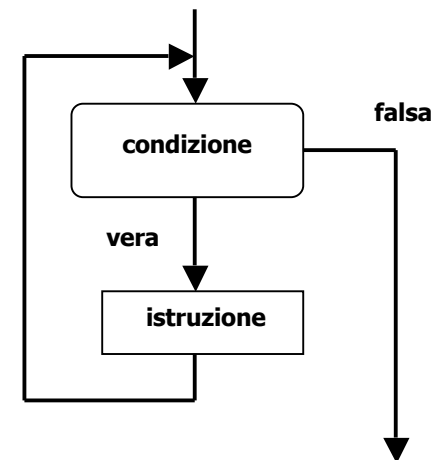
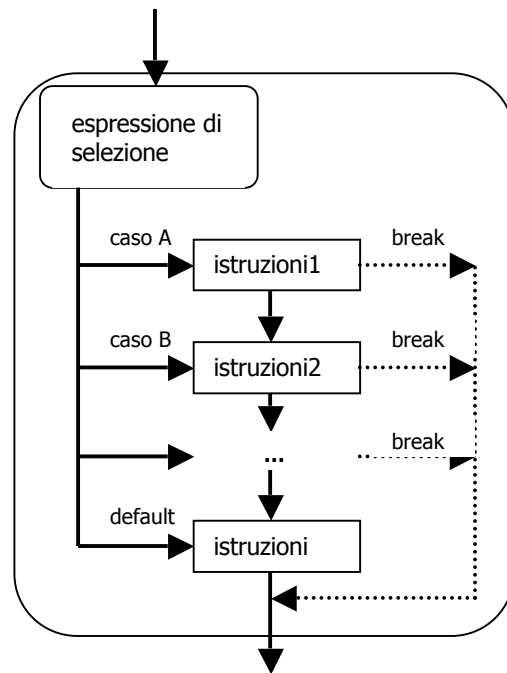


Il linguaggio C

La programmazione strutturata



Programmazione strutturata (Dijkstra, 1969)

La programmazione strutturata nasce come proposta per regolamentare e standardizzare le metodologie di programmazione.

Obiettivo:

rendere piu' facile la lettura dei programmi (e quindi la loro modifica e manutenzione).

Idea di base:

Ogni programma viene visto come un comando (complesso, o *strutturato*) ottenuto componendo altri comandi mediante alcune regole di composizione (***strutture di controllo***).

Strutture di controllo:

- **concatenazione** (o composizione, blocco);
- **alternativa** (o istruzione condizionale)
- **ripetizione** (o iterazione)

Si puo` dimostrare (*teorema Böhm-Jacopini*) che queste strutture sono sufficienti per esprimere qualunque algoritmo.

Programmazione Strutturata

Ogni programma viene visto come un comando (complesso, o *strutturato*) ottenuto componendo altri comandi **soltanto mediante le 3 strutture di controllo**: concatenazione, alternativa e ripetizione.

NB: Altre strutture non possono essere utilizzate
→ abolizione dei *salti incondizionati* (**goto**) nel flusso di controllo.

Programmazione Strutturata

Vantaggi:

- leggibilità dei programmi
- supporto a metodologia di progetto **top-down**: soluzione di problemi complessi attraverso scomposizione in sotto-problemi, a loro volta scomponibili in sotto problemi, etc. La soluzione si ottiene componendo le soluzioni dei sottoproblemi attraverso **concatenazione, alternativa e ripetizione**.
- supporto a metodologia **bottom-up**: la soluzione di problemi avviene aggregando componenti già disponibili mediante **concatenazione, alternativa e ripetizione** (programmazione **per componenti**).
- facilità di **verifica e manutenzione**

Teorema di Böhm e Jacopini

Th: le strutture di **concatenazione, alternativa e ripetizione** costituiscono un **insieme completo** in grado di esprimere tutte le funzioni calcolabili.

Ne consegue che:

1. Un linguaggio composto dalle istruzioni:
 - lettura, scrittura (scanf, printf) } istruzioni semplici
 - assegnamento }
 - istruzione di concatenazione (o composta) : **{...}**
 - istruzione alternativa (o condizionale): **if...else...**
 - istruzione di ripetizione (o iterazione): **while...**
- e' **un linguaggio completo** (cioe', in grado di esprimere tutte le funzioni calcolabili).
2. L'uso di queste sole strutture di controllo non limita il potere espressivo.

Istruzioni strutturate in C

Il C e` un linguaggio strutturato.

3 Categorie di istruzioni strutturate:

- istruzione **composta**: (o blocco) { }
- istruzioni **alternative**: `if`, `switch`
- istruzioni **ripetitive (o iterative)**: `while`, `do`, `for`

NB: ogni istruzione strutturata e` sintatticamente equivalente a una singola istruzione (strutturata).

