

# Istruzioni

---

# 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 **semplici**
  - ➔ Istruzioni **di controllo**

# Istruzioni semplici

---

- Qualsiasi *espressione* (aritmetica, logica, di assegnamento) 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 */`

→ `; /* istruzione 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**)

# Programmazione strutturata

---

## Concetti chiave:

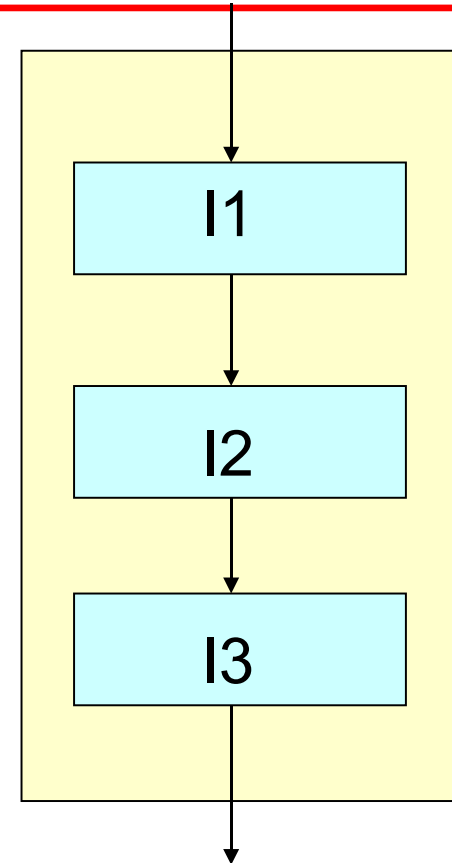
- concatenazione o composizione **BLOCCO**
- istruzione condizionale **SELEZIONE**
  - ramifica il flusso di controllo in base al valore vero o falso di una espressione (“*condizione di scelta*”)
- ripetizione o iterazione **CICLO**
  - esegue ripetutamente un’istruzione finché rimane vera una espressione (“*condizione di iterazione*”)
- **Bohm e Jacopini (1966)**: tali strutture di controllo 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)



# Blocco - esempio

---

```
/* programma che letti due numeri interi
   a terminale ne stampa la somma*/

#include <stdio.h>
main()
{
    /* INIZIO BLOCCO */
    int X,Y;
    printf("Inserisci due numeri interi: ");
    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 (1)

---

- Un identificatore **non** è visibile **prima** della sua dichiarazione

## Scorretto

```
main() {  
    int x = y*2;  
    int y = 3;  
}
```

## Corretto

```
main() {  
    int y = 3;  
    int x = y*2;  
}
```

# Regole di visibilità (2)

---

- *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***

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

# Regole di visibilità (3)

---

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

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



**Scorretto**

# Regole di visibilità (4)

---

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

## Scorretto

```
main() {  
  int x;  
  {  
    int y = 3;  
  }  
  x = y;  
  ...  
}
```

## Corretto

```
main() {  
  int x;  
  {  
    int y = 3;  
    x = y;  
  }  
  ...  
}
```

# Istruzioni condizionali

---

if, if-else, switch

# Istruzioni condizionali

---

- In C esistono due tipi di istruzioni condizionali:
  - Istruzione di **scelta** (semplice) (**if**, **if-else**)
  - Istruzione di **scelta multipla** (**switch**) (quest'ultima non è essenziale ma migliora l'espressività)
- In C, l'espressione condizionale ternaria (...?....:....) fornisce già un modo per esprimere il concetto di selezione, ma è poco leggibile in situazioni di media/alta complessità
- L'istruzione di scelta fornisce un più alto livello di espressività, comprensibilità e leggibilità

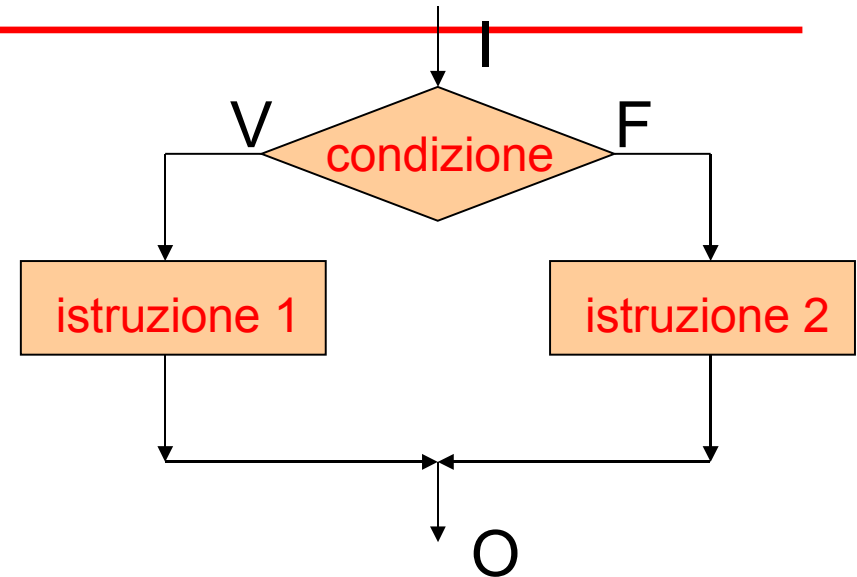
# Istruzione di scelta semplice

## Sintassi

