

ISTRUZIONI

- Le *istruzioni* esprimono *azioni* che, una volta eseguite, comportano una *modifica permanente dello stato interno* del programma o del mondo circostante.
- Le *strutture di controllo* permettono di aggregare istruzioni semplici in istruzioni più complesse.

ISTRUZIONI

- Un'istruzione C è espressa dalle seguenti produzioni:

`<istruzione> ::= <istruzione-semplice>`

`<istruzione> ::= <istruzione-di-controllo>`

`<istruzione-semplice> ::= <espressione> ;`

ISTRUZIONI SEMPLICI

- Qualsiasi *espressione* seguita da un punto e virgola è una *istruzione semplice*.

- Esempi

```
x = 0; y = 1; /* due istruzioni */
```

```
x = 0, y = 1; /* una istruzione */
```

```
k++;
```

```
3; /* non fa nulla */
```

```
; /* istruz. vuota*/
```

ISTRUZIONI DI CONTROLLO

- Una istruzione di controllo può essere:
 - una *istruzione composta* (blocco)
 - una *istruzione condizionale* (selezione)
 - una *istruzione di iterazione* (ciclo)
- Le istruzioni di controllo sono alla base della programmazione strutturata (Dijkstra, 1969).

PROGRAMMAZIONE STRUTTURATA

- **Obiettivo:** rendere più facile la lettura dei programmi (e quindi la loro modifica e manutenzione).
- Abolizione di salti incondizionati (**go to**) nel flusso di controllo.
- La parte esecutiva di un programma viene vista un comando (complesso) ottenuto da **istruzioni elementari**, mediante alcune regole di composizione (**strutture di controllo**).

STRUTTURE DI CONTROLLO

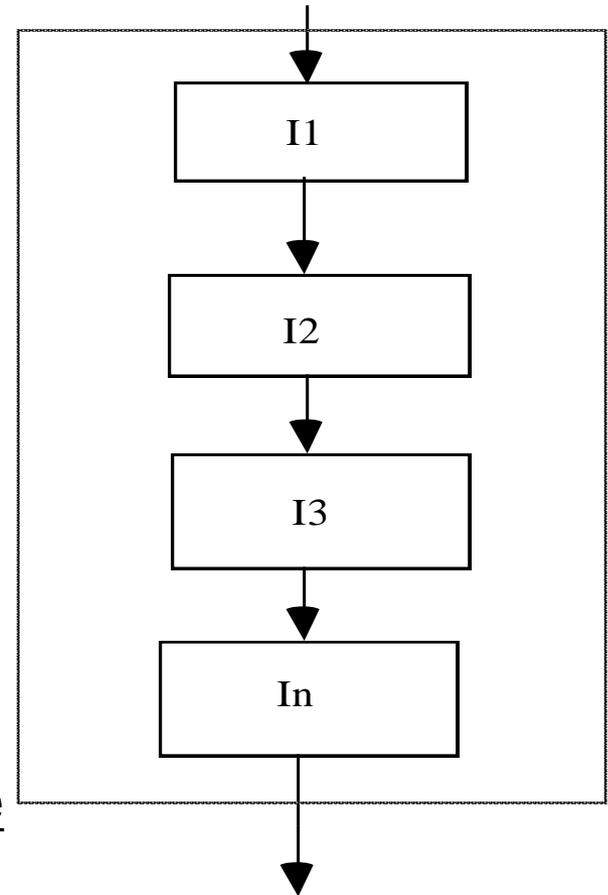
Concetti chiave:

- concatenazione o composizione **BLOCCO**
- istruzione condizionale **SELEZIONE**
 - ramifica il flusso di controllo in base al valore vero o falso di una espressione (“*condizione di scelta*”)
- ripetizione o iterazione **CICLO**
 - esegue ripetutamente un’istruzione finché rimane vera una espressione (“*condizione di iterazione*”)

BLOCCO

<blocco> ::= {
[<dichiarazioni e definizioni>]
{ <istruzione> }
}

- Il campo di visibilità dei simboli del blocco è ristretto al blocco stesso
- dopo un blocco non occorre il punto e virgola (esso *termina* le istruzioni semplici, non *separa* istruzioni)



ESEMPIO di BLOCCO

```
/* programma che letti due numeri a
   terminale ne stampa la somma*/
#include <stdio.h>
int main()
{ /* INIZIO BLOCCO */
int X,Y;
    printf("Inserisci due numeri ");
    scanf("%d%d", &X, &Y);
    printf("%d", X+Y);
return 0;
} /* FINE BLOCCO */
```