Istruzione composta { }

L'istruzione composta (o blocco) serve per aggregare una sequenza di istruzioni in un'unica istruzione strutturata.

Sintassi:

```
{  
  <Dichiarazioni e Definizioni>  
  <Sequenza di Istruzioni>  
}
```

<sequenza-istruzioni> ::= <istruzione> {; <istruzione> }

Effetto:

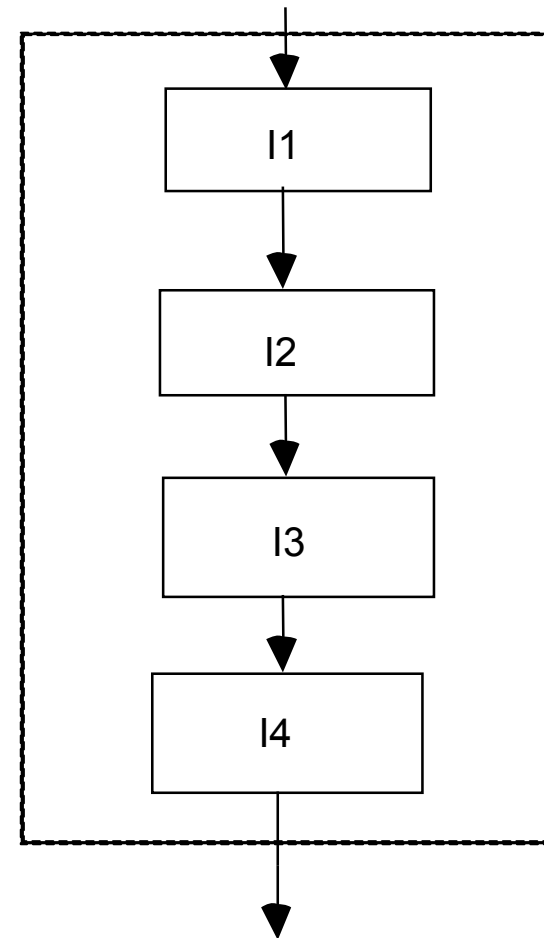
esecuzione delle istruzioni componenti nell'ordine testuale.

Istruzione Composta

Esempio:

```
{ I1;  
  I2  
  I3;  
  I4;  
}
```

e' equivalente a:



Istruzione Composta

Dichiarazioni e definizioni:

E' possibile inserire in un blocco definizioni di variabili; per tali variabili il campo di azione e' individuato dal blocco in cui sono definite (cioe', hanno **visibilita'** limitata al blocco).

Ad esempio:

```
main()
{
  int X;
  ...
  {
    int A; /* def. locale al blocco */
    A=100;
    printf("Valore di A: %d\n", A);
  }
  A=X; /* errore!! qui A non e` visibile*/
  ...
}
```

Istruzioni di alternativa: `if`

Sintassi:

```
if (<selettore>
    <istruzione1>
    [else
    <istruzione2>]
```

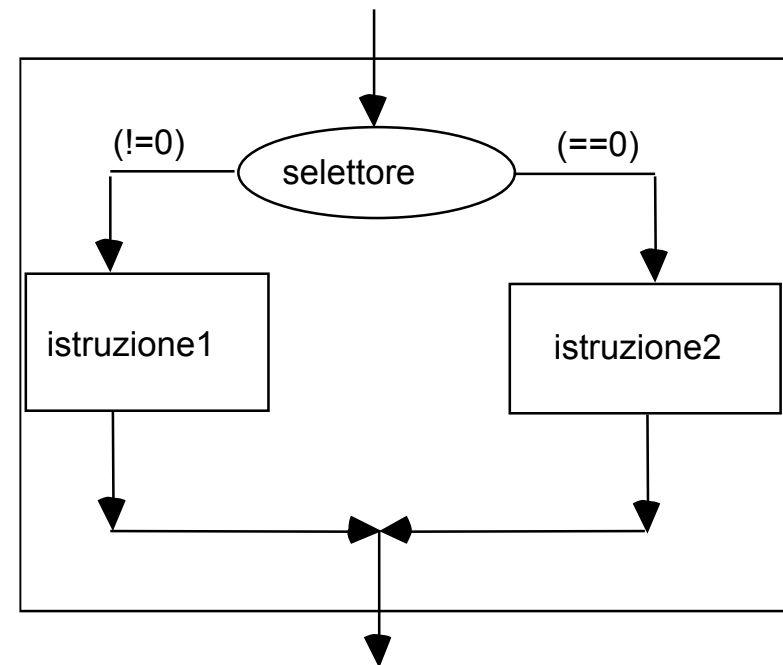
Sintassi:

- viene selezionata una e una sola tra le istruzioni componenti (<istruzione1> e <istruzione2>) in base al risultato di una condizione (<selettore>):
 - se il valore di <selettore> è vero (cioè, diverso da zero) viene eseguita <istruzione1>,
 - altrimenti (se il valore di <selettore> è falso) viene eseguita <istruzione2>.

