Cosa si intende con stato

- una particolare configurazione delle informazioni di una macchina, che in qualche modo "memorizza" le condizioni in cui si trova, e che cambia nel tempo passando ad un'altra configurazione, in funzione dei segnali d'ingresso.
- Ad esempio, lo stato può rappresentare la posizione dell'ascensore ad un certo piano di un edificio. In base allo stato si determina il modo in cui l'ascensore si muove: se si intende andare al terzo piano e ci troviamo al primo piano, occorre che l'ascensore salga.
- un sistema è *stateful* se "ricorda' gli eventi precedenti; le informazioni ricordate sono chiamate lo *stato* del sistema.

Il concetto di stato

- La specifica (astratta) di un problema consiste nella descrizione di uno stato iniziale (che descrive i dati del problema) e di uno stato finale (che descrive i risultati attesi).
- Si deve individuare una rappresentazione degli oggetti coinvolti (e dunque dello stato) direttamente manipolabile dall'esecutore.
- Un algoritmo è una sequenza di passi elementari che, se eseguiti, comportano ripetute modifiche dello stato fino al raggiungimento dello stato finale desiderato.
- Le azioni di base che l'esecutore è in grado di effettuare devono dunque prevedere, tra le altre, azioni che hanno come effetto cambiamenti dello stato.
- Possiamo pensare che lo stato sia il contenuto della memoria e che il cambiamento dello stato abbia come effetto il cambiamento di alcune posizioni della memoria.

Un'astrazione dello stato

Dato che il calcolo procede attraverso l'elaborazione dello stato, si devono poter esprimere valori dipendenti dallo stato.

Astrazione del concetto di Stato

Uno stato è un insieme di associazioni tra nomi simbolici e valori.

In uno stato, ad ogni nome simbolico è associato al più un valore.

Rappresentiamo una associazione tra il nome simbolico x ed il valore val con la notazione

$$x \rightsquigarrow val$$

Il valore di x dipende dallo stato corrente, che gli associa un particolare valore val.

Lo stato: esempi

Stati corretti

- $\{\text{nome} \rightsquigarrow \text{Antonio}, \text{cognome} \rightsquigarrow \text{Rossi}, \text{età} \rightsquigarrow 25\}$
- $\{\text{importo} \rightsquigarrow \$1650, \text{tasso} \rightsquigarrow 10\%, \text{interesse} \rightsquigarrow \$165\ \}$
- $\{\text{a} \rightsquigarrow 25, \text{b} \rightsquigarrow 3, \text{c} \rightsquigarrow 50\}$

Stati non corretti

- $\{\text{nome} \rightsquigarrow \text{Antonio}, \text{nome} \rightsquigarrow \text{Paolo}, \text{età} \rightsquigarrow 25\}$
- $\{\text{b} \rightsquigarrow 45, \text{a} \rightsquigarrow 150, \text{b} \rightsquigarrow 10\}$

Prima introduzione al linguaggio C

- Abbiamo visto come un programma non sia altro che un algoritmo codificato in un linguaggio di programmazione.
- Problema: quale linguaggio scegliere per la codifica di un algoritmo?
  - Il linguaggio naturale sarebbe facilmente comprensibile ma non è eseguibile da una macchina.
  - Il linguaggio macchina è eseguibile ma di difficile comprensione.
- Due requisiti fondamentali di un qualsiasi linguaggio per la descrizione di algoritmi:
  - deve essere preciso per non lasciare adito a dubbi interpretativi
  - deve essere sintetico per non rendere difficile la comprensione dei programmi.

- Il linguaggio naturale e il linguaggio macchina si collocano in posizioni opposte, soddisfacendo uno solo dei requisiti.
- I linguaggi di programmazione ad alto livello sono progettati proprio per colmare tale divario.
  $\Longrightarrow$ sono linguaggi adatti a codificare algoritmi pur rimanendo comprensibili.
- La fatica di tradurre un programma nel linguaggio macchina è affidata a particolari programmi, i compilatori, che traducono programmi scritti nel linguaggio di più alto livello in programmi equivalenti nel linguaggio macchina.