```
if condizione
    istruzione 1;
else
    istruzione 2;
```

← notare il ;

← opzionale



- **condizione** è una **espressione logica** che viene valutata al momento dell'esecuzione dell'istruzione `if`
- Se la condizione è **vera** allora viene eseguita *istruzione 1* **altrimenti** viene eseguita *istruzione 2*
- In entrambi i casi l'esecuzione continua poi con l'istruzione che segue l'istruzione *if*
- **NOTA:** se *condizione* è falsa e la parte *else* (opzionale) è omessa, si passa subito all'istruzione che segue l'*if*

# Osservazione 1

---

- <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 e stampa 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);
}
```

# Osservazione 2

---

## Istruzione 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) */
```

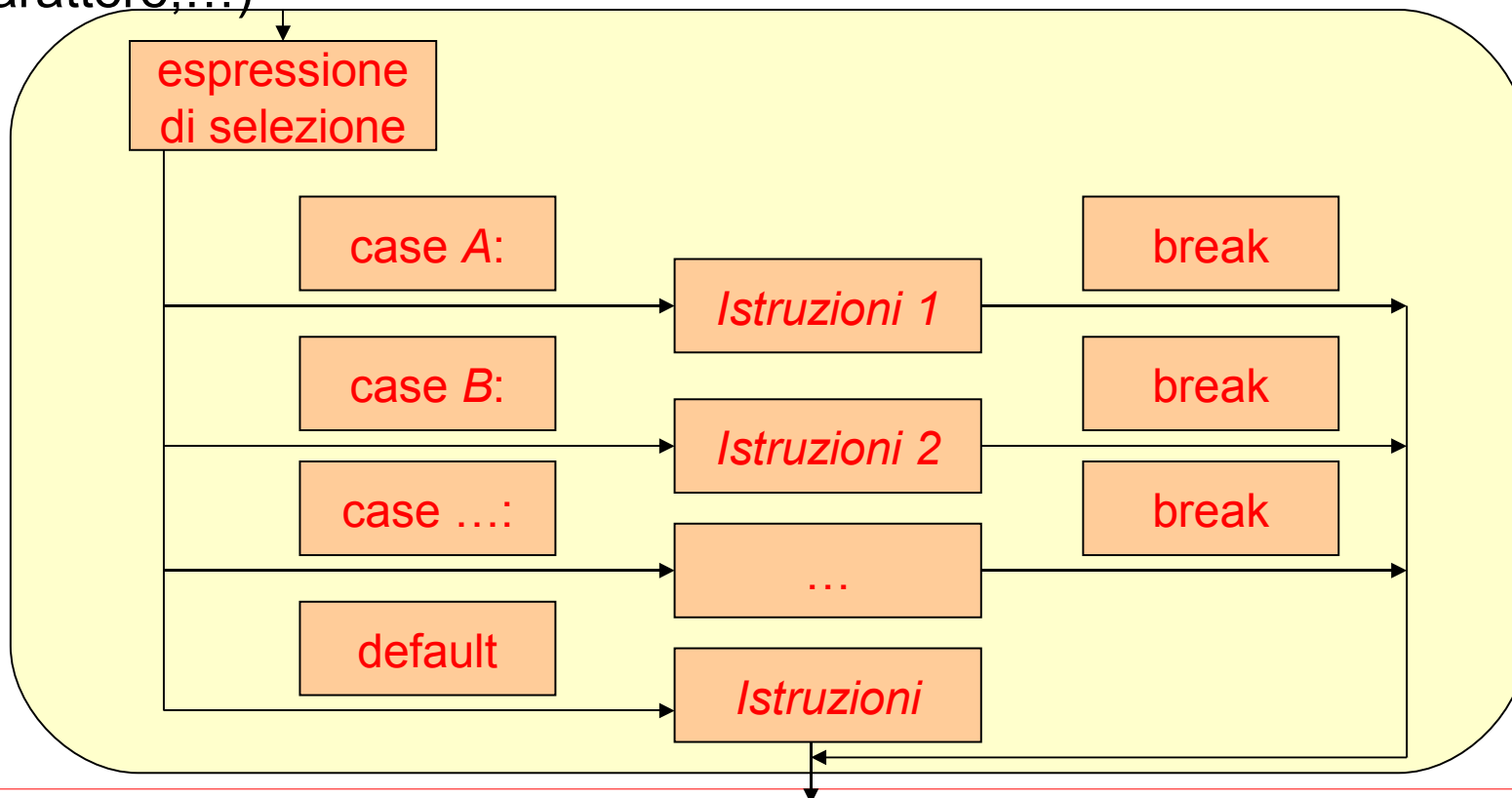
**else** è sempre associato all' **if** più interno

