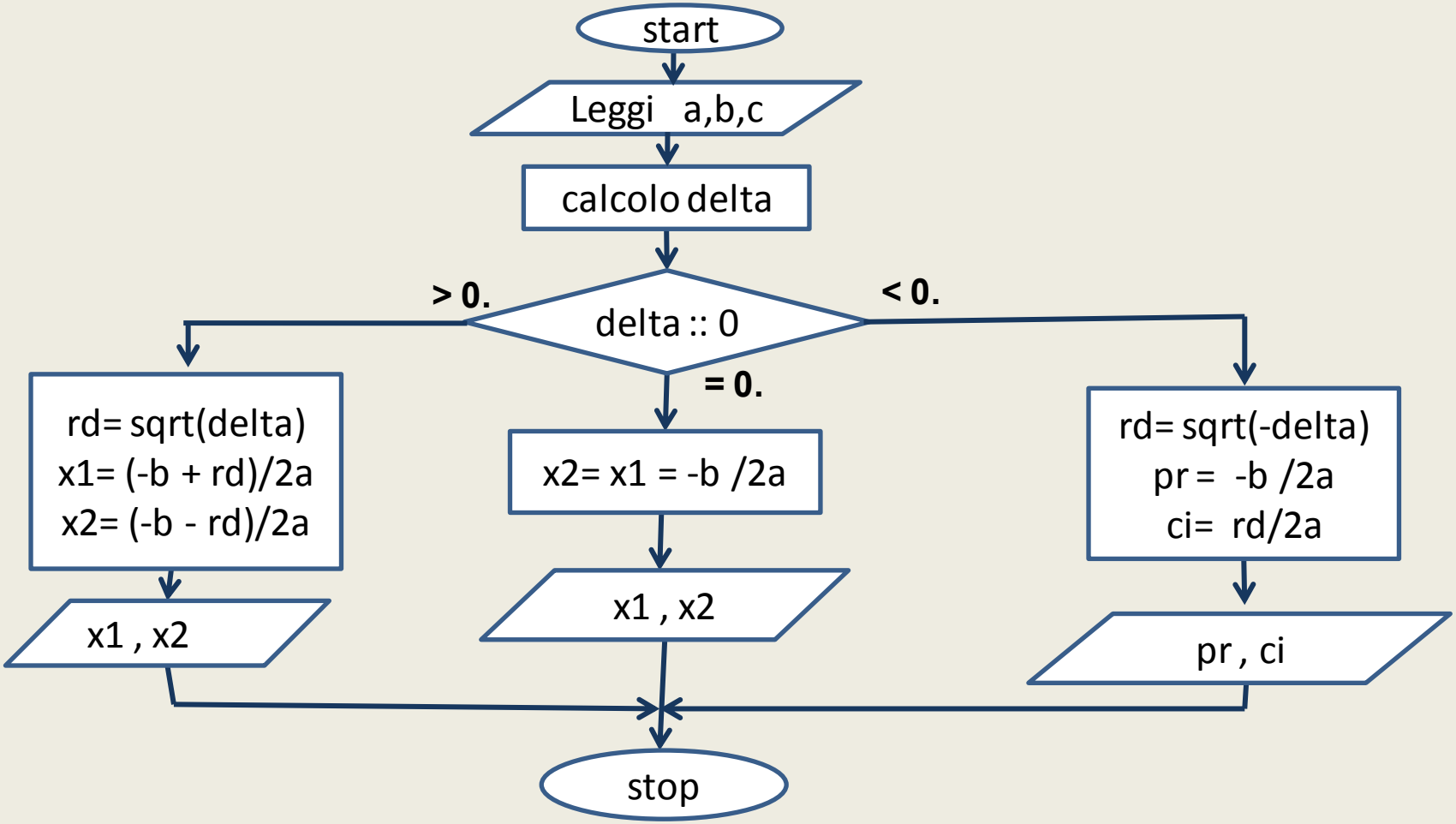
```
! Output dei coefficienti per la verifica
WRITE (*,*)
WRITE (*,*) 'Il valore inserito e''', n
WRITE (*,*)

! inizializza  il calcolo
i = 1
fatt = 1

! controllo dell'iterazione
100 IF ( i > n) THEN        !  il calcolo è completo
    WRITE (*,*)
    WRITE (*,*) 'Il fattoriale di ', n , ' e''', fatt
    WRITE (*,*)
    STOP
  ELSE   ! iterazione
    fatt = fatt * i
    i = i + 1
  END IF
GO TO 100

END PROGRAM fattoriale
```

ESEMPIO: algoritmo per la risoluzione dell'equazione algebrica di secondo grado



Strutture algoritmiche elementari

oppure *'strutture algoritmiche di base'*

o anche *'costrutti fondamentali'*

Strutture algoritmiche elementari

Le tre strutture fondamentali di controllo del flusso delle operazioni, ed i corrispondenti *costrutti base di un linguaggio di programmazione,* sono le *tessere* elementari che permettono di descrivere qualsiasi algoritmo. Esse sono:

- ***sequenza di istruzioni***
- ***selezione***
- ***iterazione***

Il teorema di Boehm e Jacopini

Problema : Le strutture elementari di sequenza, selezione e iterazione sono utili per la costruzione di algoritmi; ma possiamo affermare che usando solo strutture di controllo del tipo sequenza, selezione e iterazione siamo in grado di costruire qualsiasi algoritmo? La risposta è si, infatti…

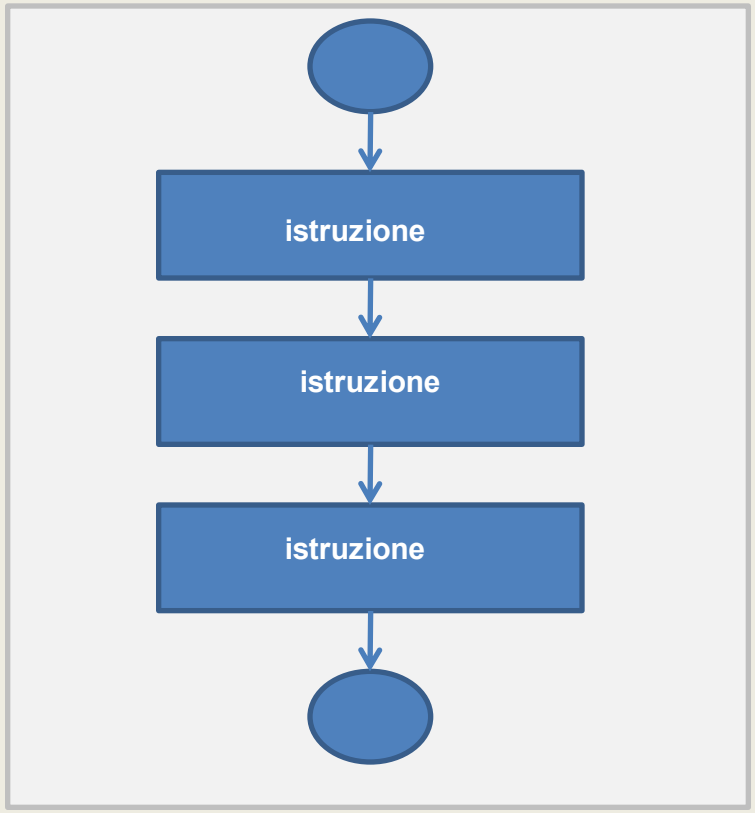
Definizione: Due *algoritmi* sono *equivalenti* se, eseguiti sugli stessi dati in ingresso, producono gli stessi risultati, per qualsiasi set di dati in ingresso.

Teorema di Boehm Jacopini

Dato un algoritmo, è sempre possibile costruirne un altro equivalente composto esclusivamente con strutture di controllo elementari del tipo sequenza, selezione, iterazione.