Il linguaggio C

- Introduciamo inizialmente l'insieme di costrutti linguistici che costituiscono il nucleo di un qualunque linguaggio di programmazione reale, usando già la sintassi del C.
- Senza entrare in eccessivi dettagli formali, nel presentare le notazioni utilizzate (sintassi) diamo anche una descrizione informale (semantica) di ciò che accade al momento dell'esecuzione in corrispondenza dei vari costrutti.

Il linguaggio contiene costrutti per:

- rappresentare semplici calcoli attraverso le comuni operazioni logico/aritmetiche (espressioni)
- modificare le associazioni nello stato (assegnamento e ingresso)
- controllare l'ordine di esecuzione delle azioni (controllo)
- fornire i risultati (produzione in uscita dello stato finale)

Espressioni

Il ruolo delle espressioni è quello di denotare valori (dip. dallo stato).

Espressioni Numeriche

Il linguaggio consente di rappresentare semplici calcoli algebrici, attraverso le usuali espressioni costruite a partire dai valori numerici e dalle operazioni di

- somma $+$
- sottrazione -
- prodotto *
- divisione intera /
- modulo o resto della divisione intera %.

Il significato di un'espressione è il suo valore ottenuto secondo le usuali regole di calcolo.

Esempio:

- il valore di $3 * 5 + 6$ è 21
- il valore di $3 * (5 + 6)$ è 33

Espressioni (cont.)

Oltre ai valori numerici, le espressioni possono contenere nomi simbolici: il calcolo di un'espressione che contiene un nome simbolico x dipende dallo stato, e precisamente dal valore associato al nome x nello stato.

Esempio:
il valore di $3 * (5 + x)$ è

- 33 in uno stato che contiene l'associazione $x \rightsquigarrow 6$
- 18 in uno stato che contiene l'associazione $x \rightsquigarrow 1$

Gli operatori possiedono regole di precedenze che determinano come avviene la valutazione delle espressioni. Come nell'aritmetica tradizionale, + e - hanno lo stesso grado di priorità, inferiore a quello di *,/ e %.

Espressioni (cont.)

Espressioni Booleane

Si possono rappresentare condizioni ovvero espressioni il cui valore è un valore di verità (true o false).

Le condizioni sono costruite attraverso le usuali operazioni di confronto (==, !=, <, >, <=, >=) e, come nel caso delle espressioni aritmetiche, il loro valore può dipendere dallo stato.

Esempio: il valore di y==x+1 è

- true in uno stato che contiene le associazioni $x \rightsquigarrow 5$ e $y \rightsquigarrow 6$
- false in uno stato che contiene le associazioni $x \rightsquigarrow 5$ e $y \rightsquigarrow 9$

Condizioni più complesse possono essere costruite attraverso operatori logici quali negazione (simbolo !), congiunzione (simbolo &&) e disgiunzione (simbolo ||).

Espressioni (cont.)

- Significato di ! - il valore di verità di ! P è
  - true se il valore di verità di P è false
  - false se il valore di verità di P è true
- Significato di && - il valore di verità di P && Q è
  - true se i valori di verità di P e Q sono entrambi true
  - false altrimenti
  - Se P è false si restituisce false senza valutare Q
- Significato di || - il valore di verità di P || Q è
  - false se i valori di verità di P e Q sono entrambi false
  - true altrimenti
  - Se P è true si restituisce true senza valutare Q
- || and && hanno lo stesso grado di priorità, inferiore a quello di !