```
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

# 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,...)

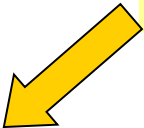


# Istruzione di scelta multipla

Sintassi:

```
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*
- altrimenti si eseguono le istruzioni del blocco *default*

# Istruzione di scelta multipla - Nota

---

- **<istruzioni>** denota una sequenza di istruzioni per cui non occorre un blocco per specificare più istruzioni
- I vari rami *non sono mutuamente esclusivi*: imboccato un ramo, l'esecuzione prosegue in generale con le istruzioni dei rami successivi
- Per avere rami mutuamente esclusivi occorre **forzare esplicitamente l'uscita** mediante l'istruzione **break**

# Istruzione di scelta multipla - esempio

---

```
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 - esempio

---

*Alternativa possibile (1):*

```
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 scelta multipla - esempio

---

*Alternativa possibile (2):*

```
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;
}
```

# Istruzione di scelta multipla - esempio

---

*Alternativa possibile (3):*

```
switch (mese) {
    case 2:    if (bisestile) giorni = 29;
              else giorni = 28;
              break;

    case 4:    case 6:
    case 9:    case 11:    giorni = 30;
              break;

    case 1:    case 3:    case 5:
    case 7:    case 8:    case 10:
    case 12:   giorni = 31;
              break;

    default:  printf("Errore! Mese non valido!");
}
}
```

## Scelta multipla – pro e contro

---

- L'istruzione **switch** evita una (lunga) serie di **if**
- **Tuttavia:**
  - È utilizzabile solamente con espressioni ed etichette di tipo **numerabile** (int, char)
  - **Non è utilizzabile** con numeri **reali** (float, double) o con tipi **strutturati** (stringhe, vettori, strutture,...)

# Istruzioni di iterazione

---

while, do-while, for

# Istruzioni di iterazione

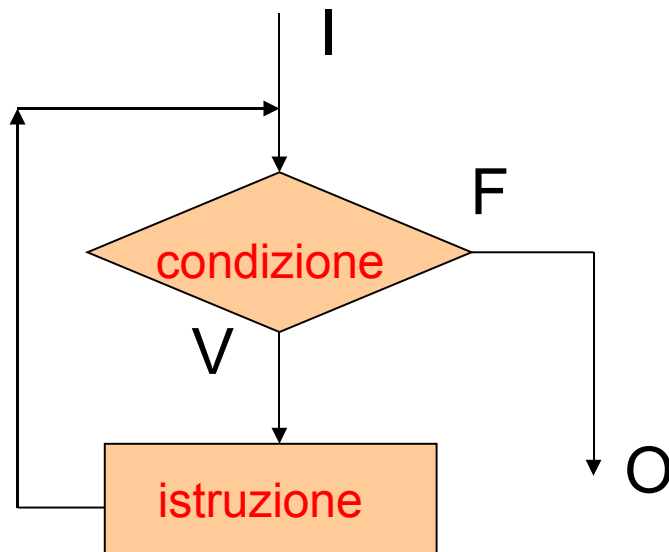
---

- Le istruzioni di iterazione forniscono strutture di controllo per esprimere la necessità di **ripetere** una certa istruzione **durante il verificarsi di una certa condizione**
- Per il Teorema di Jacopini-Böhm, una sola struttura di controllo iterativa è sufficiente (insieme al blocco e a un'istruzione di scelta) per calcolare ogni funzione computabile.
- Tuttavia, per **migliorare l'espressività del linguaggio**, in C vengono rese disponibili vari tipi di istruzioni iterative:
  - **while ()**
  - **do ... while ();**
  - **for (...)**

# Istruzione iterativa **while**

## Sintassi

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



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

# Istruzione iterativa **while**

---

- Prima o poi, *direttamente o indirettamente*, l'istruzione deve *modificare la condizione*: altrimenti → **CICLO INFINITO**
- Per questo motivo, quasi sempre *<istruzione>* è in realtà un blocco, che contiene una *istruzione in cui si modifica* qualche variabile che compare nella condizione

# Istruzione while - esempio

---

