

Costrutti di iterazione

Stefano Ferrari

Università degli Studi di Milano
stefano.ferrari@unimi.it

Programmazione

anno accademico 2017–2018

Iterazione

Data un'espressione logica e un blocco di attività di processo

1. si valuta l'espressione logica
2. se ha valore *vero*: si esegue il blocco e si torna al punto 1
se ha valore *falso*: si passa oltre

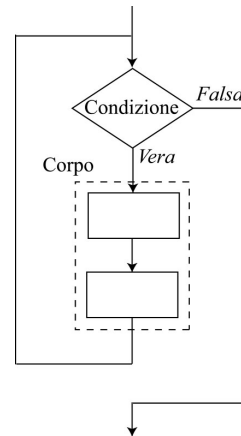
Si definisce *condizione* del ciclo l'espressione, *corpo* il blocco

- ▶ è possibile non eseguire mai il blocco
- ▶ le variabili dell'espressione devono essere inizializzate
- ▶ il blocco deve poter modificare il valore dell'espressione;
altrimenti verrebbe eseguito indefinitamente (*ciclo infinito*)
- ▶ al termine del ciclo, la condizione è sempre falsa

Il costrutto while

È detto anche **ciclo a condizione iniziale**

```
while (espressione)
{
    istruzione/i
}
```



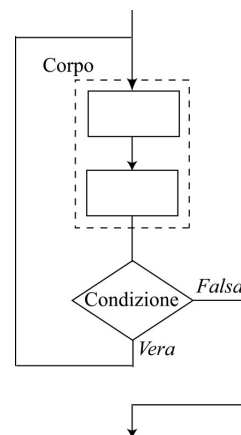
Se la condizione ha valore *vero*, esegue il corpo e rivaluta la condizione; altrimenti passa all'istruzione successiva al corpo

- ▶ ogni istruzione termina con ;
- ▶ il blocco nel suo complesso no

Il costrutto do...while

È detto anche **ciclo a condizione finale**

```
do {
    istruzione/i
} while (espressione);
```



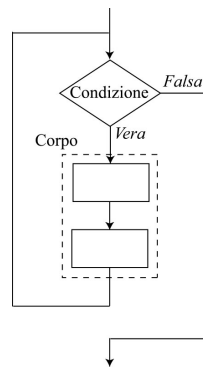
Esegue il corpo una volta; se la condizione ha valore *vero*, lo riesegue e rivaluta la condizione; altrimenti procede

- ▶ ogni istruzione termina con ;
- ▶ **l'intero costrutto termina con ;**

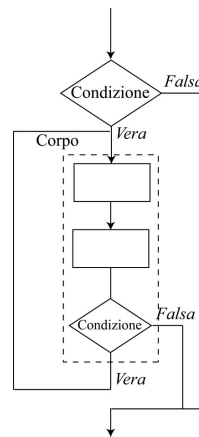
Equivalenza while \rightarrow do...while

I cicli a condizione iniziale e a condizione finale sono equivalenti

```
while (espressione)
{
    istruzione/i
}
```



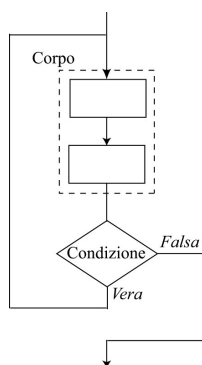
```
if (espressione)
{
    do {
        istruzione/i
    } while (espressione);
}
```



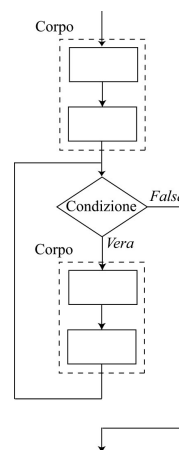
Equivalenza do...while \rightarrow while

I cicli a condizione iniziale e a condizione finale sono equivalenti

```
do {
    istruzione/i
}
while (espressione);
```



```
{
    istruzione/i
}
while (espressione)
{
    istruzione/i
}
```



Il costrutto for

```
for (espressione1 ; espressione2 ; espressione3)
{
    istruzione/i
}
```

1. valuta *espressione1*
 2. valuta *espressione2*: se ha valore *falso* procede;
se ha valore *vero*, esegue il corpo e poi valuta *espressione3*
- ▶ *espressione2* è di tipo logico
 - ▶ *espressione1* e *espressione3* hanno effetti collaterali
 - ▶ *espressione1* inizializza *espressione2*
 - ▶ *espressione3* modifica *espressione2*

Cicli a conteggio

Il costrutto for tipicamente serve a realizzare **cicli a conteggio**,
cioè i **cicli il cui corpo va eseguito un numero dato di volte**

Per *i* che cresce da 1 a *n*...