Esempio:

- Il valore di `(y >= x) &&(x > 5)` è
  - true in uno stato che contiene le associazioni x $\rightsquigarrow$ 15 e y $\rightsquigarrow$ 30
  - false in uno stato che contiene le associazioni x $\rightsquigarrow$ 3 e y $\rightsquigarrow$ 30
- Il valore di `(y >= x) || (x > 5)` è
  - true in uno stato che contiene le associazioni x $\rightsquigarrow$ 2 e y $\rightsquigarrow$ 30
  - false in uno stato che contiene le associazioni x $\rightsquigarrow$ 3 e y $\rightsquigarrow$ 1

Tavole di verità dei principali operatori logici

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \iff q$	$p \Leftarrow q$
F	F	T	F	F	T	T	T
F	T	T	F	T	T	F	F
T	F	F	F	T	F	F	T
T	T	F	T	T	T	T	T

Lo stato e le variabili

Lo stato rappresenta il contenuto (modificabile) della memoria.

Una posizione della memoria, destinata a contenere dati, si identifica con una variabile Le variabili dunque rappresentano, nei programmi, le associazioni dello stato
$\Longrightarrow$ cf. $x \rightsquigarrow val$ introdotto prima

Il valore di x dipende dallo stato corrente, che gli associa un particolare valore val.

Le associazioni possono essere modificate

Programmazione imperativa e modifica dello stato

- Nella programmazione imperativa, la computazione viene descritta in termini di stato di programma.
- Il ruolo delle istruzioni (I_i) è quello di modificare, con la loro esecuzione, lo stato.

$Stato_{in}$	$I_1 \ldots I_n$	$Stato_1$	$I_1 I_2 \ldots I_n$	$Stato_2$	...	$Stato_{n-1}$	$I_1 I_2 \ldots I_n$	$Stato_{fin}$

Modifica dello stato: ASSEGNAMENTO

- Una delle istruzioni che consentono di rappresentare modifiche di stato è l'assegnamento.
- L'assegnamento consente di cambiare un'associazione nello stato, ovvero il valore associato nello stato ad un nome simbolico.
- Useremo per l'assegnamento la seguente notazione

  $$\boxed{\texttt{x = exp;}}$$

  dove x è un nome simbolico (la variabile in C) e exp una espressione.
- L'esecuzione dell'assegnamento x = exp; consiste nel
  (i) calcolare il valore, sia esso val, dell'espressione exp
  (ii) introdurre nello stato l'associazione $x \rightsquigarrow val$
- Si noti che (ii) comporta la rimozione dallo stato della eventuale associazione già presente per x (si parla a questo proposito di assegnamento distruttivo).

ASSEGNAMENTO: esempi

Vediamo alcuni esempi, indicando lo stato prima e dopo l'esecuzione degli assegnamenti proposti. Le associazioni modificate nello stato finale sono evidenziate in verde.

Stato iniziale	Assegnamento	Stato Finale
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20 \}$	x = 5;	$\{ x \rightsquigarrow 5, y \rightsquigarrow 20 \}$
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20 \}$	x = y*2;	$\{ x \rightsquigarrow 40, y \rightsquigarrow 20 \}$
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20 \}$	x = x+1;	$\{ x \rightsquigarrow 11, y \rightsquigarrow 20 \}$

Si noti come, nel terzo esempio, lo stesso nome simbolico x giochi un duplice ruolo:

- a destra del simbolo = indica un valore (il valore associato ad x nello stato iniziale)
- a sinistra del simbolo = indica l'associazione da modificare nello stato a seguito dell'assegnamento.