```
/* Calcolo della Media di N voti*/
#include <stdio.h>
main() {
    int sum=0,voto,N,i=1;
    float media;
    printf("Quanti sono i voti? ");
    scanf("%d",&N);
    while (i <= N) {
        printf("\nInserisci il voto n.%d:",i);
        scanf("%d",&voto);
        sum=sum+voto;
        i=i+1;
    }
    media=(float)sum/N;      /* ipotesi: N>0 */
    printf("\n Risultato: %f \n",media);
}
```

# Istruzione while - esempio

---

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

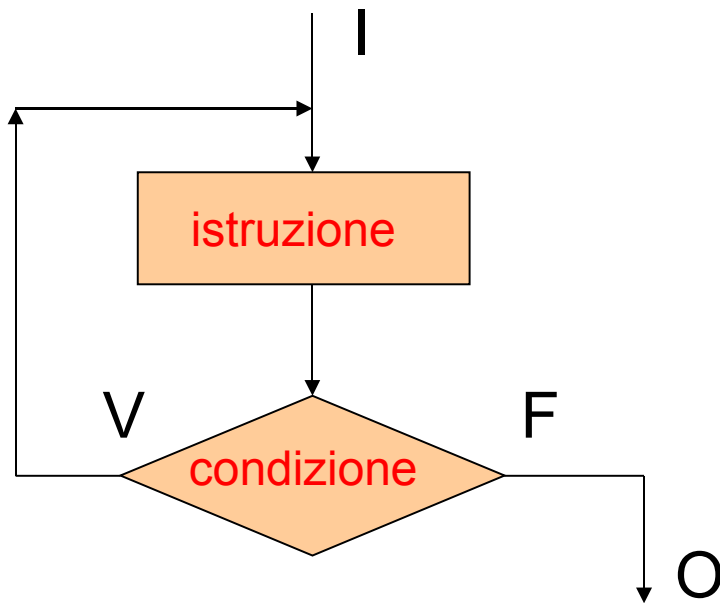
#include <stdio.h>
main() {
    int F, N, i;
    F=1;          /* inizializzazione del fattoriale*/
    i=1;          /* inizializzazione del contatore*/
    printf("Calcolo del fattoriale di un numero N\n\n");
    printf("Dammi N:");
    scanf("%d",&N);          /* ipotesi N >= 0 */
    while (i <= N) {
        F = F*i;
        i = i+1;
    }
    printf("Il fattoriale di %d è %d", N, F);
}
```

# Istruzione iterativa **do...while**

## Sintassi

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

- È una variante della precedente: la condizione viene verificata **dopo** aver eseguito <istruzione>
- Se la condizione è falsa, l'istruzione **viene comunque eseguita almeno una volta**



# Istruzione do..while - esempio

---

```
/* Calcolo del fattoriale di un numero N */
#include <stdio.h>
main() {
    int F, N, i;
    F=1;      /* inizializzazione del fattoriale */
    i=1;      /* inizializzazione del contatore */
    printf("Dammi N:");
    scanf("%d",&N);      /* ipotesi N > 0 */
    do {
        F = F*i;
        i = i+1;
    } while (i <= N);
    printf("Il fattoriale di %d è %d", N, F);
}
```

