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: $n_1, \ldots, n_k, 0$ (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:

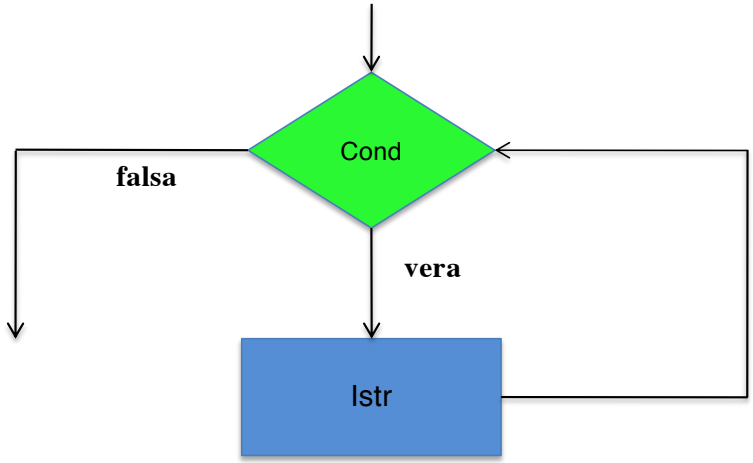
```
do
  istruzione
while (espressione);
```

Semantica: è equivalente a

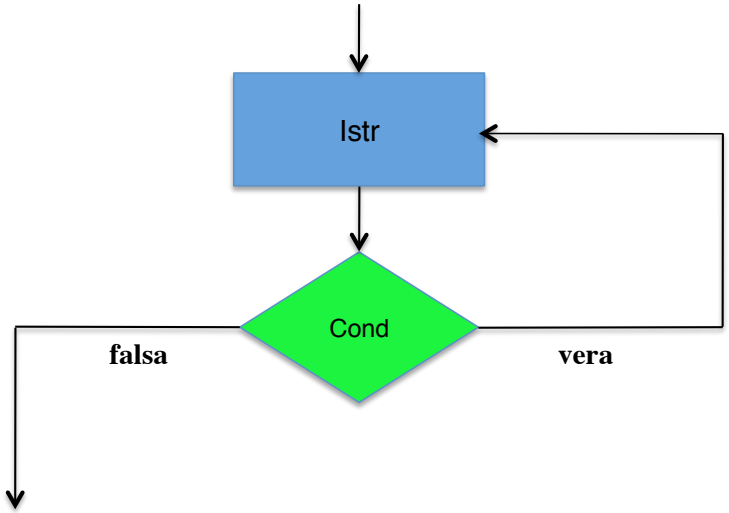
```
istruzione
while (espressione)
  istruzione
```

$\Longrightarrow$ una iterazione viene eseguita comunque. Ha quindi senso usare **do-while** quando sappiamo che deve essere eseguita **almeno una** iterazione.

while cond istr



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)

Assegnamento e altri operatori

- In C, l'operazione di assegnamento `x = exp` è un'espressione
  - il valore dell'espressione è il valore di `exp` (che è a sua volta un'espressione)
  - la valutazione dell'espressione `x = exp` ha un side-effect: quello di assegnare alla variabile `x` il valore di `exp`
- Dunque in realtà, "=" è un operatore (associativo a destra).
  Esempio: Qual è l'effetto di `x = y = 4` ?
  - È equivalente a: `x = (y = 4)`
  - `y = 4` ... espressione di valore `4` con modifica (side-effect) di `y`
  - `x = (y = 4)` ... espressione di valore `4` con ulteriore modifica su `x`
- L'eccessivo uso di assegnamenti come espressioni rende il codice difficile da comprendere e quindi anche da correggere/modificare.

Operatori di incremento e decremento

- Assegnamenti del tipo:
```
i = i + 1
i = i - 1
```
sono molto comuni.
  - operatore di incremento: `++`
  - operatore di decremento: `--`
- In realtà `++` corrisponde a due operatori:
- postincremento: `i++`
  - il valore dell'espressione è il valore di `i`
  - side-effect: incrementa `i` di `1`
- L'effetto di

```
int i,j;
i=6;
j=i++;
```

è j=6, i=7.

- preincremento: `++i`
  - il valore dell'espressione è il valore di `i+1`
  - side-effect: incrementa `i` di `1`
- L'effetto di

```
int i,j;
i=6;
j=++i;
```

è j=7, i=7.

(analogamente per `i--` e `--i`)

- Nota sull'uso degli operatori di incremento e decremento

Esempio:

	Istruzione	x	y	z
1	`int x, y, z;`	?	?	?
2	`x = 4;`	4	?	?
3	`y = 2;`	4	2	?
4a	`z = (x + 1) + y;`	4	2	7
4b	`z = (x++) + y;`	5	2	6
4c	`z = (++x) + y;`	5	2	7

- N.B.: Non usare mai in questo modo!
  In un'istruzione di assegnamento non ci devono essere altri side-effect (oltre a quello dell'operatore di assegnamento) !!!
- Riscrivere, ad esempio, come segue:

4b: `z = (x++) + y;` $\Longrightarrow$

```
z = x + y;
x++;
```

4c: `z = (++x) + y;` $\Longrightarrow$

```
x++;
z = x + y;
```

Ordine di valutazione degli operandi

- In generale il C non stabilisce quale è l'ordine di valutazione degli operandi nelle espressioni. Dipende quindi dal compilatore.

  Esempio:
  ```
  int x, y, z;
  x = 2;
  y = 4;
  z = x++ + (x * y);
  ```

- Quale è il valore di `z`?
  - se viene valutato prima `x++`: $2 + (3 * 4) = 14$
  - se viene valutato prima `x*y`: $(2 * 4) + 2 = 10$

Forme abbreviate dell'assegnamento

```
a = a + b;   ⟹   a += b;
a = a - b;   ⟹   a -= b;
a = a * b;   ⟹   a *= b;
a = a / b;   ⟹   a /= b;
a = a % b;   ⟹   a %= b;
```

Espressioni condizionali

Al posto di

```
int x, y, z;
if (x > y)
z = x;
else
z = y;
```

possiamo scrivere, parafrasando quanto scritto, come:

```
int x, y, z;
z = (x > y)? x : y;
```

Abbiamo usato quelle che vengono definite espressioni condizionali, la cui sintassi è:

```
(espr-1)? espr-2 : espr-3
```

A seconda dell'esito della valutazione della prima espressione, si valuta la seconda oppure la terza.