Modifica dello stato: INPUT

- La seconda istruzione di modifica dello stato consente di acquisire valori dal mondo esterno al momento dell'esecuzione. La notazione è

  `scanf(...,&x);`

  dove x indica un nome simbolico, di cui &x rappresenta l'indirizzo in memoria, e in ... troveremo la stringa usata per il controllo del formato (aspetti su cui torneremo in seguito).
- L'esecuzione consiste nel:
  (i) Acquisire un nuovo valore, sia esso val (quando viene eseguita il programma si mette in attesa che l'utente immetta un valore.)
  (ii) Introdurre nello stato l'associazione $x \rightsquigarrow val$
- Come nel caso dell'assegnamento, il punto (ii) comporta la rimozione dallo stato dell'eventuale associazione già presente per il nome simbolico (variabile in C) x
- La presenza di tale istruzione permette di descrivere algoritmi generali in cui non tutti i dati sono noti a priori, ma lo saranno solo al momento dell'esecuzione.

Istruzioni di controllo: SEQUENZA

- Negli esempi visti in precedenza gli algoritmi sono stati descritti come sequenze di passi elementari del tipo

  Passo 1. azione 1
  Passo 2. azione 2
  ...

- Abbiamo utilizzato una sorta di numerazione per indicare l'ordine di esecuzione delle varie azioni: prima azione 1 poi azione 2 poi ....
- Nel linguaggio che stiamo introducendo, e in C, una sequenza di azioni viene rappresentata mediante un blocco

```
{
istruzione 1
istruzione 2
...
istruzione n
}
```

SEQUENZA (cont.)

- Scriveremo dunque

  {
  istruzione 1
  istruzione 2
  ...
  }

  ad indicare che l'ordine di esecuzione dei singoli passi è quello testuale del programma.
- Si noti che ogni istruzione viene eseguita a partire dallo stato risultante dall'esecuzione dell'istruzione che la precede nella sequenza.
- Assegnamento e ingresso sono istruzioni semplici il blocco è un'istruzione composta.

SEQUENZA: esempi

Stato iniziale	Sequenza	Stato Finale
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	$\{ x = 5; x = x+y; \}$	$\{ x \rightsquigarrow 25, y \rightsquigarrow 20\}$
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	$\{ x = x+1; y = x+1; \}$	$\{ x \rightsquigarrow 11, y \rightsquigarrow 12\}$
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	$\{ x = x+y; y = x+y; \}$	$\{ x \rightsquigarrow 30, y \rightsquigarrow 50\}$

- In tutti gli esempi, lo stato finale si ottiene dall'esecuzione del secondo assegnamento nello stato intermedio risultante dall'esecuzione del primo assegnamento.
- Ad esempio, nel terzo caso:

Stato iniziale	Prima istruzione	Stato Intermedio
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	$x = x+y;$	$\{ x \rightsquigarrow 30, y \rightsquigarrow 20\}$
Stato intermedio	**Seconda istruzione**	**Stato Finale**
$\{ x \rightsquigarrow 30, y \rightsquigarrow 20\}$	$y = x+y;$	$\{ x \rightsquigarrow 30, y \rightsquigarrow 50\}$

Istruzioni di controllo: CONDIZIONALE

- Permette di determinare l'azione da intraprendere a seconda del verificarsi o meno di una condizione. La notazione utilizzata è la seguente

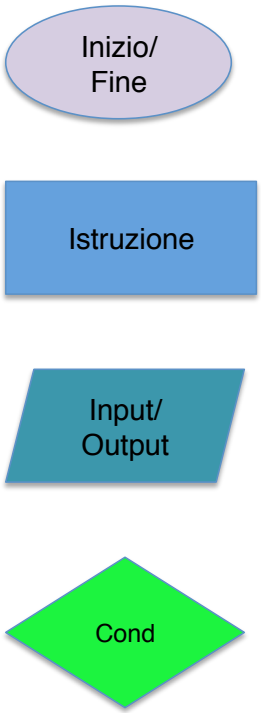
  if (condizione) istruzione1 **else** istruzione2

  dove condizione indica una espressione booleana, e istruzione1 istruzione2 sono istruzioni (semplici o composte).
- L'esecuzione del condizionale **if** (C) S1 **else** S2 consiste nel
  (i) Calcolare il valore, sia esso val, dell'espressione booleana C
  (ii) Eseguire S1 se val è true, eseguire S2 se val è false.
- Si noti che la presenza dell'istruzione condizionale non è in conflitto con il requisito di non ambiguità degli algoritmi: l'azione da intraprendere è univocamente determinata dal valore di verità della condizione (nello stato dato) e dunque l'esecutore non deve scegliere.
- Possiamo anche avere **if** annidati (in cascata) quando l'istruzione del ramo **then** o **else** è un'istruzione **if** o **if-else**.