# Istruzione iterativa **do...while**

---

- Analogamente al *while*, per evitare il ciclo infinito, **<istruzione>** deve modificare prima o poi la condizione
- Si noti che, come nel caso del *while*, si esce dal ciclo quando la condizione è falsa
- **È adatta** a quei casi in cui, per valutare condizione, è necessario aver già eseguito **<istruzione>**  
(esempio tipico: **controllo di valori di input**)
- **Non è adatta** a quei casi in cui il ciclo può non dover essere *mai eseguito*

# Confronti - 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..while** 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);
```

- **Che cosa succede se n==0???**

# Esempio – controllo valori di input

---

- Esempio 1: *n deve essere positivo*

do

```
scanf ("%d", &n);
```

```
while (n<=0);
```

- Esempio 2: *n deve essere compreso tra 3 e 15*

do

```
scanf ("%d", &n);
```

```
while ((n<3) || (n>15));
```

- Esempio 3: *n deve essere negativo o compreso tra 10 e 20*

do

```
scanf ("%d", &n);
```

```
while ((n>=0) && (n<10) || (n>20));
```

# Esempio

---

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

```
...
```

```
do
```

```
    scanf ("%c", &ch);
```

```
while (ch != '.')
```

- Oppure:

```
ch = '*';
```

```
while (ch != '.')
```

```
    scanf ("%c", &ch);
```

# Istruzione iterativa **for**

---

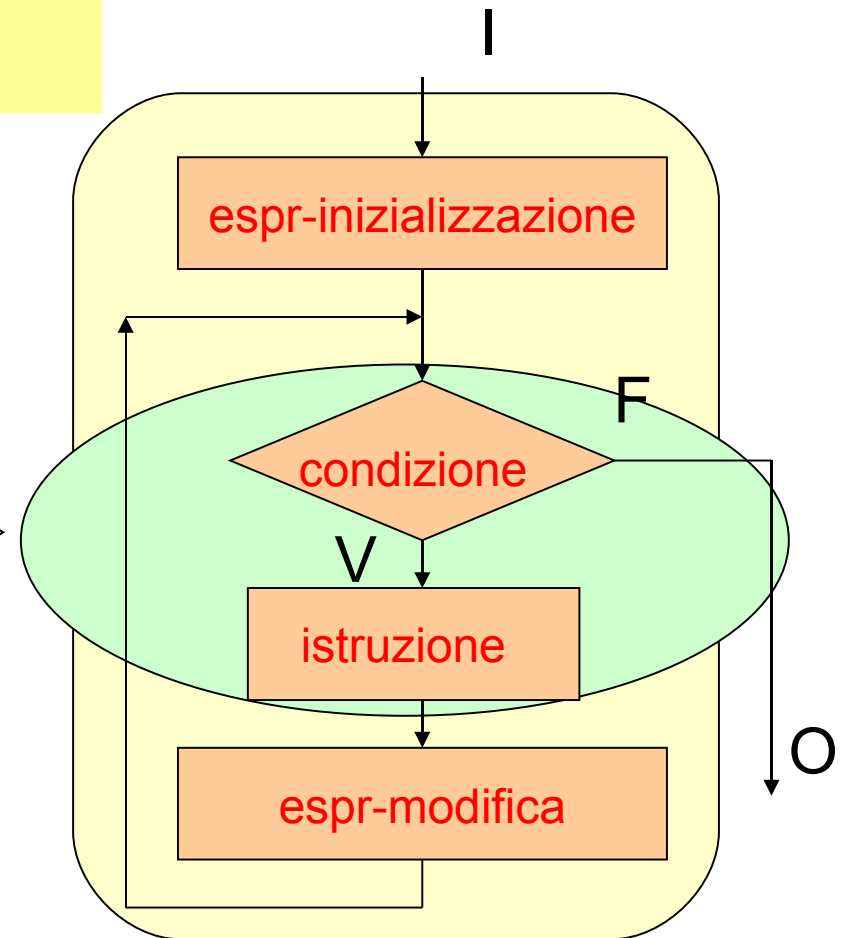
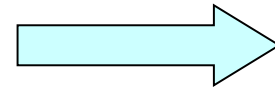
- Anche se, in genere, si usa quando è noto quante volte il ciclo dovrà essere eseguito (**contatore**), in C **NON** è (solo) l'istruzione per implementare cicli "**definiti**" come in altri linguaggi
- È una evoluzione dell'istruzione **while** che mira a eliminare alcune frequenti sorgenti di errore:
  - mancanza delle *inizializzazioni delle variabili*
  - mancanza della *fase di modifica del ciclo* (**rischio di ciclo senza fine**)
- Per questo, l'istruzione **for** comprende esplicitamente:
  - Una espressione di **inizializzazione**
  - Una espressione di **modifica del ciclo**

# Istruzione iterativa for

## Sintassi

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

struttura  
del while



# Istruzione iterativa for

## Sintassi

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

*Espressione di inizializzazione:* **<espr-iniz>**

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

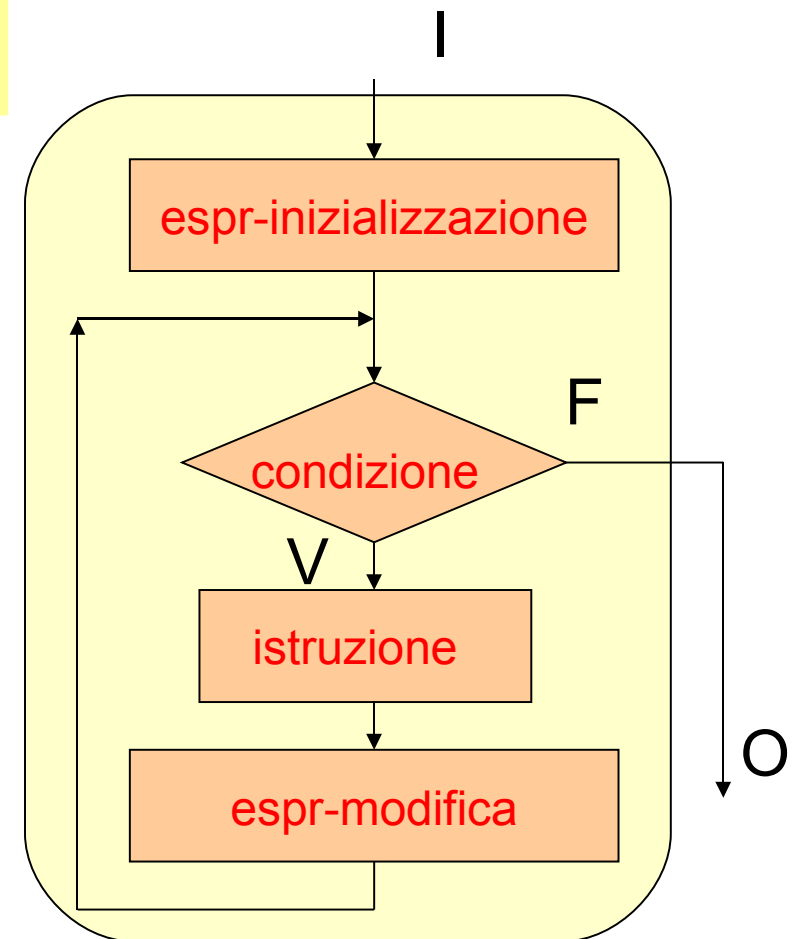
*Condizione:* **<cond>**

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;
}
```