REGOLE DI VISIBILITÀ (SCOPE), TEMPI DI VITA VARIABILI E... BLOCCHI

Un identificatore di variabile non è visibile prima della sua dichiarazione

SCORRETTO

```
int main() {  
int x = y*2;  
int y = 5;  
...  
return 0;}
```

CORRETTO

```
int main() {  
int y = 5;  
int x = y*2;  
...  
return 0;}
```

REGOLE DI VISIBILITÀ (SCOPE), TEMPI DI VITA VARIABILI E... BLOCCHI

In ciascun ambiente un identificatore può essere dichiarato una sola volta

```
int main() {  
float x = 3.5;  
char x;  SCORRETTO  
  
...  
return 0;  
}
```

REGOLE DI VISIBILITÀ (SCOPE), TEMPI DI VITA VARIABILI E... BLOCCHI

- *Se in un ambiente sono visibili due dichiarazioni dello stesso identificatore, la dichiarazione valida è quella dell'ambiente più vicino al punto di utilizzo*
- *In ambienti diversi si può dichiarare lo stesso identificatore per denotare due dati diversi*

```
int main() {  
float x = 3.5F;  
  {  
    char x;  CORRETTO  
    x='a';  
  }  
x=5.6F;  CORRETTO  
return 0;  
}
```

REGOLE DI VISIBILITÀ (SCOPE), TEMPI DI VITA VARIABILI E... BLOCCHI

Un identificatore dichiarato in un ambiente è visibile in tutti gli ambienti in esso contenuti

SCORRETTO

```
int main() {  
  int x;  
  {  
    int y = 5;  
  }  
  x = y;  
  ...  
  return 0; }
```

CORRETTO

```
int main() {  
  int x;  
  {  
    int y = 5;  
    x = y;  
  }  
  ...  
  return 0; }
```

Qual è il tempo di vita di y?

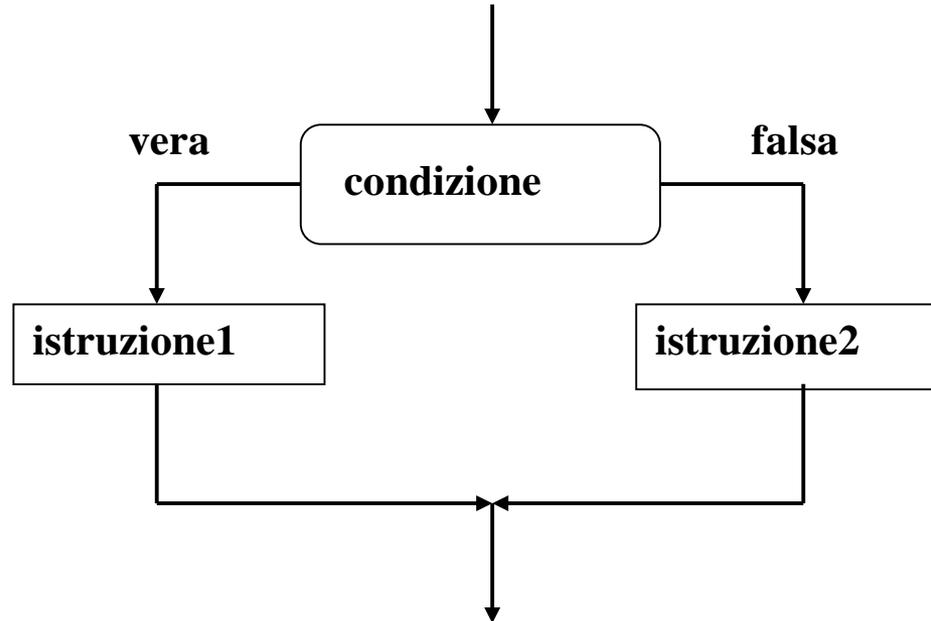
ISTRUZIONI CONDIZIONALI

`<selezione> ::=`
`<scelta> | <scelta-multipa>`

- la seconda *non è essenziale*, ma migliora l'espressività.
- l'espressione condizionale ternaria (`.. ? ... : ...`) fornisce un mezzo per fare scelte, ma è *poco leggibile* in situazioni di medio/alta complessità. L'istruzione di scelta fornisce un altro modo per esprimere alternative.