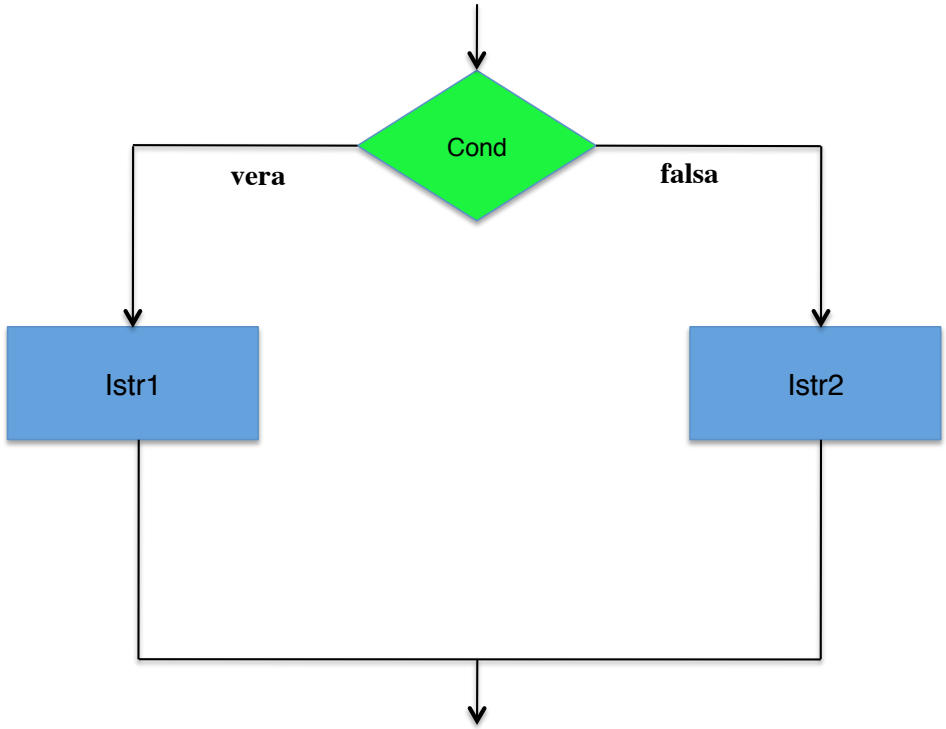
CONDIZIONALE: esempi

Stato iniziale	Condizionale	Stato Finale
	if (x > y) z = x; **else** z = y;	
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20 \}$		$\{ x \rightsquigarrow 10, y \rightsquigarrow 20, z \rightsquigarrow 20 \}$
$\{ x \rightsquigarrow 10, y \rightsquigarrow 5 \}$		$\{ x \rightsquigarrow 10, y \rightsquigarrow 5, z \rightsquigarrow 10 \}$

Diagrammi di flusso: forme



if cond istr1 **else** istr2



Istruzioni di controllo: CONDIZIONALE (2)

- Esiste anche un'istruzione **if-else** in cui manca la parte **else**, nella quale se la condizione è vera, viene eseguita l'istruzione, altrimenti non si fa niente

```
if (condizione) istruzione1
```

- Esiste inoltre una selezione a più vie (vedi **switch**).

