

Istruzioni iterative

Esempio: Leggere 5 interi, calcolarne la somma e stamparli.

- ▶ Variante non accettabile: 5 variabili, 5 istruzioni di lettura, 5 ...

```
int i1, i2, i3, i4, i5;
scanf("%d", &i1);
...
scanf("%d", &i5);
printf("%d", i1 + i2 + i3 + i4 + i5);
```

- ▶ Variante migliore che utilizza solo 2 variabili:

```
int somma, i;
somma = 0;
scanf("%d", &i);
somma = somma + i;
...          /* per 5 volte */
scanf("%d", &i);
somma = somma + i;
printf("%d", somma);
```

⇒ conviene però usare un'istruzione iterativa

Iterazione determinata e indeterminata

- ▶ Le **istruzioni iterative** permettono di ripetere determinate azioni più volte:

- ▶ un numero di volte fissato ⇒ **iterazione determinata**

Esempio:

fai un giro del parco di corsa per 10 volte

- ▶ finché una condizione rimane vera ⇒ **iterazione indeterminata**

Esempio:

finché non sei sazio

prendi una ciliegia dal piatto e mangiala

Istruzione **while**

Permette di realizzare l'iterazione in C.

Sintassi:

```
while (espressione)
    istruzione
```

- ▶ **espressione** è la **guardia** del ciclo
- ▶ **istruzione** è il **corpo** del ciclo (può essere un blocco)

Semantica:

1. viene valutata l'**espressione** (**condizione di continuazione**)
2. se è vera si esegue **istruzione** e si torna ad eseguire l'intero **while**
3. se è falsa si termina l'esecuzione del **while**

- ▶ Nota: se **espressione** è falsa all'inizio, il ciclo non fa nulla.

Iterazione determinata

Esempio: Stampa 100 asterischi.

- ▶ Si utilizza un **contatore** per contare il numero di asterischi stampati.

```
Algoritmo: stampa di 100 asterischi
inizializza il contatore a 0
while il contatore è minore di 100
{ stampa un '*'
  incrementa il contatore di 1 }
```

- ▶ Implementazione:

```
int i;
i = 0;
while (i < 100) {
    putchar('*');
    i = i + 1;
}
```

- ▶ come già sappiamo, la variabile **i** viene detta **variabile di controllo** del ciclo.

Iterazione determinata

Esempio: Leggere 10 interi, calcolarne la somma e stamparla.

- ▶ Si utilizza un contatore per contare il numero di interi letti.

```
int conta, dato, somma;
printf("Immetti 10 interi: ");
somma = 0;
conta = 0;
while (conta < 10) {
    scanf("%d", &dato);
    somma = somma + dato;
    conta = conta + 1;
}
printf("La somma è %d\n", somma);
```

Esempio: Leggere un intero N seguito da N interi e calcolare la somma di questi ultimi.

- ▶ Simile al precedente: il numero di ripetizioni necessarie non è noto al momento della scrittura del programma ma lo è al momento dell'esecuzione del ciclo.

```
int lung, conta, dato, somma;
printf("Immetti la lunghezza della sequenza ");
printf("seguita dagli elementi della stessa: ");
scanf("%d", &lung);
somma = 0;
conta = 0;
while (conta < lung) {
    scanf("%d", &dato);
    somma = somma + dato;
    conta = conta + 1;
}
printf("La somma è %d\n", somma);
```

Esempio: Leggere 10 interi **positivi** e stamparne il massimo. 

- ▶ Si utilizza un **massimo corrente** con il quale si confronta ciascun numero letto.

```
int conta, dato, massimo;
printf("Immetti 10 interi: ");
massimo = 0;
conta = 0;
while (conta < 10) {
    scanf("%d", &dato);
    if (dato > massimo)
        massimo = dato;
    conta = conta + 1;
}
printf("Il massimo è %d\n", massimo);
```

Esercizio

Leggere 10 interi **arbitrari** e stamparne il massimo.

Istruzione for

- ▶ I cicli visti fino ad ora hanno queste caratteristiche comuni:
 - ▶ utilizzano una variabile di controllo
 - ▶ la guardia verifica se la variabile di controllo ha raggiunto un limite prefissato
 - ▶ ad ogni iterazione si esegue un'azione
 - ▶ al termine di ogni iterazione viene incrementato (decrementato) il valore della variabile di controllo

Esempio: Stampare i numeri **pari** da 0 a N.

```
i = 0; /* Inizializzazione della var. di controllo */
while (i <= N) { /* guardia */
    printf("%d ", i); /* Azione da ripetere */
    i=i+2; /* Incremento var. di controllo */
}
```

- ▶ L'istruzione **for** permette di gestire direttamente questi aspetti:

```
for (i = 0; i <= N; i=i+2)
    printf("%d", i);
```

Sintassi:

```
for (istr-1; espr-2; istr-3)
    istruzione
```

- ▶ `istr-1` serve a inizializzare la variabile di controllo
- ▶ `espr-2` è la verifica di fine ciclo
- ▶ `istr-3` serve ad aggiornare la variabile di controllo alla fine del corpo del ciclo
- ▶ `istruzione` è il corpo del ciclo

Semantica: l'istruzione `for` precedente è equivalente a

```
istr-1;
while (espr-2) {
    istruzione
    istr-3
}
```