Istruzioni di alternativa: `if`

```
if (<selettore>
    <istruzione1>
else
    <istruzione2>
```

e' equivalente a



Istruzione if: esempio

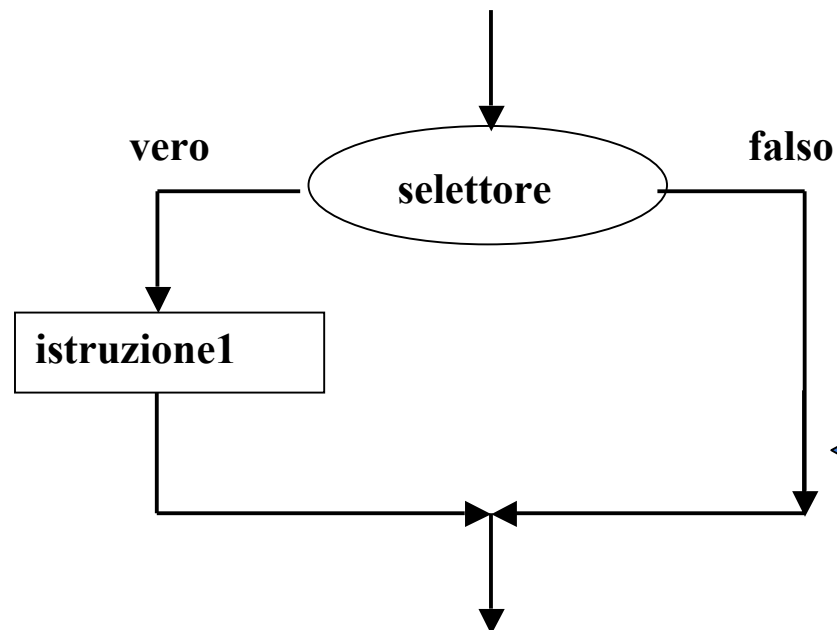
```
#include <stdio.h>
main()
{ int A, B;
  scanf("%d%d", &A, &B);
  if (B==0)
    printf ("B vale zero!\n");
  else
    printf ("Quoziente: %d\n", A/B);
  printf ("Fine!\n");
}
```

Indentazione:

- il rientro (*indent*) delle linee del testo del programma descrive l'annidamento delle varie istruzioni → si aumenta la leggibilità del programma (modifica più facile).

ISTRUZIONE if

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



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

```
if (<selettore>  
    <istruzione1>;
```

Istruzione if: esempio

La parte else dell'istruzione if è opzionale:

```
#include <stdio.h>
main()
{ int A, B;
  scanf("%d%d", &A, &B);
  if (B!=0)
      printf("Quoziente:%d\n", A/B);
  printf("Fine!\n");
}
```

Istruzione if

- <istruzione1> e <istruzione2> sono ciascuna una **singola istruzione**
- Se in una alternativa occorre 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

```
/* determina il maggiore tra due numeri */  
  
#include <stdio.h>  
void 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 :