Iterazione determinata e indeterminata

- Le istruzioni iterative permettono di ripetere determinate azioni più volte:
  - un numero di volte fissato $\Longrightarrow$ iterazione determinata
    Esempio:
    ```
    fai un giro del parco di corsa per 10 volte
    ```
  - finchè una condizione rimane vera $\Longrightarrow$ iterazione indeterminata
    Esempio:
    ```
    finché non sei sazio
    prendi una ciliegia dal piatto e mangiala
    ```

Istruzioni di controllo: RIPETIZIONE

- Consente di ripetere l'esecuzione di una istruzione (o sequenza di istruzioni) fino al verificarsi di una certa condizione (cfr. esempio del calcolo del prodotto tra due numeri).

  while (condizione) Istruzione

  dove condizione indica una espressione booleana.
- Terminologia: in **while** (C) S , la condizione C è detta guardia e l'istruzione S è detta corpo (del ciclo).
- L'esecuzione di **while** (C) S consiste nel
  - (i) Calcolare il valore, sia esso val, dell'espressione booleana C
  - (ii) Se val è false, terminare l'esecuzione.
  - (iii) Se val è true, eseguire S e ripetere dal punto (i).
- Nota: se condizione è falsa all'inizio, il ciclo non fa nulla.

RIPETIZIONE

Esempio: Riformulazione del passo 4 dell'algoritmo per il prodotto (versione 2):

```
while (b>0)
{
  c =  c+a;
  b =  b-1;
}
```

- Intuitivamente, l'esecuzione di

  while (guardia) corpo

  corrisponde all'esecuzione di una sequenza del tipo

  { corpo corpo corpo ... corpo ...}

- In tale sequenza, ogni ripetizione dell'istruzione corpo viene detta iterazione del ciclo.

  In generale, non è possibile determinare a priori il numero di iterazioni (che può anche essere 0 nel caso in cui la guardia sia falsa nello stato iniziale), dipende dal verificarsi della guardia.

RIPETITIVO: esempi

```
while (b>0)
{
  c =  c+a;
  b =  b-1;
}
```

Stato iniziale	Stato intermedio	Stato Finale
$\{ a \rightsquigarrow 3, b \rightsquigarrow 2, c \rightsquigarrow 0\}$	$\{ a \rightsquigarrow 3, b \rightsquigarrow 1, c \rightsquigarrow 3\}$	$\{ a \rightsquigarrow 3, b \rightsquigarrow 0, c \rightsquigarrow 6\}$

```
while (somma<16)
{
  somma =  somma+a;
}
```

Stato iniziale	Stato intermedio	Stato Finale
$\{ a \rightsquigarrow 9, somma \rightsquigarrow 0\}$	$\{ a \rightsquigarrow 9, somma \rightsquigarrow 9\}$	$\{ a \rightsquigarrow 9, somma \rightsquigarrow 18\}$

RIPETIZIONE

- L'aspetto più critico è il fatto che l'esecuzione di un ciclo può essere fonte di non terminazione dell'esecuzione dell'intero algoritmo.

  while (3>0) x = 0;

- Un ciclo siffatto provoca la non terminazione dell'esecuzione, dal momento che il valore di verità della guardia è sempre true e dunque l'esecuzione corrisponde ad una sequenza infinita

  x = 0; x = 0; .....; x = 0; .....

- È compito di chi definisce l'algoritmo assicurare che i cicli presenti non diano luogo a non terminazione.
- La pratica programmativa, ed il buon senso, suggeriscono ad esempio di utilizzare cicli in cui:
  - il valore di verità della guardia dipende dallo stato;
  - l'esecuzione del corpo comporta modifiche di associazioni nello stato dalle quali dipende il valore di verità della guardia.

RIPETIZIONE

- Le indicazioni appena viste non sono tuttavia sufficienti a garantire la terminazione dell'esecuzione. Si consideri ad esempio il ciclo:

  while (x>0) x = x+1;

  che soddisfa entrambi i requisiti richiesti, ma che può non terminare nel caso in cui venga eseguito a partire da uno stato dove il valore associato al nome x è un valore positivo.
- A partire dagli anni '70 sono stati sviluppati dei metodi formali per la verifica di correttezza di programmi, che comprendono tecniche per la dimostrazione formale di proprietà di terminazione dei cicli.