ISTRUZIONE DI SCELTA SEMPLICE

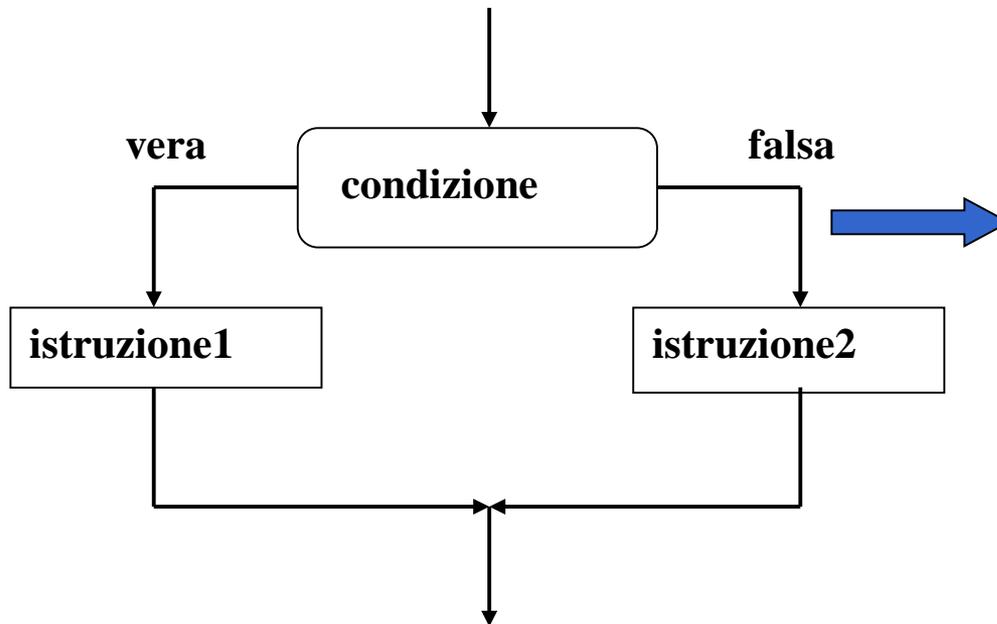
`<scelta> ::= if (<cond>) <istruzione1>
[else <istruzione2>]`



La condizione viene valutata al momento dell'esecuzione dell'if.

ISTRUZIONE DI SCELTA SEMPLICE

`<scelta> ::= if (<cond>) <istruzione1>
[else <istruzione2>]`



La parte **else** è *opzionale*:
se omessa, in caso di
condizione falsa si passa
subito all'istruzione che
segue l'**if**.

ESEMPIO di ISTRUZIONE IF

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

```
if (n > 0) {           /* inizio blocco */
    a = b + 5;
    c = a;
}                       /* fine blocco */
else n = b;
```

ESEMPIO di ISTRUZIONE IF

```
/* stampa il maggiore tra due numeri
   letti da utente */
#include <stdio.h>
int main()
{
    int primo,secondo;
    printf("inserisci due numeri");
    scanf("%d%d",&primo,&secondo);
    if (primo >secondo)
        printf("%d",primo);
    else printf("%d",secondo);
    return 0;
}
```

ISTRUZIONI IF ANNIDATE

- Come caso particolare, <istruzione1> o <istruzione2> potrebbero essere un altro if
- Occorre attenzione *ad associare le parti else (che sono opzionali) all' if corretto*

```
if (n > 0)
  if (a>b) n = a;
  else n = b; /* riferito a if(a>b) */
```

Regola semantica:
l'**else** è sempre associato
all'**if** più interno

```
if (n > 0)
  { if (a>b) n = a; }
else n = b; /* riferito a if(n>0) */
```

Se vogliamo cambiare questa
semantica, dobbiamo inserire un
blocco

ESEMPIO

Dati tre valori $a \leq b \leq c$ (*ipotesi semplificativa, che consente di avere 1 solo vincolo, mostrato sotto – anziché i tre vincoli usuali da verificare sui tre lati*) che rappresentano le lunghezze di tre segmenti, valutare se possono essere i tre lati di un triangolo, e se sì deciderne il tipo (scaleno, isoscele, equilatero).

Vincolo: deve essere $c < (a+b)$

- Rappresentazione delle informazioni:
 - la variabile booleana `triangolo` indica se i tre segmenti possono costituire un triangolo
 - le variabili booleane `scaleno`, `isoscele` e `equil` indicano il tipo di triangolo.