```
if (n > 0)
  if (a>b) n = a; /* if "annidato"*/
  else n = b; /* a quale if e` riferito? */
```

→ Attenzione ad associare le parti else (che sono opzionali) all' if corretto:

Regola :

l'**else** è sempre associato all'**if** più "interno"

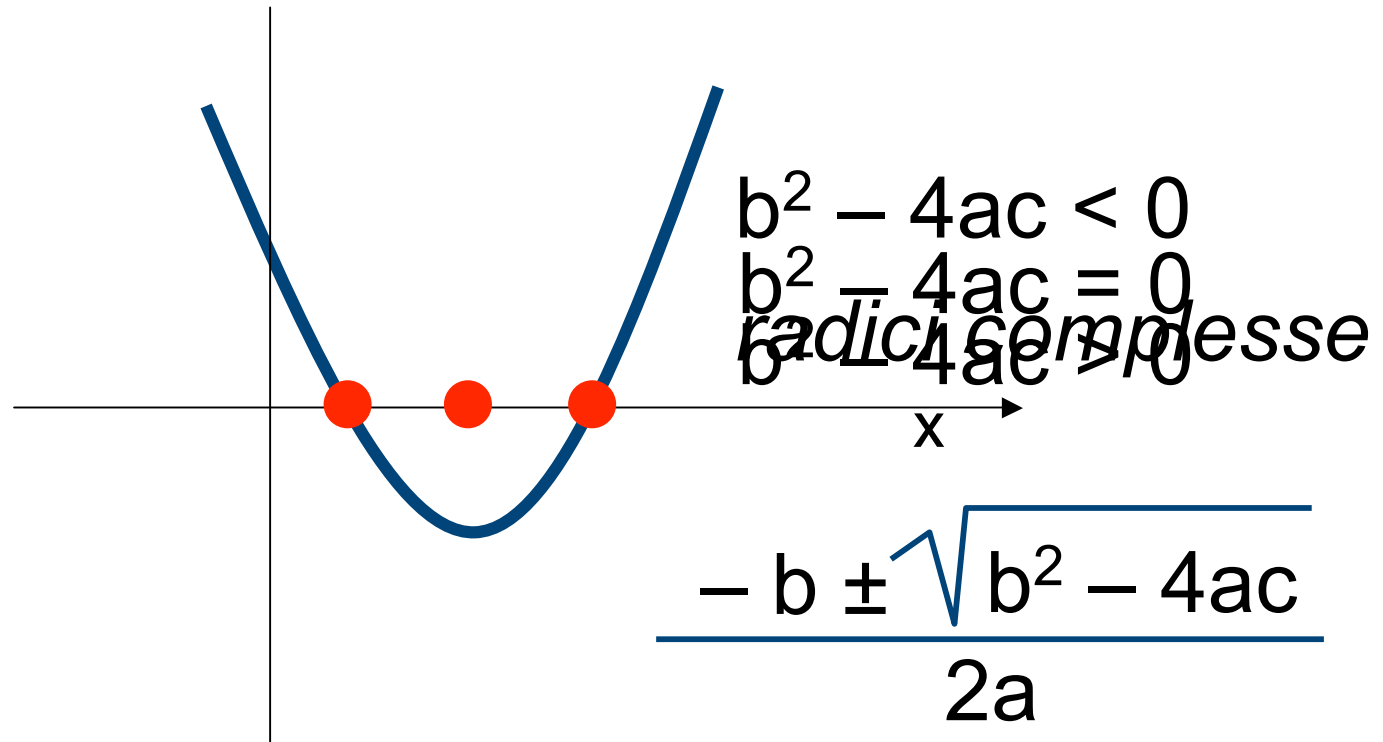
```
if (n > 0)
  if (a>b) n = a;
  else n = b; /* e` riferito a if (a>b) */
```

→ Per associare l'else all'if piu` esterno dobbiamo usare il **blocco:**

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

Equazioni di secondo grado...

$$ax^2 + bx + c = 0$$



Esempio (i f): equazione di secondo grado

```
/* Programma che calcola le radici di un'equazione di
   secondo grado:  $ax^2+bx+c=0$  */
#include <stdio.h>
#include <math.h> /*libreria standard matematica*/
main()
{
    float a,b,c; /*coefficienti e termine noto*/
    float d,x1,x2;

    scanf("%f%f%f",&a,&b,&c);
    if ((b*b) < (4*a*c))
        printf("radici complesse");
    else
    {
        d=sqrt(b*b-4*a*c); /*sqrt: funzione di libreria */
        x1=(-b+d)/(2*a);
        x2=(-b-d)/(2*a);
        printf("\nRadici reali: %f\t%f",x1,x2);
    }
}
```

ESERCIZIO

Dati tre valori a , b , 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: la somma di ogni coppia di lati deve essere maggiore del terzo lato.

Rappresentazione delle informazioni:

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

ESERCIZIO

Specifica:

se $a+b>c$ and $a+c>b$ and $b+c>a$

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

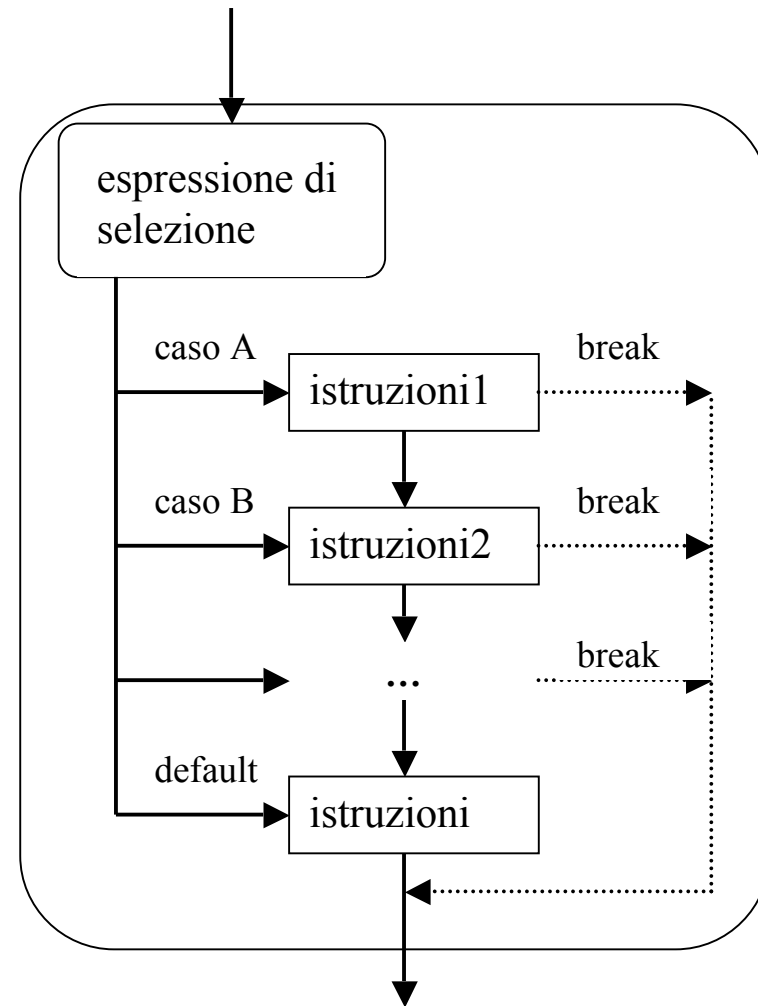
{ scaleno=vero;
equil=isoscele=falso }

altrimenti

triangolo = falso

Istruzione di scelta multipla: `switch`

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



Istruzione di scelta multipla: `switch`

Sintassi:

```
switch (<selettore>
{
    case <costante1>: <blocco1>; [break;]
    case <costante2>: <blocco2>; [break;]
    ...
    case <costanteN>: <bloccoN>; [break;]
    [default: <bloccoDiDefault>;]
}
```

Dove:

- **<selettore>** e` un'espressione di tipo enumerabile (intero o char)
- **<costante1>**, **<costante2>** ecc. sono costanti dello stesso tipo del selettore (**etichette**)
- **<blocco1>**, **<blocco2>** , ecc. sono sequenze di istruzioni .

Istruzione di scelta multipla: `switch`

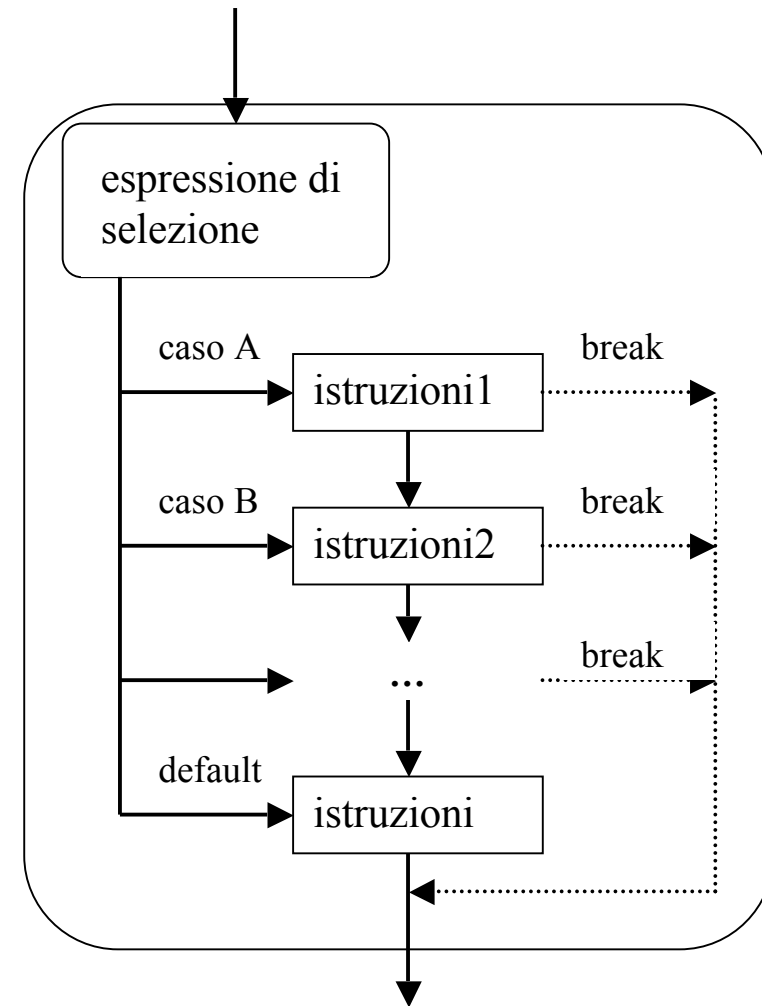
```
switch (<selettore>
{
    case <costante1>: <blocco1>; [break;]
    case <costante2>: <blocco2>; [break;]
    ...
    case <costanteN>: <bloccoN>; [break;]
    [default: <bloccoDiDefault>;]
}
```

Significato:

- Il valore dell'espressione <selettore> viene confrontato con ogni etichetta (<costante1>, <costante2>, ecc.) nell'ordine testuale.
- Se l'espressione <selettore> restituisce un valore uguale ad una delle costanti indicate (per esempio <costantei>), si esegue il <blocco1> e tutti i blocchi dei rami successivi.
- I blocchi **non sono** mutuamente esclusivi: possibilità di eseguire in sequenza più blocchi di istruzioni.

switch

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.



Switch: esempio

Calcolo della durata di un mese: 12 rami mutuamente esclusivi

