Istruzioni di Controllo in C

Emilio Di Giacomo

Limite delle istruzioni viste

- L’insieme delle istruzioni che abbiamo visto fino ad ora consiste per lo più di:
  - dichiarazioni e assegnazioni di variabili
  - espressioni aritmetiche
  - invocazioni di funzioni di libreria
- Con queste istruzioni non sono molte le cose che possiamo fare

Istruzioni di controllo

- In questa lezione introduciamo le istruzioni di controllo; esse consentono di:
  - confrontare dei dati e stabilire comportamenti diversi in funzione dell'esito del confronto (istruzioni condizionali)
  - ripetere una stessa sequenza di istruzioni più volte, fintanto che certe condizioni permangono (istruzioni iterative)

Istruzioni condizionali

- Un'istruzione condizionale serve a verificare se una determinata condizione è vera o falsa
  - la condizione è tipicamente un predicato
  - poiché però in C i predicati sono espressioni intere, qualunque espressione intera può essere usata come condizione
- In funzione dell'esito, l'istruzione condizionale stabilisce le prossime istruzioni da eseguire

Istruzioni condizionali

- In C esistono tre istruzioni condizionali:
  - if
  - if-else
  - switch
- Esiste inoltre un operatore simile all'if-else
  - operatore condizionale (o operatore ternario)

L'istruzione if: sintassi

- La sintassi dell'istruzione *if* è la seguente

  if (⟨*condizione*⟩)

  ⟨*istruzione*⟩
- ⟨*condizione*⟩ è un'espressione di tipo intero, detta condizione dell'*if*
  - tipicamente essa sarà un predicato
- ⟨*istruzione*⟩ è una qualunque istruzione C detta corpo dell'*if*

L’istruzione if: semantica

if (⟨*condizione*⟩)

⟨*istruzione*⟩

- la ⟨*condizione*⟩ viene valutata
- se è vera (diversa da zero) viene eseguito il corpo dell’*if*
- se invece è falsa (uguale a zero) il corpo non viene eseguito

Istruzione if: esempio

- Vogliamo scrivere un programma che letti due numeri interi *a* e *b* dall'utente, stampa il minore dei due
- Se i numeri sono uguali, stamperà indifferentemente il valore di uno dei due

Istruzione if: esempio

```
#include <stdio.h>

int main(){
   int a, b;
   printf("inserisci due interi\n");
   scanf("%d%d", &a, &b);
   int min = a;
   if (b<a)
       min = b;
   printf("Il  minimo tra i due valori è %d\n", min);
}
```

La variabile *min* viene inizializzata con il valore di *a*

- Se *b* è più piccolo di *a*, il valore di *min* viene cambiato in *b*
- Altrimenti non accade nulla e *min* resta pari ad *a*

Istruzioni multiple nel corpo dell'if

- La sintassi dell'*if* impone che il corpo sia costituito da una sola istruzione
- È possibile eseguire più istruzioni nel corpo dell'*if*?
- Sì, bisogna racchiuderle tra parentesi graffe

Esempio

- Supponiamo di voler scrivere un programma che legge un intero *a* da tastiera e che, se *a* è pari lo divide per 2 e stampa il risultato; se *a* è dispari non stampa nulla.

Esempio

```
#include <stdio.h>

int main(){
   int a;
   printf("inserisci un intero\n");

   scanf("%d", &a);
   if (a%2 == 0){
       int b = a/2;
       printf("La metà del numero inserito è %d\n", b);
   }
}
```

