

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

- 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 come un comando (complesso) ottenuto tramite **istruzioni elementari**, mediante alcune regole di composizione (**strutture di controllo**)

STRUTTURE DI CONTROLLO

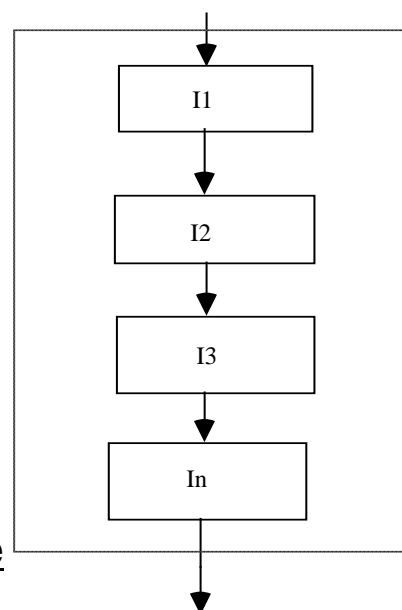
Concetti chiave:

- 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*”)
- **Bohm e Jacopini (1966): sono sufficienti per definire tutte le funzioni computabili**

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 stampi la somma*/
#include <stdio.h>
main()
{ /* INIZIO BLOCCO */
int X,Y;
    printf("Inserisci due numeri ");
    scanf("%d%d",&X,&Y);
    printf("%d",X+Y);
} /* FINE BLOCCO */
```

REGOLE DI VISIBILITÀ

Esistono delle **regole di visibilità** per gli identificatori (nomi di variabili, di funzioni, costanti) che definiscono in **quali parti** del programma tali identificatori possono essere usati

In un programma esistono diversi **ambienti**:

- area globale
- **il main**
- ogni singola funzione
- **ogni blocco**

REGOLE DI VISIBILITÀ

- Un identificatore **non** è visibile **prima** della sua dichiarazione
- Un identificatore definito in un ambiente è visibile in **tutti gli ambienti in esso contenuti**
- Se in un ambiente sono visibili **due definizioni** dello stesso identificatore, la definizione valida è quella dell'ambiente **più vicino** al punto di utilizzo
- In ambienti diversi si può definire lo stesso identificatore per denotare due oggetti diversi
- In ciascun ambiente un identificatore può essere definito una sola volta

REGOLE DI VISIBILITÀ

- Un identificatore non è visibile prima della sua dichiarazione

SCORRETTO

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

CORRETTO

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

REGOLE DI VISIBILITÀ

- *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 oggetti diversi*

```
main(){  
    float x = 3.5;  
    {int y, x = 5;  
      y = x; /* y vale 5 */}  
    ...}
```

REGOLE DI VISIBILITÀ

- *In ciascun ambiente un identificatore può essere dichiarato una sola volta*

```
main(){  
    float x = 3.5;  
    char x; → SCORRETTO  
    ...}
```

REGOLE DI VISIBILITÀ

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

SCORRETTO

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

CORRETTO

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

ESEMPIO di BLOCCO

```
#include <stdio.h>
main()
{ /* INIZIO BLOCCO1 */
  int X;
  printf("Inserisci il numero X");
  scanf("%d",&X);
  { /* INIZIO BLOCCO2 */
    int Y;
    printf("Inserisci il numero Y");
    scanf("%d",&Y);
    printf("%d",X+Y);} /* FINE BLOCCO2 */
} /* FINE BLOCCO1 */
```

ESEMPIO di BLOCCO