```
int mese;  
...  
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 5:  giorni =31;    break;  
  case 6:  giorni = 30;   break;  
  case 7:  giorni =31;    break;  
  case 8:  giorni =31;    break;  
  case 9:  giorni =30;    break;  
  case 10: giorni =31;    break;  
  case 11: giorni =31;    break;  
  case 12: giorni = 31;  
}
```

Esempio switch

Oppure, soluzione piu` sintetica mediante il ramo **default**:

```
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;
}
```

Esempio switch

Oppure:

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

Istruzioni 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 istruzione*.

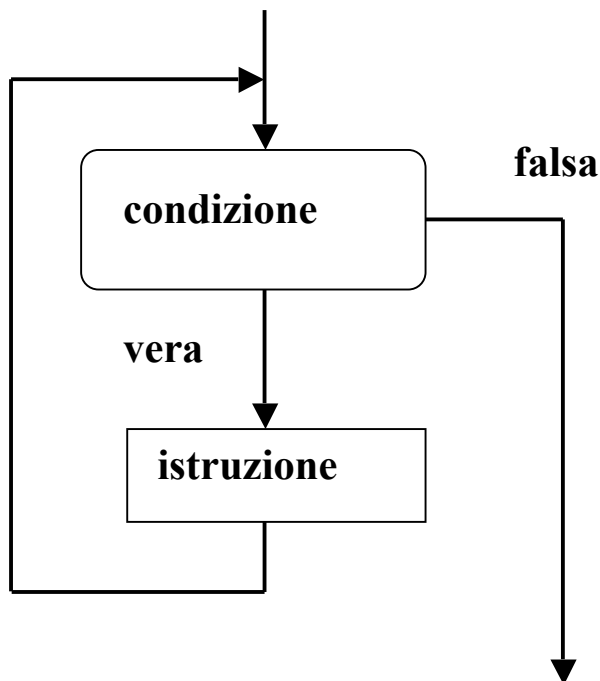
Istruzione `while`

Sintassi:

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

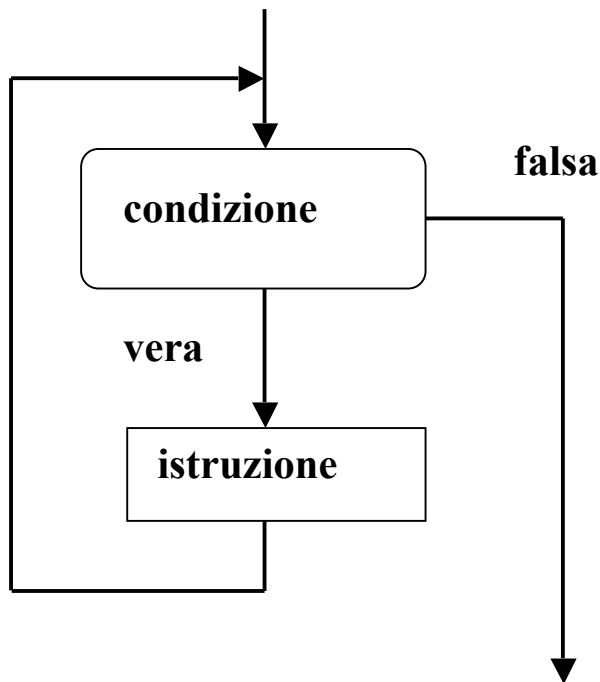
Significato:

- Se la condizione è **vera**, viene eseguita l'istruzione; successivamente si torna a valutare la condizione.
- Se la condizione è **falsa**, si passa ad eseguire l'istruzione successiva a `while`.

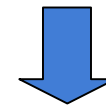


ISTRUZIONE while

```
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 :

```
/* Media di N voti*/
#include <stdio.h>
main() {      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);
}
```


ESEMPIO :

```
/* moltiplicazione come sequenza di somme */
#include <stdio.h>
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);
}
```