Esempio:

```
for (i = 1; i <= 10; i=i+1)    ==>  i: 1, 2, 3, ..., 10
for (i = 10; i >= 1; i=i-1)   ==>  i: 10, 9, 8, ..., 2, 1
for (i = -4; i <= 4; i = i+2) ==>  i: -4, -2, 0, 2, 4
for (i = 0; i >= -10; i = i-3) ==> i: 0, -3, -6, -9
```

- ▶ In realtà, la sintassi del `for` è


```
for (espr-1; espr-2; espr-3)
    istruzione
```

 dove `espr-1`, `espr-2` e `espr-3` sono delle espressioni qualsiasi (in C anche l'assegnamento è un'espressione ...).
- ▶ È buona prassi:
 - ▶ usare ciascuna `espr-i` in base al significato descritto prima
 - ▶ non modificare la variabile di controllo nel corpo del ciclo
- ▶ Ciascuna delle tre `espr-i` può anche mancare:
 - ▶ i “;” vanno messi lo stesso
 - ▶ se manca `espr-2` viene assunto il valore vero
- ▶ Se manca una delle tre `espr-i` è meglio usare un'istruzione `while`

Esempio: Leggere 10 interi **positivi** e stamparne il massimo. ▶

```
int conta, dato, massimo;
printf("Immetti 10 interi: ");
massimo = 0;
for (conta=0; conta<10; conta=conta+1)
{
    scanf("%d", &dato);
    if (dato > massimo)
        massimo = dato;
}
printf("Il massimo è %d\n", massimo);
```

Cicli annidati

- ▶ Il corpo di un ciclo può contenere a sua volta un ciclo.

Esempio: Stampa della **Tavola Pitagorica**
Algoritmo

```
for ogni riga tra 1 e 10
{ for ogni colonna tra 1 e 10
  stampa riga * colonna
  stampa un a capo }
```

- ▶ Traduzione in C

```
int riga, colonna;
const int Nmax = 10; /* indica il numero di righe e di
colonne */
for (riga = 1; riga <= Nmax; riga=riga+1) {
    for (colonna = 1; colonna <= Nmax; colonna=colonna+1)
        printf("%d ", riga * colonna);
    putchar('\n'); }
```

Iterazione indefinita

- ▶ In alcuni casi il numero di iterazioni da effettuare non è noto prima di iniziare il ciclo, perché dipende dal verificarsi di una **condizione**.

Esempio: Leggere una sequenza di interi che termina con 0 e calcolarne la somma.

Input: $n_1, \dots, n_k, 0$ (con $n_i \neq 0$)

Output: $\sum_{i=1}^k n_i$

```
int dato, somma = 0;
scanf("%d", &dato);
while (dato != 0) {
    somma = somma + dato;
    scanf("%d", &dato);
}
printf("%d", somma);
```

Istruzione do-while

- ▶ Nell'istruzione **while** la condizione viene controllata all'**inizio** di ogni iterazione.
- ▶ L'istruzione **do-while** è simile all'istruzione **while**, ma la **condizione** viene controllata alla **fine** di ogni iterazione

Sintassi:

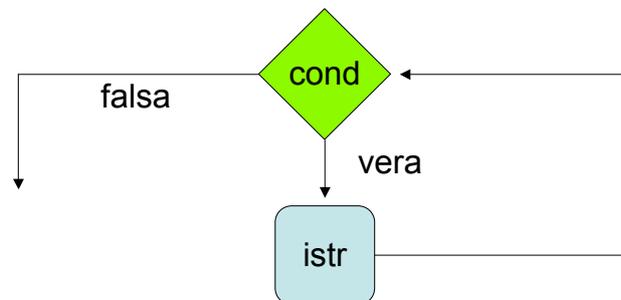
```
do
    istruzione
while (espressione);
```

Semantica: è equivalente a

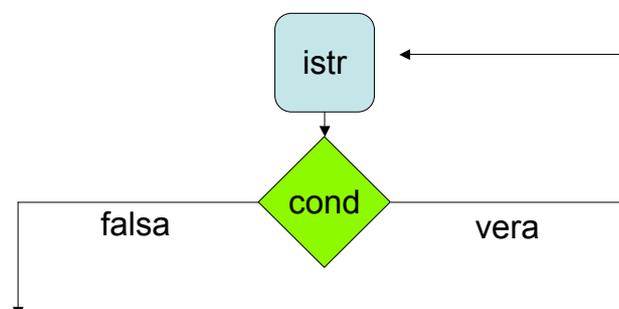
```
istruzione
while (espressione)
    istruzione
```

⇒ una iterazione viene eseguita **comunque**.

Istr. di controllo: while (cond) istr



Istr. di controllo: do istr while (cond)



Esempio: Lunghezza di una sequenza di interi terminata da 0, usando **do-while**.

```
main() {
    int lunghezza = 0; /* lunghezza della sequenza */
    int dato; /* dato letto di volta in volta */
    printf("Inserisci una sequenza di interi (0 fine seq.)\n");
    do {
        scanf("%d", &dato);
        lunghezza=lunghezza+1;
    } while (dato != 0);
    printf("La sequenza è lunga %d\n", lunghezza - 1);
}
```

- ▶ Nota: lo 0 finale non è conteggiato (non fa parte della sequenza, fa da terminatore)