```
#include <stdio.h>
main()
{ /* INIZIO BLOCCO1 */
int X;
    printf("Inserisci il numero X");
    scanf("%d",&X);
    { /* INIZIO BLOCCO2 */
    int Y;
    printf("Inserisci il numero Y");
    scanf("%d",&Y);} /* FINE BLOCCO2 */
    printf("%d",X+Y);} /* FINE BLOCCO1 */
```

NO: Le regole di visibilità non sono rispettate

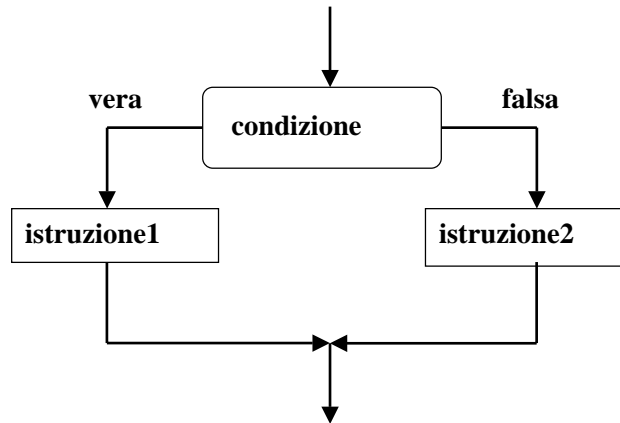
ISTRUZIONI CONDIZIONALI

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

- la seconda *non è essenziale*, ma migliora l'espressività
- l'espressione condizionale ternaria (`.. ? ... : ...`) fornisce *già* 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

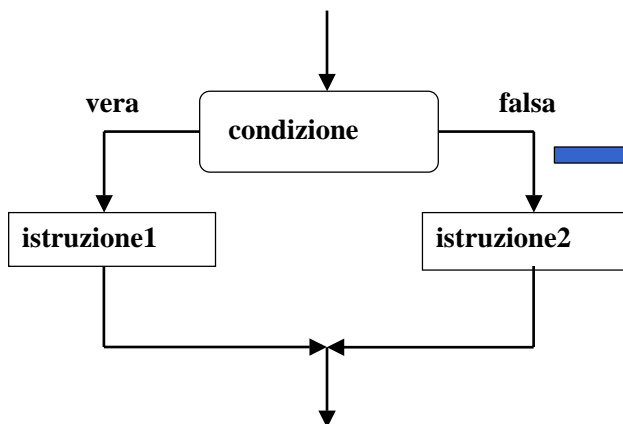
```
<scelta> ::= if (<cond>) <istruzione1>  
           [ else <istruzione2> ]
```



La condizione viene valutata al momento dell'esecuzione di 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 *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

```
/* determina il maggiore tra due numeri */

#include <stdio.h>
main()
{
    int primo,secondo;

    scanf("%d%d",&primo,&secondo);
    if (primo > secondo)
        printf("%d",primo);
    else printf("%d",secondo);
}
```

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) */
```

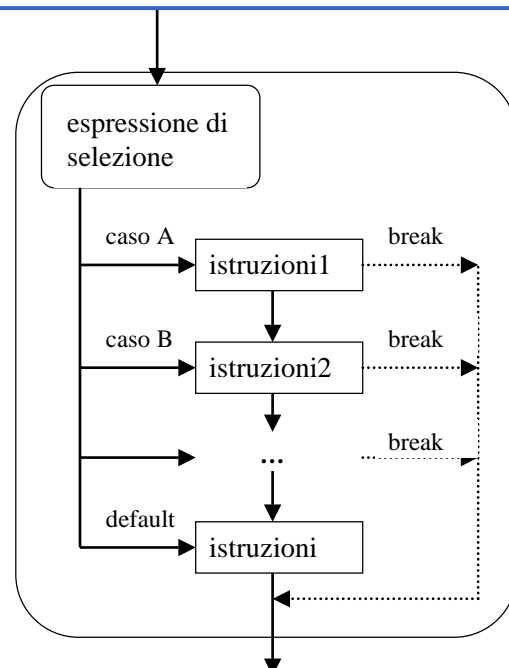
Regola semantica:
else è sempre associato
a if più interno

```
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

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

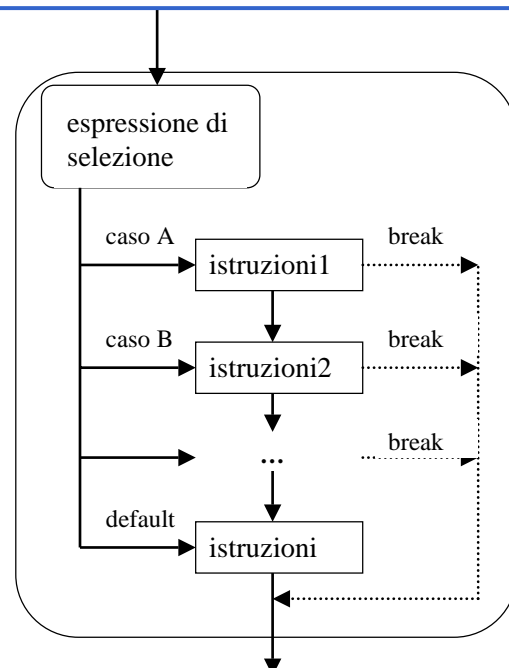
Sequenze, non occorre il blocco

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): *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 possibile:

```
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

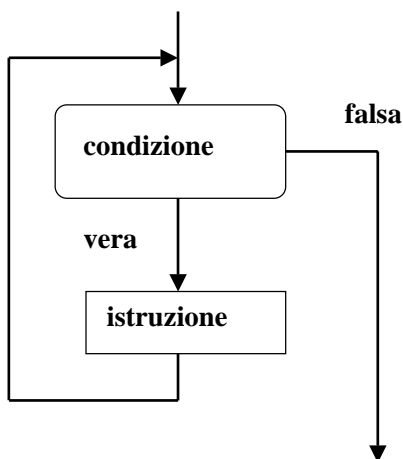
```
<iterazione> ::=  
    <while> | <for> | <do-while>
```

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`

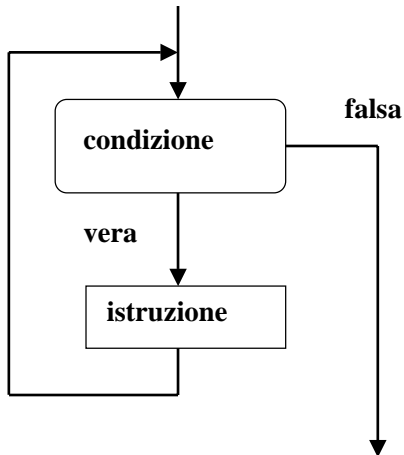
```
<while> ::=  
    while(<condizione>) <istruzione>
```



- 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, **CICLO INFINITO**



Quasi sempre *istruzione* è un blocco, in cui si modifica qualche variabile che compare nella condizione

ESEMPIO ISTRUZIONE DI CICLO

```
#include <stdio.h>
main() /* Media di N voti*/
{ int    sum,voto,N,i;
  float  media;
  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=(float)sum/N; /* ipotesi: N>0 */
  printf("Risultato: %f",media);}
```

ESEMPIO ISTRUZIONE DI CICLO

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

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

ESEMPIO ISTRUZIONE DI CICLO

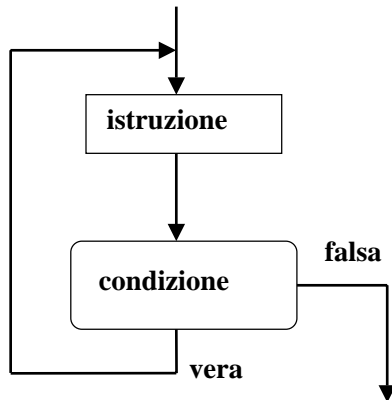
```
/* Calcolo del fattoriale di un numero N */

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

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


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>  
main()  
{ int F, N, I;  
  F=1; /* inizializzazione del fattoriale*/  
  I=1; /* inizializzazione del fattoriale*/  
  printf("Dammi N:");  
  scanf("%d",&N);  
  do  
  {F = I*F;  
   I = I+1;  
  }  
  while (I <= N)  
  printf("Il fattoriale è %d", F);  
}
```

ESEMPIO

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

```
...
somma=0; j=1; n=10;
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; n=10;
do
    { somma = somma + j; j++; }
while (j <= n);
```

Cosa succede se n==0???