ESEMPIO

Specifica:

se $a+b>c$

triangolo = vero

se $a=b=c$ { equil=isoscele=vero
scaleno=falso }

altrimenti

se $a=b$ o $b=c$ o $a=c$ { isoscele=vero;
equil=scaleno=falso }

altrimenti

{ scaleno=vero;
equil=isoscele=falso }

altrimenti

triangolo = falso

ESEMPIO

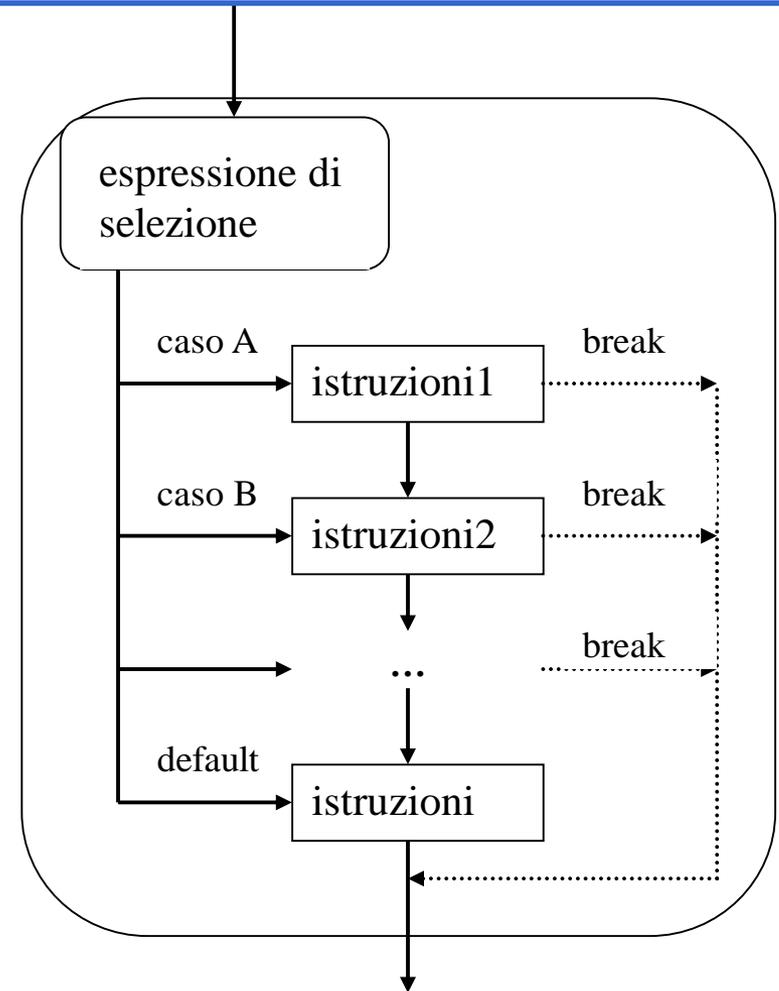
*I lati a,b,c dovrebbero essere letti
in input dall'utente.*

Modificare il programma in tal senso

```
int main () {
    float a=1.5, b=3.0, c=4.0;
    int triangolo, scaleno, isoscele, equil;
    triangolo = (a+b>c);
    if (triangolo) {
        if (a==b && b==c)
            { equil=isoscele=1; scaleno=0; }
        else if (a==b || b==c || a==c)
            { isoscele=1; scaleno=equil=0;}
        else
            { scaleno=1; isoscele=equil=0;}
    }
    return 0;
}
```

ISTRUZIONE DI SCELTA MULTIPLA

- Consente di scegliere fra *molte istruzioni* (*alternative o meno*) in base al valore di una *espressione di selezione*.
- L'espressione di selezione deve *denotare un valore numerabile* (intero, carattere,...).



ISTRUZIONE DI SCELTA MULTIPLA

```
<scelta-multipa> ::=  
  switch (selettore) {  
    case <etichetta1> : <istruzioni> [break;]  
    case <etichetta2> : <istruzioni> [break;]  
    ...  
    [ default : < istruzioni> ]  
  }
```