# Istruzione for - esempio

```
/* Calcolo della Media di N voti - uso del for*/
#include <stdio.h>
main() {
    int sum=0,voto,N,i;
    float media;
    printf("Quanti sono i voti? ");
    scanf("%d",&N);
    for(i=1; i <= N; i++) {
        printf("\nInserisci il voto n.%d:",i);
        scanf("%d",&voto);
        sum=sum+voto;
    }
    media=(float)sum/N;    /* ipotesi: N>0 */
    printf("\n Risultato: %f \n",media);
}
```

non c'è più (non serve) l'inizializzazione del contatore i al di fuori del ciclo

non c'è più la modifica del contatore i all'interno del ciclo

# Istruzione for - esempio

---

```
/* Calcolo del fattoriale di un numero N - uso del for */

#include <stdio.h>
main() {
    int F, N, i;
    F=1;          /* inizializzazione del fattoriale*/

    printf("Calcolo del fattoriale di un numero N\n\n");
    printf("Inserisci N:");
    scanf("%d",&N);          /* ipotesi N > 0 */
    for (i=1; i <= N; i++)
        F = F*i;

    printf("Il fattoriale di %d è %d", N, F);
}
```

# Istruzione for - osservazioni

---

- Per inizializzare (o modificare) più variabili si può usare una **espressione composta** (operatore , )
- L'espressione di modifica della condizione appare in modo più evidente (la sua mancanza "si nota"!)
- L' <istruzione> che compare nel corpo del ciclo è *solo l'operazione vera e propria da ripetere*:
  - migliore leggibilità
  - (spesso non è più necessario un blocco)
- Ognuna delle espressioni può essere omessa, ma il separatore ; deve rimanere
- Se manca <condizione> la si assume vera

# Istruzioni **break** e **continue**

---

- Istruzioni **senza operandi** che modificano il normale percorso dei cicli
- Il **break** provoca l'immediata uscita da un blocco o ciclo
  - ➔ L'istruzione eseguita dopo il **break** è quella **successiva** al blocco in cui compare il break
  - ➔ L'utilizzo del break è già stato visto insieme all'istruzione "switch...case"
  - ➔ All'interno di cicli, serve soprattutto per interrompere l'esecuzione in caso di errori o di condizioni irregolari

# Istruzioni **break** e **continue**

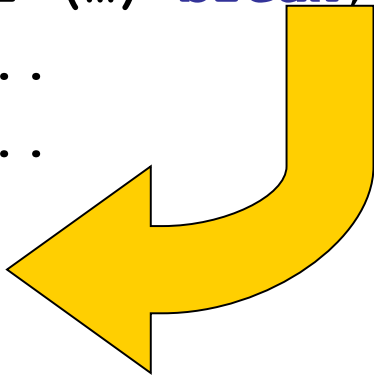
---

- Il **continue** evita l'esecuzione delle istruzioni del blocco successive (*come se fosse un salto alla parentesi } che chiude il blocco*) e causa la nuova valutazione dell'espressione condizionale

# Istruzione **break**

---

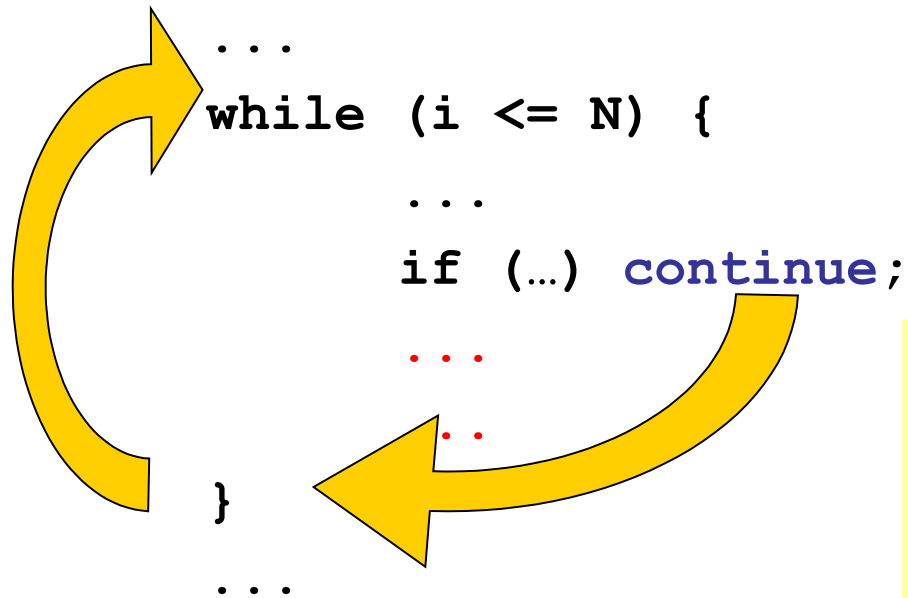
```
...  
while (i <= N) {  
    ...  
    if (...) break;  
    ...  
    ...  
}  
...
```



Quando viene eseguito il **break**,  
il controllo viene trasferito alla  
**prima istruzione successiva**  
al blocco in cui si trova il **break**

# Istruzione `continue`

---



Quando viene eseguito il `continue`, il controllo viene trasferito alla **fine del blocco** causando una nuova valutazione della condizione che controlla il ciclo

Dunque **si saltano le istruzioni** tra il `continue` e la `parentesi }` che chiude il blocco

# Istruzioni **break** e **continue**

---

## **ATTENZIONE:**

- Le istruzioni *break* e *continue* alterano la normale struttura di un programma
- Sono quindi da usare **con cautela**
- E' sempre possibile fare a meno del loro uso mediante l'utilizzo di apposite variabili booleane di controllo