Istruzione **for**

Quando il numero di ripetizioni necessarie non è noto al momento della scrittura del programma si può ricorrere al **for**

for (istr-1; espr-2; istr-3)
  istruzione

- istr-1 serve a inizializzare il contatore o variabile di controllo
- espr-2 è la verifica di fine ciclo: si verifica se la variabile di controllo ha raggiunto un limite prefissato
- istr-3 serve ad aggiornare la variabile di controllo alla fine del corpo del ciclo
- istruzione è il corpo del ciclo

Ad esempio: `for (i = 1; i <= 10; i=i+1) x = x+1;`

FOR: esempi

```
while (b>0)
{
  c =  c+a;
  b =  b-1;
}
```

può essere reso dall'equivalente programma con il **for**

```
for (i = 0; i < b; i=i+1)
{
  c =  c+a;
}
```

dove `i` è la variabile di controllo che conta quante volte dobbiamo sommare `a`.

Istruzione **do-while**

- Nell'istruzione **while** la condizione viene controllata all'inizio di ogni iterazione.
- L'istruzione **do-while** è simile all'istruzione **while**, ma la condizione viene controllata alla fine di ogni iterazione

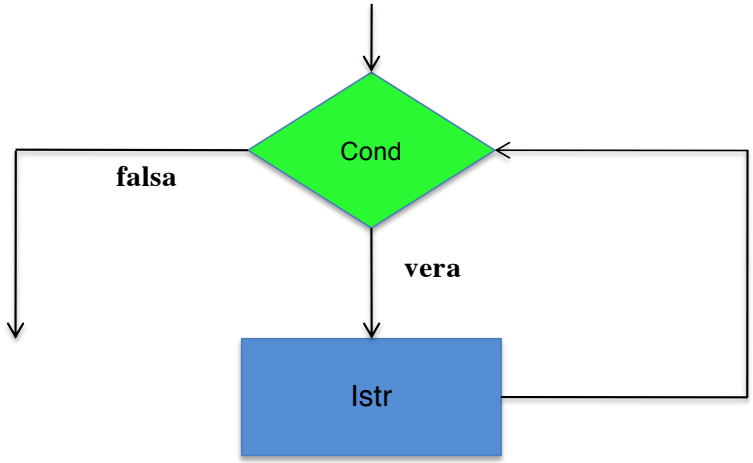
```
do
   istruzione
while (espressione);
```

L'istruzione è *semanticamente* equivalente a

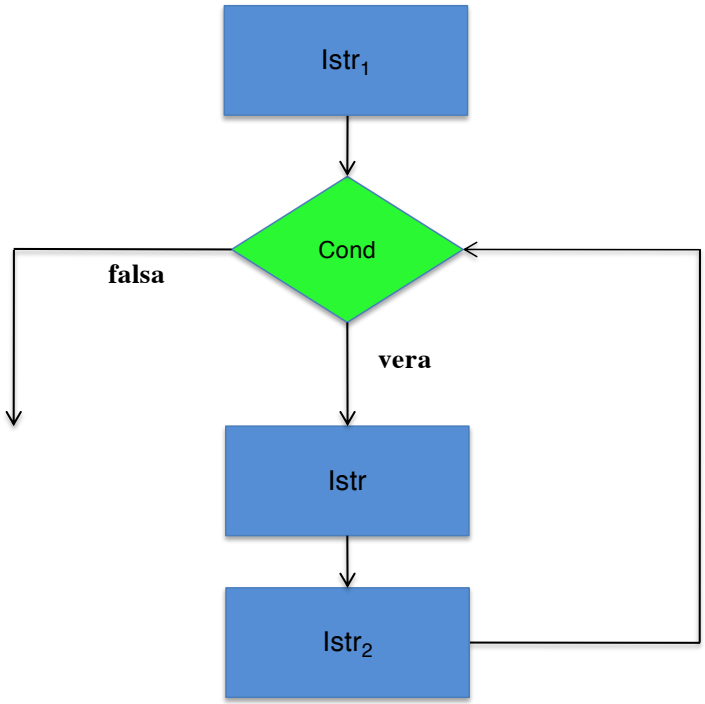
```
istruzione
while (espressione)
  istruzione
```