Per *i* che cala da *n* a 1...

```
for (i = 1; i <= n; i++)
{
    istruzione/i
}
```

```
for (i = n; i >= 1; i--)
{
    istruzione/i
}
```

Le varie parti del costrutto corrispondono agli elementi già elencati

- ▶ *espressione1*: **inizializzazione**
- ▶ *espressione2*: **condizione di permanenza**
- ▶ *espressione3*: **aggiornamento**

Equivalenza while \leftrightarrow for

I cicli for equivalgono ai cicli while

```
{ inizializzazione; condizione di permanenza; aggiornamento }  
for (espressione1 ; espressione2 ; espressione3)  
{  
    istruzione/i  
}  
  
espressione1; { inizializzazione }  
while (espressione2) { condizione di permanenza }  
{  
    istruzione/i  
    espressione3; { aggiornamento }  
}
```

L'istruzione vuota

Può avvenire che non occorran istruzioni nel corpo di un ciclo perché tutto ha luogo nell'inizializzazione e aggiornamento

Allora si usa un'istruzione priva di simboli, costituita dal solo ;

```
for (espressione1 ; espressione2 ; espressione3) ;
```

Esempio: valuta se n è composto o primo (d sta per "divisore")

```
for (d = 2 ; (n % d != 0) && (d < n) ; d++);  
return ( (d < n) ? FALSE : TRUE );
```

L'istruzione vuota può confondersi con la seguente e indurre in errore: meglio separarle con una riga vuota!

```
for (d = 2 ; (n % d != 0) && (d < n) ; d++);  
  
return ( (d < n) ? FALSE : TRUE );
```

L'operatore virgola

Può accadere di dover modificare più variabili in una sola istruzione, volendo però evitare effetti collaterali multipli poco controllabili

L'operatore virgola combina più espressioni qualsiasi in un'istruzione sola

espressioneA, espressioneB, espressioneC

È **associativo da sinistra a destra**:

- ▶ si valutano le espressioni da sinistra a destra
- ▶ il suo valore è il valore dell'ultima espressione
- ▶ ha la priorità più bassa (per cui non occorrono parentesi)

Di solito **si usa con espressioni che hanno effetti collaterali**:

serve a **raggrupparle in una sola istruzione** leggibile

(es.: **inizializzazioni** e **aggiornamenti multipli** nei cicli for)

L'operatore virgola (esempi)

```
int i, q;  
for (i = 1; i <= n; i++)  
{  
    q = i * i;  
    StampaIntero(q);  
    ACapo();  
}
```

Che cosa succede se nel codice sopra riportato

1. si accoda il calcolo del quadrato all'inizializzazione $i = 1$?

```
for (i = 1, q = i * i; i <= n; i++)  
{ StampaIntero(q); ACapo(); }
```

Si calcola solo il quadrato di $i = 1$

2. si premette il calcolo del quadrato all'inizializzazione $i = 1$?

```
for (q = i * i, i = 1; i <= n; i++)  
{ StampaIntero(q); ACapo(); }
```

Si calcola solo una volta il quadrato di un numero casuale

L'operatore virgola (esempi) (2)

```
int i, q;  
for (i = 1; i <= n; i++)  
{  
    q = i * i;  
    StampaIntero(q);  
    ACapo();  
}
```

Che cosa succede se nel codice sopra riportato

3. si premette il calcolo del quadrato all'aggiornamento?

```
for (i = 1; i <= n; q = i * i, i++)  
{ StampaIntero(q); ACapo(); }
```

Il primo quadrato è sbagliato, gli altri ritardati di un passo

4. si accoda il calcolo del quadrato all'aggiornamento?

```
for (i = 1; i <= n; i++, q = i * i)  
{ StampaIntero(q); ACapo(); }
```

Il primo quadrato è sbagliato, gli altri giusti e sincronizzati

Sarebbe corretto:

```
for (i = 1, q = 1; i <= n; i++, q = i * i)  
{ StampaIntero(q); ACapo(); }
```

L'operatore virgola (esempi) (3)

- `(i = 1), j = 2, k = i + j;`

Al termine, l'espressione vale 3, i vale 1, j vale 2, k vale 3

- `i = 1;
j = 5;
++i, i+j;`

Al termine, l'espressione vale 7, i vale 2, j vale 5

- `i = 1,2,3;`

Al termine, l'espressione vale 3, i vale 1

- `i = (1,2,3);`

Al termine, l'espressione vale 3, i vale 3

L'istruzione break

In C si può uscire da un ciclo in punti diversi dalla condizione di permanenza (ciò è contrario alla programmazione strutturata)

L'istruzione `break` provoca l'uscita dal blocco corrente (qualunque blocco sia: `switch`, `while`, `do-while` e `for`)

Esempio: valuta se n è composto o primo (d sta per "divisore")

```
for (d = 2 ; d < n ; d++)  
    if (n % d == 0)  
        break;
```

Si può evitare combinando la condizione negata con quella di permanenza

```
for (d = 2 ; (n % d != 0) && (d < n); d++);
```

L'istruzione continue

L'istruzione `continue` provoca il passaggio all'iterazione successiva, ignorando il resto del corpo

- ha senso solo nel corpo di un ciclo

È sempre sostituibile da un `if` con la condizione negata

Esempio: somma i quadrati dei numeri dispari compresi fra 1 e n

```
s = 0;  
for (i = 1 ; i <= n ; i++)  
{  
    if (i % 2 == 0)  
        continue;  
    s += i * i;  
}
```

```
s = 0;  
for (i = 1 ; i <= n ; i++)  
{  
    if (i % 2 != 0)  
    {  
        s += i * i;  
    }  
}
```