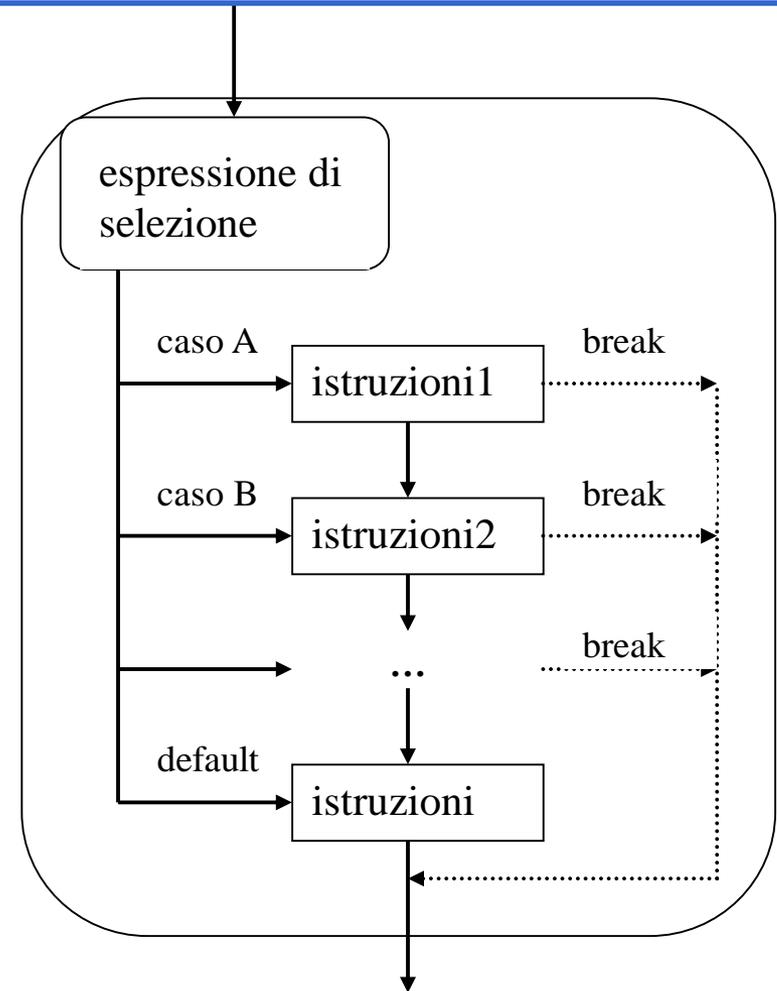
Sequenze, non occorre il blocco

Se nessuna etichetta corrisponde, si prosegue con il ramo **default** se esiste, altrimenti non si fa niente

- Il valore dell'espressione *selettore* viene confrontato con le etichette (costanti dello stesso tipo del selettore) dei vari casi: *l'esecuzione prosegue dal ramo corrispondente (se esiste)*.

NOTA

I vari rami non sono mutuamente esclusivi:
imboccato un ramo, si eseguono anche tutti i rami successivi a meno che non ci sia il comando **break** a forzare esplicitamente l'uscita.



ISTRUZIONE DI SCELTA MULTIPLA

```
switch (mese)
{
case 1 : giorni = 31; break;
case 2: if (bisestile) giorni = 29;
        else giorni = 28;
        break;
case 3:  giorni = 31;  break;
case 4:  giorni = 30;  break;
...
case 12: giorni = 31;
}
```

ISTRUZIONE DI SCELTA MULTIPLA

- Alternativa

```
switch (mese)
{
case 2:
    if (bisestile) giorni = 29;
    else giorni = 28;
    break;
case 4:   giorni = 30;   break;
Case 6:   giorni = 30;   break;
case 9:   giorni = 30;   break;
case 11:  giorni = 30;   break;
default:  giorni = 31;
}
```

ISTRUZIONE DI ITERAZIONE

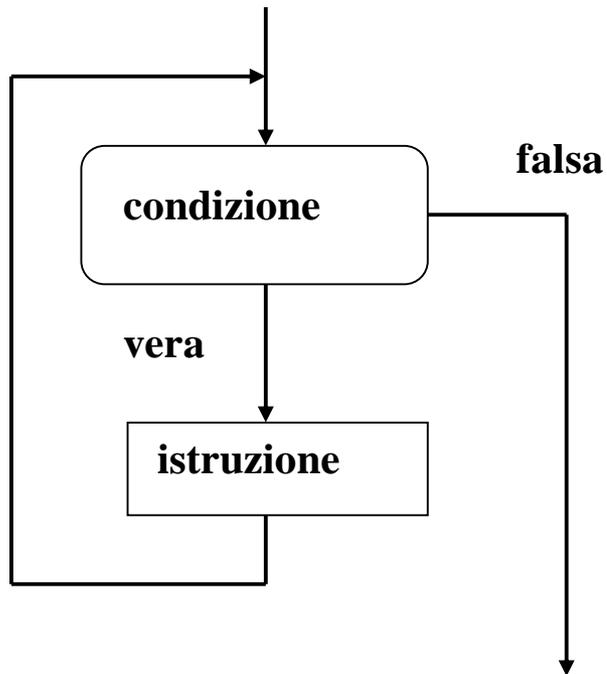
`<iterazione> ::=`
`<while> | <for> | <do-while>`

- Le istruzioni di iterazione:
 - hanno *un solo punto di ingresso* e *un solo punto di uscita* nel flusso del programma
 - perciò possono essere interpretate *come una singola azione* in una computazione sequenziale.