ESEMPIO ISTRUZIONE DI CICLO

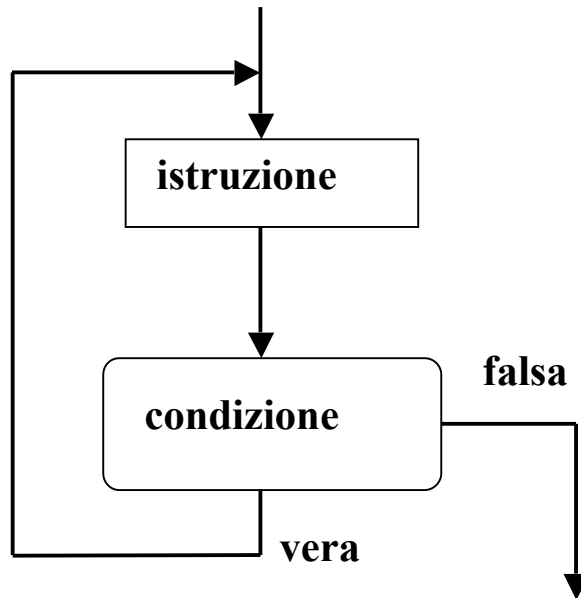
```
/* Calcolo del fattoriale di un numero N */

#include <stdio.h>
main()
{   int F, N, I;
    F=1; /* inizializzazione del fattoriale*/
    I=0; /* inizializzazione del contatore*/
    printf("Dammi N:");
    scanf("%d", &N);

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

ISTRUZIONE do..while

```
do <istruzione> while (<condizione>);
```



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

Anche se la condizione è falsa, l'istruzione **viene comunque eseguita almeno una volta**.

Esempio:

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

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

while e do

- 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

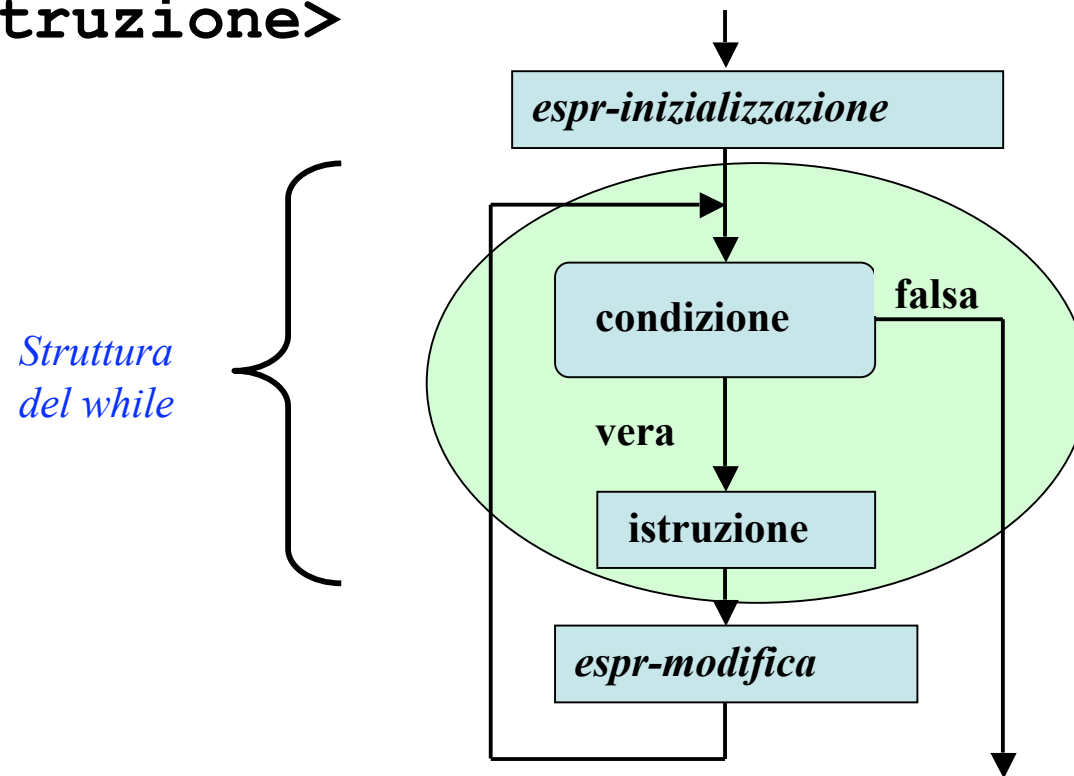
È una evoluzione dell'istruzione while che mira a eliminare alcune frequenti sorgenti di errore:

- mancanza delle *inizializzazioni delle variabili*
- mancanza della *fase di modifica del ciclo* (rischio di ciclo senza fine)
- In genere si usa quando e' noto il numero di volte in cui dovrà' essere eseguito il ciclo.

ISTRUZIONE for

Sintassi:

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



ISTRUZIONE for

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

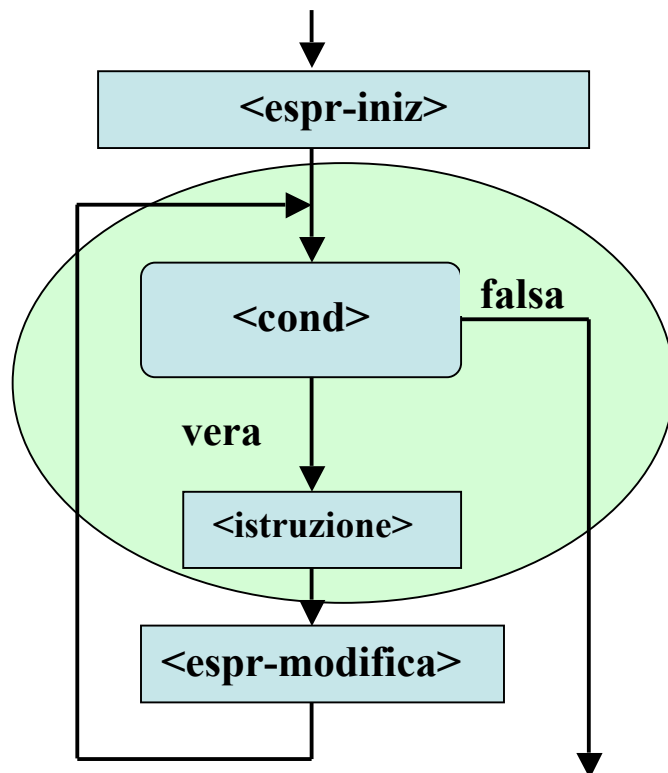
Inizializzazione: <espr-iniz>
valutata una e una sola volta
prima di iniziare l'iterazione.

Controllo: <cond>

valutata all'inizio di ogni iterazione,
per decidere se proseguire (come in un
while). Se manca si assume vera!

Modifica: <espr-modifica>

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



Esempio: for