$\Longrightarrow$ una iterazione viene eseguita comunque.

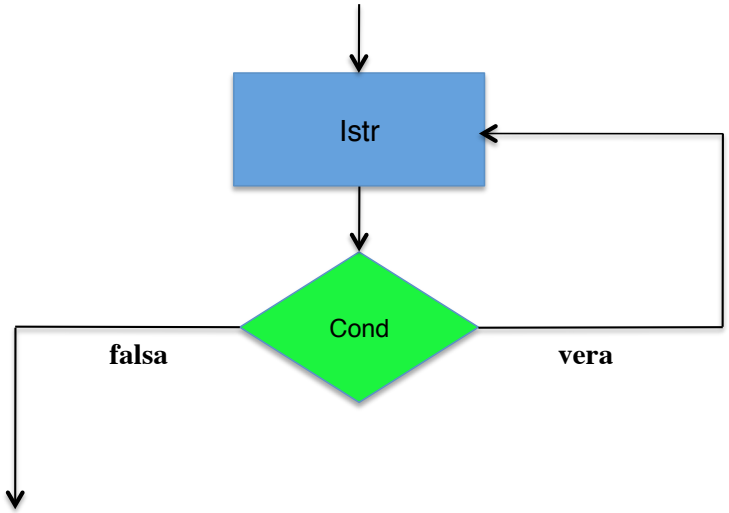
while cond istr



for (istr$_1$: cond; istr$_2$) istr;



do istr **while** cond



Istruzioni di uscita: OUTPUT

- Permette di rendere visibile all'esterno la parte desiderata dello stato. La notazione utilizzata è

  `printf(...,x);`

  dove x indica un nome simbolico e in ... troveremo la stringa usata per il controllo del formato (aspetti su cui torneremo).
- L'esecuzione di questa istruzione consiste nel:
  (i) Recuperare il valore, sia esso val, associato nello stato al nome simbolico x.
  (ii) Produrre in uscita tale valore.
- Questa operazione non comporta la modifica dello stato. Nei linguaggi di programmazione reali, l'esecuzione delle operazioni di uscita produce di solito la visualizzazione di valori su supporti fisici quali video, stampanti etc.

Programmazione strutturata

Si parla di programmazione strutturata se si utilizzano solo le seguenti strutture (concatenate in sequenza o annidate una dentro l'altra, ma non intrecciate tra loro), per alterare il flusso del controllo:

- sequenziale
- condizionale
- iterativa

È stato dimostrato che queste tre strutture sono sufficienti per esprimere un qualsiasi algoritmo. Inoltre ci permettono di:

- scrivere programmi facilmente leggibili e modificabili, e di
- evitare l'uso della programmazione a salti (`go to`) che può portare a quello che si definisce dispregiativamente spaghetti code, una programmazione che porta ad una struttura di controllo del flusso complessa e poco comprensibile.

Attributi degli algoritmi

Un algoritmo deve essere:

- corretto, ovvero deve fornire un risultato corretto;
- composto con i costrutti più opportuni;
- di facile comprensione, per favorire la manutenzione dei programmi;
- possibilmente elegante;
- efficiente in termini di tempo (quanto lavoro o istruzioni occorrono) e di spazio (quanta memoria si occupa). Solitamente l'efficienza si misura in termini di ordine di grandezza.