ISTRUZIONE `while`

`<while> ::=`

`while (<condizione>) <istruzione>`

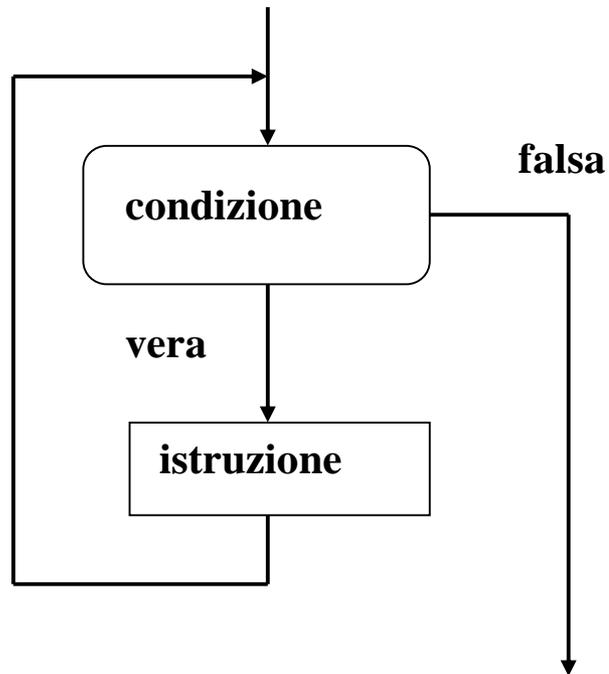


- L'istruzione viene ripetuta *per tutto il tempo in cui la condizione rimane vera*.
- Se la condizione è falsa, l'iterazione non viene eseguita *neppure una volta*.
- In generale, *non è noto **quante volte*** l'istruzione sarà ripetuta.

ISTRUZIONE while

`<while> ::=`

`while (<condizione>) <istruzione>`



Prima o poi, *direttamente o indirettamente*, l'istruzione deve modificare la condizione: altrimenti, l'iterazione durerà *per sempre!*
CICLO INFINITO



Perciò, quasi sempre *istruzione* è *un blocco*, al cui interno si *modifica qualche variabile che compare nella condizione*.

ESEMPIO ISTRUZIONE DI CICLO

```
#include <stdio.h>
int main() /* Media di n voti*/
{ int    voto,N,i;
  float  media, sum;

  printf("Quanti sono i voti ?");
  scanf("%d",&N);
  sum = 0;
  i = 1;
  while (i <= N)
  { printf("Dammi il voto n.%d:",i);
    scanf("%d",&voto);
    sum=sum+voto;
    i=i+1;
  }
  media=sum/N;
  printf("Risultato: %f",media);
  return 0;
}
```

ESEMPIO ISTRUZIONE DI CICLO

```
/* moltiplicazione come sequenza di somme */
#include <stdio.h>
int main()
{
    int  X,Y,Z;

    printf("Dammi i fattori:");
    scanf("%d%d", &X, &Y);
    Z=0;
    while (X!=0)
        { /* corpo ciclo while */
            Z=Z+Y;
            X=X-1;
        }
    printf("%d", Z);
    return 0;
}
```

ESEMPIO ISTRUZIONE DI CICLO

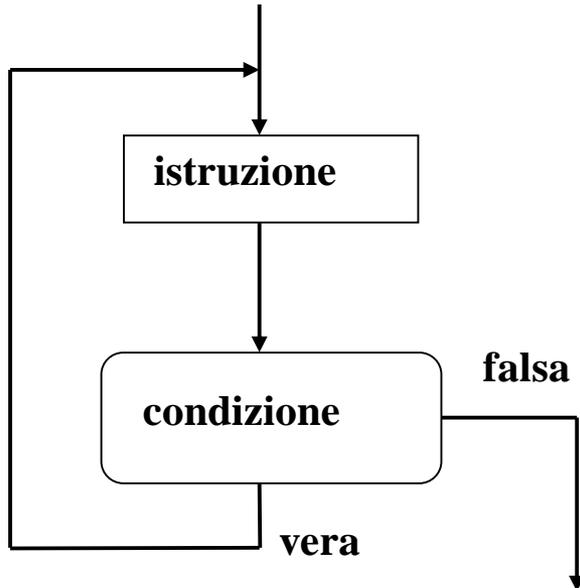
```
/* Calcolo del fattoriale di un numero N */

#include <stdio.h>
int main()
{
    int F, N, I;
    F=1; /* inizializzazione del fattoriale*/
    I=1; /* inizializzazione del contatore*/
    printf("Dammi N:");
    scanf("%d", &N);

    while (I <= N)
    {
        F = I*F;
        I = I+1;
    }
    printf("Il fattoriale e' %d", F);
    return 0;
}
```

ISTRUZIONE `do..while`

`<do-while> ::=`
`do <istruzione> while (<condizione>);`



È una variante della precedente:
la condizione viene verificata dopo aver eseguito l'istruzione.