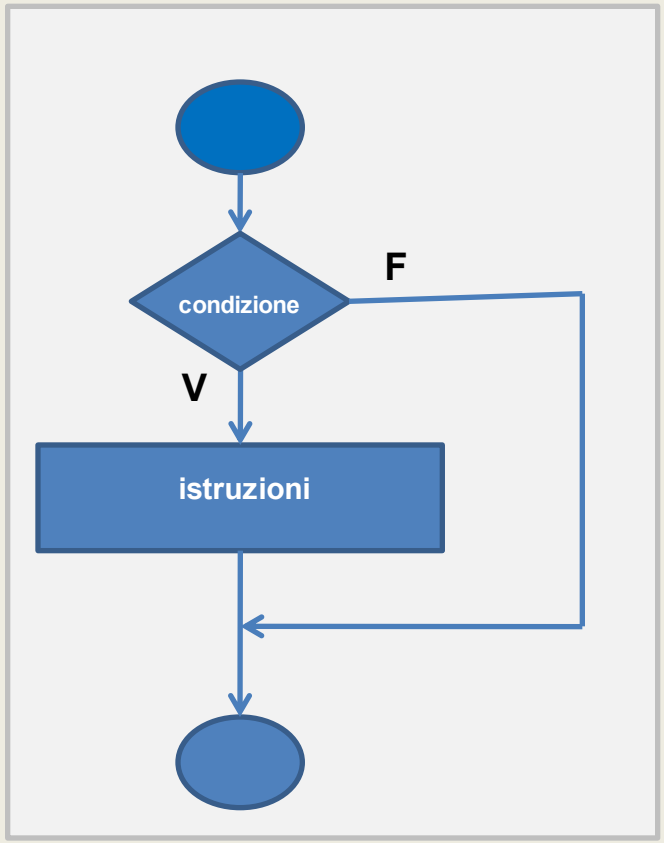
Strutture algoritmiche elementari

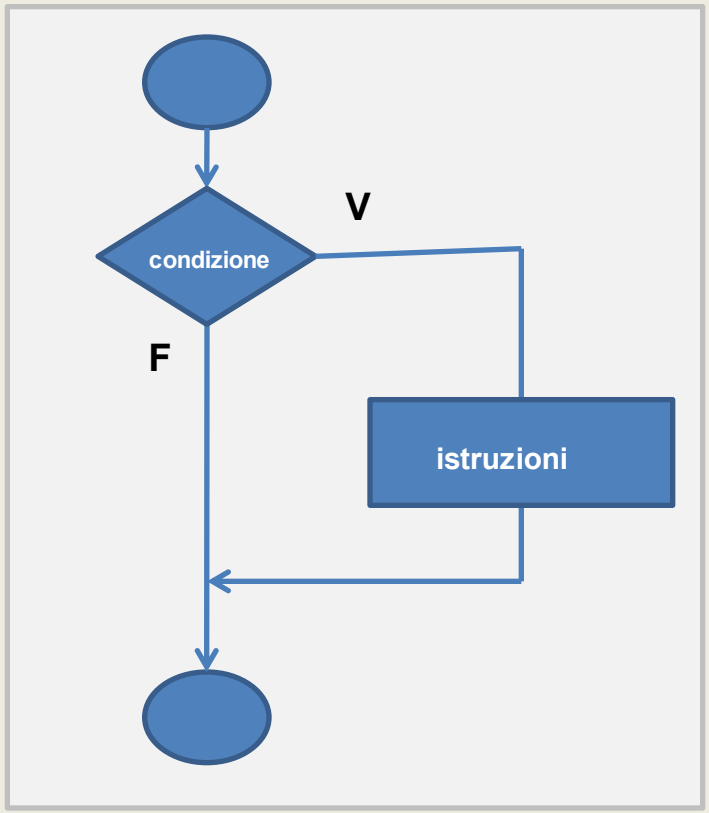
sequenza di istruzioni



Strutture algoritmiche elementari

selezione (N.B. le due forme **sono equivalenti**)

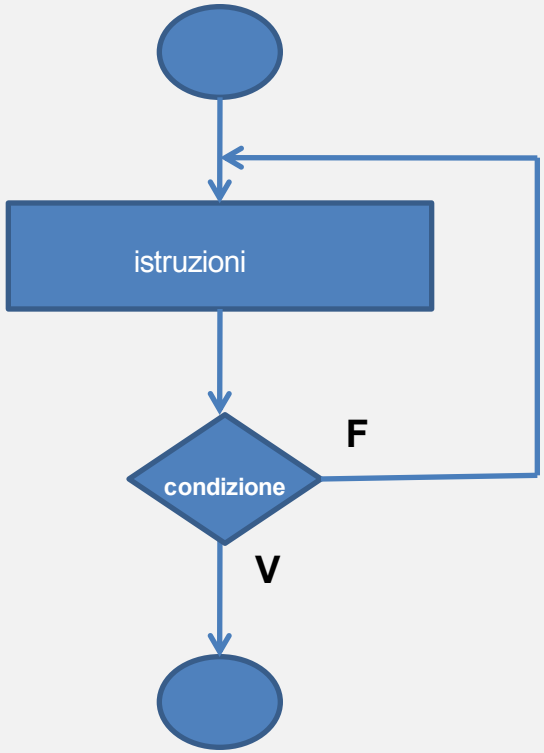




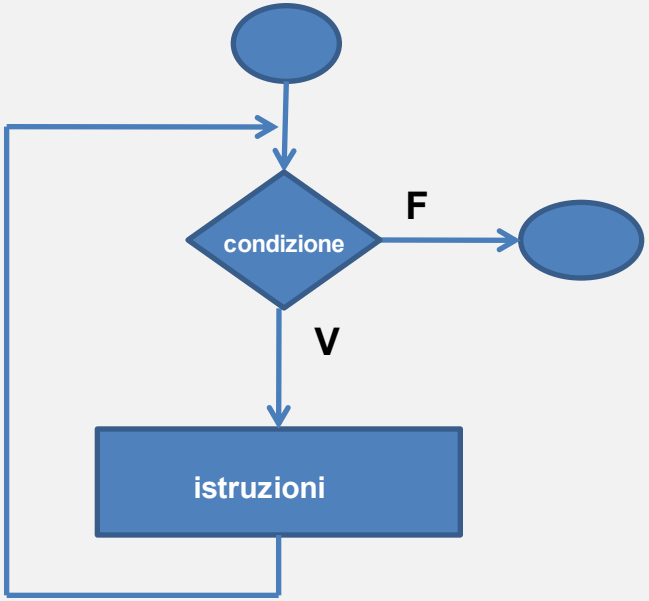
Strutture algoritmiche elementari

iterazione (n.b. le due forme **non sono equivalenti**)

repeat ... until



while ... do



Importanza del teorema di Boehm e Jacopini per la progettazione dei linguaggi di programmazione

Affinché un linguaggio simbolico di programmazione permetta di implementare qualsiasi algoritmo (cioè affinché permetta di scrivere il programma che istruisce la macchina ad eseguire qualsiasi algoritmo di calcolo) è sufficiente che in quel linguaggio siano disponibili costrutti di base corrispondenti a tutte e tre le strutture algoritmiche elementari: sequenza, selezione, iterazione.

Ora possiamo iniziare lo studio
di un linguaggio specifico:

IL FORTRAN