```
#include <stdio.h>
void 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);
}
```

```
sum = 0;    i = 1;
while (i <= N) {
    printf("Dammi il voto");
    scanf("%d",&voto);
    sum=sum+voto;
    i=i++;
}
```

Esempio `for`: fattoriale

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

void 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);
}
```

Fattoriale con il while

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

#include <stdio.h>
void main()
{   int F, N, I;
    F=1; /* inizializzazione del fattoriale*/
    I=0; /* inizializzazione del contatore*/
    printf("Dammi N:");
    scanf("%d", &N);

    while (I < N)
    {I = I+1;
     F = I*F;

    }
    printf("Il fattoriale e' %d", F);
}
```

Cicli Annidati

Riprendiamo l'istruzione ripetitiva **while**:

```
while (<espressione>) <istruzione>;
```

- <istruzione> puo` essere una qualunque singola istruzione (semplice o strutturata): eventualmente anche una istruzione **while** →

cicli *annidati*

Cicli annidati: esempio

Si legga un numero naturale $N > 0$. Scrivere un programma che stampi una volta il numero 1, due volte il numero 2, ..., N volte il numero N.

- Dominio di ingresso $(0, 1, 2, \dots, N)$
- Dominio di uscita $((), (1), (1, 2, 2), (1, 2, 2, 3, 3, 3) \dots)$.

Soluzione:

```
#include <stdio.h>
main()
{
    int    N,I,J;
    printf("dammi N:");
    scanf("%d",&N);
    I=1;
    while ( I<=N )
        {
            /* corpo ciclo esterno */
            printf("Prossimo valore:");
            printf("%d",I);
            J=1;
            while (J<I)
                {
                    /* corpo ciclo interno */
                    printf("%d",I);
                    J=J+1;
                }
            I=I+1;
        }
}
```

Operatore condizionale

Operatore condizionale:

<condizione> ? <parteVera> : <parteFalsa>

restituisce il valore di <parteVera> o di <parte Falsa>, in base al seguente criterio:

- la <parteVera> viene valutata solo se la <condizione> è verificata (valore diverso da 0)
- la <parteFalsa> viene valutata solo se la <condizione> non è verificata (valore uguale a zero)

Esempio:

```
ris=a>b? a:b;
```

equivale a:

```
if (a>b)
    ris=a;
else
    ris=b;
```

Istruzioni per il trasferimento del controllo:

3

`break` e `continue`

Istruzione `break`:

L'istruzione `break` provoca l'uscita immediata dal ciclo (o da un'istruzione `switch`) in cui è racchiusa.

Esempio: fattoriale

```
for (F=1, i=1; ; i++) /* manca il controllo */
    if (i>N) break;
    else F=F*i;
```

Istruzione `continue`:

L'istruzione `continue` provoca l'inizio della successiva iterazione del ciclo in cui è racchiusa (non si applica a `switch`).

Esempio: somma dei valori pari compresi tra 1 e N

```
for (sum=0, i =1; i <= N; i++)
    if (i%2) continue;
    else sum+=i;
```