Se la condizione è falsa,
l'iterazione **viene comunque**
eseguita almeno una volta.

ESEMPIO ISTRUZIONE DI CICLO

```
/* Calcolo del fattoriale di un numero N */

#include <stdio.h>
int main()
{
    int F, N, I;
    F=1; /* inizializzazione del fattoriale*/
    I=1; /* inizializzazione del contatore*/
    printf("Dammi N:");
    scanf("%d", &N);
    do
    {
        F = I*F;
        I = I+1;
    }
    while (I <= N);
    printf("Il fattoriale e' %d", F);
    return 0;
}
```

ESEMPIO

- Nell'istruzione **while**, la condizione di ripetizione viene verificata **all'inizio di ogni ciclo**

```
...
somma=0; j=1;
while (j <= n)
    { somma = somma + j;  j++; }
```

- Nell'istruzione **do** la condizione di ripetizione viene verificata **alla fine di ogni ciclo**

```
/* In questo caso: n > 0 */
somma = 0; j = 1;
do
    { somma = somma + j;  j++; }
while (j <= n);
```

ISTRUZIONE `for`

- È una evoluzione dell'istruzione `while` che mira a eliminare alcune frequenti sorgenti di errore:
 - mancanza delle *inizializzazioni delle variabili*
 - mancanza della *fase di modifica del ciclo* (rischio di ciclo senza fine)
- In genere si usa quando è noto il numero di volte in cui dovrà essere eseguito il ciclo.