ESERCIZIO

- Leggi da input caratteri fino al carattere punto (valore sentinella)

```
...
do
scanf("%c", &ch);
while (ch != '.')
```

Oppure:

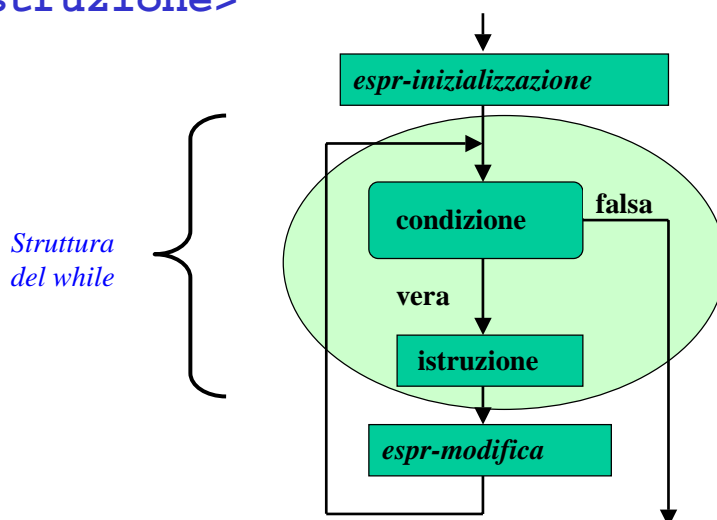
```
ch = '*'
while (ch != '.')
scanf("%c", &ch);
```

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 quante volte il ciclo dovrà essere eseguito (contatore)

ISTRUZIONE for

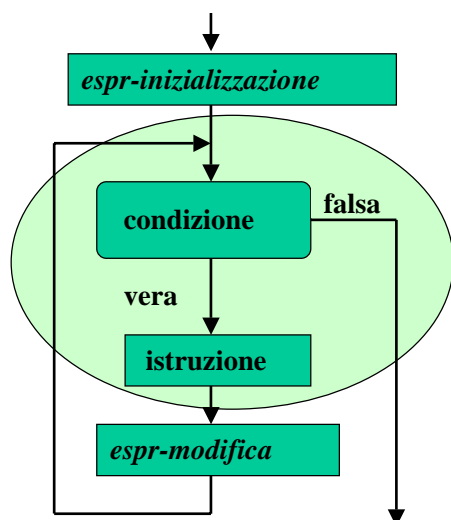
```
<for> ::=  
for (<espr-iniz>; <cond>; <espr-modifica> )  
<istruzione>
```



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

ISTRUZIONE for: equivalenza con while

```
for (e1; e2; e3)  
  <istruzione>
```

Equivalente a :

```
e1;  
while (e2)  
  {<istruzione>  
  e3;}
```

ESEMPIO ISTRUZIONE DI CICLO


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

  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=(float)sum/N;
  printf("Risultato: %f",media);
}
```

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

ESEMPIO ISTRUZIONE DI CICLO

```
/* Calcolo del fattoriale di un numero N */
#include <stdio.h>
#include <math.h>
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("Fattoriale: %d", F);
}
```

ESEMPIO

Dati tre valori $a \leq b \leq c$ 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

```
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;}
    }
}
```

ESEMPIO

- Dati due valori positivi X e Y , calcolarne la divisione intera X/Y come sequenza di sottrazioni, ottenendo quoziente e resto

Invariante di ciclo:

$$X = Q * Y + R, \text{ con } R \geq 0$$

- inizialmente, $Q=0, \quad R=X \quad (R>Y)$
- a ogni passo, $Q'=Q+1, \quad R'=R-Y \quad (R>Y)$
- alla fine, $X = Q^{(n)} * Y + R^{(n)} \quad (0<R<Y)$
che è la definizione di divisione intera

ESEMPIO

Specifica:

sia Q il quoziente, inizialmente pari a 0

sia R il resto, inizialmente pari a X

while ($R \geq Y$)

 incrementare il quoziente Q

 decrementare R di una quantità Y

Codifica

```
main() {  
    int x = 20, y = 3, q, r;  
    for (q=0, r=x; r>=y; q++, r=r-y);  
}
```

Idem per l'espressione di modifica

Notare l'uso di una espressione concatenata per concatenare due assegnamenti e inizializzare così due variabili

CHE COSA SUCCEDERE?

```
a=0;  
printf("ciao e %d", a=1);
```

```
a=0;  
printf("ciao e %d", a==1);
```

```
a=0;  
printf("ciao e %d", a==0);
```

```
a=0;  
printf("ciao e a=%d", a);
```

```
a=1;  
if (a=4)  
    printf("ciao e a=%d", a);
```