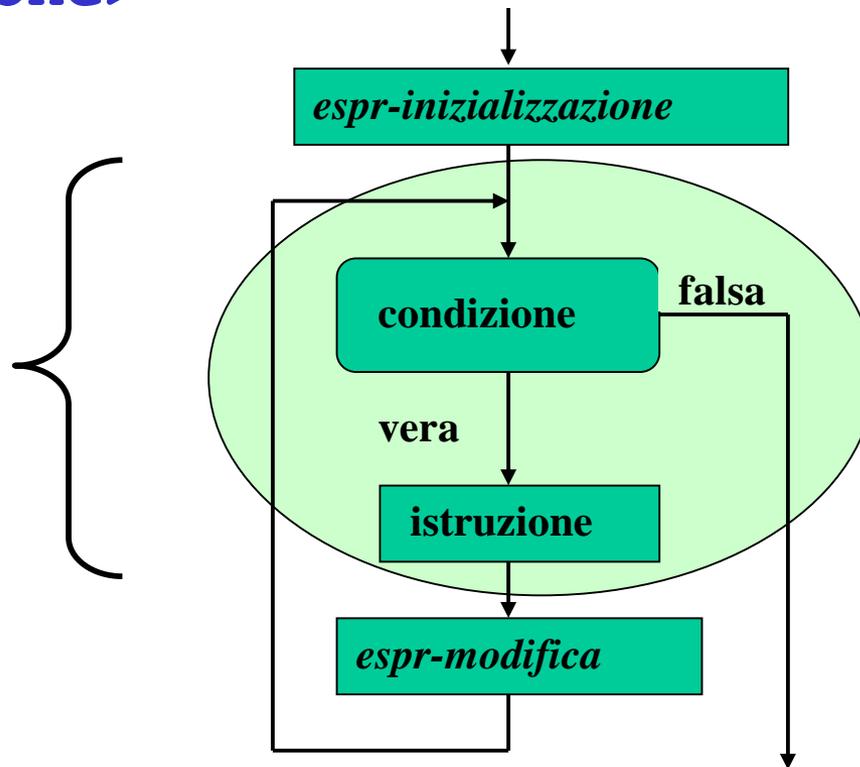
ISTRUZIONE for

`<for> ::=`

`for (<espr-iniz>; <cond>; <espr-modifica>)`

`<istruzione>`

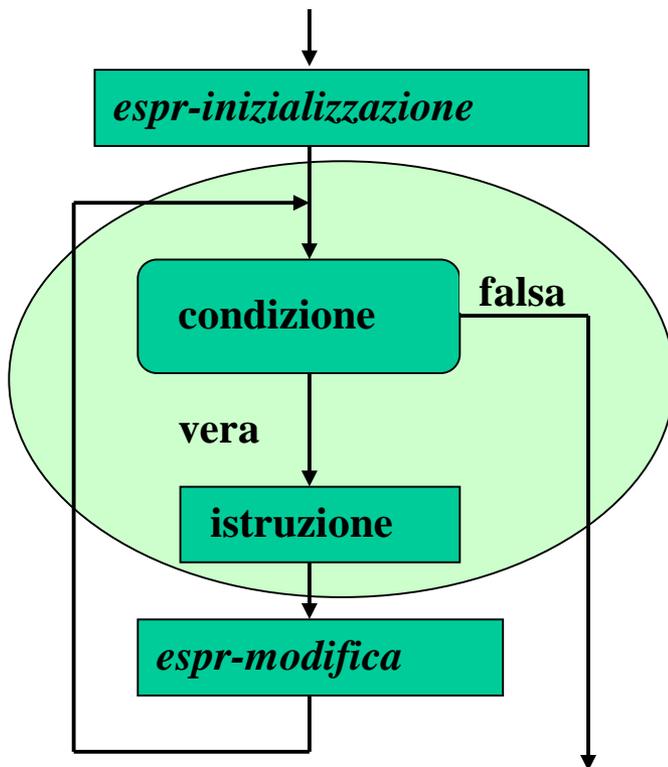
*Struttura
del while*



ISTRUZIONE for

`<for> ::=`

```
for (<espr-iniz>; <cond>; <espr-modifica>)  
<istruzione>
```



Espressione di inizializzazione:

`<espr-iniz>`

valutata *una e una sola volta* prima di iniziare l'iterazione.

Condizione: `<cond>`

valutata *a ogni iterazione*, per decidere se proseguire (come in un while). Se manca si assume *vera!*

Espressione di modifica: `<espr-modifica>`

valutata *a ogni iterazione*, dopo aver eseguito l'istruzione.

ESEMPIO ISTRUZIONE DI CICLO

```
#include <stdio.h>
int main() /* Media di n voti*/
{ int    voto,N,i;
  float  media, sum;

  printf("Quanti sono i voti ?");
  scanf("%d", &N);
  sum = 0;
  for(i = 1; i <= N;i++)
  { printf("Dammi il voto n.%d:",i);
    scanf("%d", &voto);
    sum=sum+voto;
  }
  media=sum/N;
  printf("Risultato: %f",media);
  return 0;
}
```

Nota: non serve l'inizializzazione del contatore i e l'incremento di i nel ciclo

RIPRENDIAMO IL CASO DEL WHILE

```
#include <stdio.h>
int main() /* Media di n voti*/
{ int      voto,N,i;
  float    media, sum;

  printf("Quanti sono i voti ?");
  scanf("%d", &N);
  sum = 0;
  i = 1;
  while (i <= N)
  { printf("Dammi il voto n.%d:",i);
    scanf("%d", &voto);
    sum=sum+voto;
    i=i+1;
  }
  media=sum/N;
  printf("Risultato: %f",media);
  return 0; }
```

ESEMPIO ISTRUZIONE DI CICLO

```
/* Calcolo del fattoriale di un numero N */
#include <stdio.h>

int main()
{
    int    N, F, i;

    printf("Dammi N:");
    scanf("%d", &N);
    F=1;  /*inizializzazione del fattoriale*/
    for (i = 1; i <= N ; i++)
        F=F*i;

    printf("%s%d", "Fattoriale: ", F);
    return 0;
}
```

RIPRENDIAMO IL CASO DEL WHILE

```
/* Calcolo del fattoriale di un numero N */

#include <stdio.h>
int main()
{
    int F, N, I;
    F=1; /* inizializzazione del fattoriale*/
    I=1; /* inizializzazione del contatore*/
    printf("Dammi N:");
    scanf("%d", &N);

    while (I <= N)
    {
        F = I*F;
        I = I+1;
    }
    printf("Il fattoriale e' %d", F);
    return 0;
}
```

BREAK E CONTINUE

Vediamo ora due istruzioni particolari:

- **break**

Interrompe il ciclo corrente

- **continue**

Interrompe l'iterazione corrente

Sono da usare con cautela!

- “rovinano” la programmazione strutturata
- Ma sono così utili che continuano ad essere usate