- Il corpo dell'*if* è costituito da due istruzioni

Blocchi di istruzioni

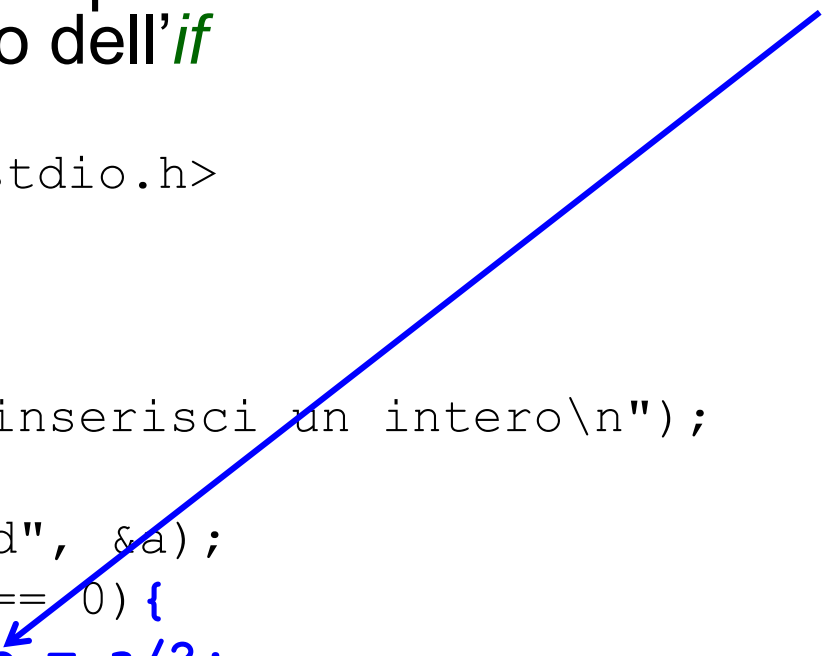
- Le due istruzioni racchiuse tra graffe dell'esempio precedente costituiscono un blocco di istruzioni
- Un blocco di istruzioni è una istruzione:
  - può essere utilizzato in qualunque punto ci si aspetta una istruzione
  - ad esempio nel corpo dell'istruzione *if*
- È possibile annidare blocchi cioè definire un blocco in un altro blocco

Blocchi e variabili

- Le variabili definite in un blocco sono visibili solo in quel blocco e in tutti quelli in esso annidati
- Non è possibile definire una variabile in un blocco se ne esiste una con lo stesso nome visibile in quel blocco

Blocchi e variabili

- Nel codice precedente la variabile *b* è visibile solo nel corpo dell'*if*



```
#include <stdio.h>

int main(){
   int a;
   printf("inserisci un intero\n");

   scanf("%d", &a);
   if (a%2 == 0){
       int b = a/2;
       printf("La metà del numero inserito è %d\n", b);
   }
}
```

Blocchi annidati e variabili

```
#include <stdio.h>
int main(){
   int a = 10;
        {// inizio blocco 1
          int b = 5;
          a += b;
               {// inizio blocco 2
                 int c = 2;
                 b += c;
                 a += c;
               }// fine blocco 2
          printf("%d",b);
        }// fine blocco 1
   printf("%d",a);
}
```

Blocchi annidati e variabili

```
#include <stdio.h>
int main(){
   int a = 10;
        {// inizio blocco 1
          int b = 5;
          a += b;
               {// inizio blocco 2
                 int c = 2;
                 b += c;
                 a += c;
               }// fine blocco 2
          printf("%d",b);
        }// fine blocco 1
   printf("%d",a);
}
```

blocco 1

Blocchi annidati e variabili

```
#include <stdio.h>
int main(){
   int a = 10;
        {// inizio blocco 1
          int b = 5;
          a += b;
               {// inizio blocco 2
                 int c = 2;
                 b += c;
                 a += c;
               }// fine blocco 2
          printf("%d",b);
        }// fine blocco 1
   printf("%d",a);
}
```

blocco 2

Blocchi annidati e variabili

```
#include <stdio.h>
int main(){
   int a = 10;
        {// inizio blocco 1
          int b = 5;
          a += b;
               {// inizio blocco 2
                 int c = 2;
                 b += c;
                 a += c;
               }// fine blocco 2
          printf("%d",b);
        }// fine blocco 1
   printf("%d",a);
}
```

visibilità di *a*

Blocchi annidati e variabili

```
#include <stdio.h>
int main(){
   int a = 10;
        {// inizio blocco 1
          int b = 5;
          a += b;
               {// inizio blocco 2
                 int c = 2;
                 b += c;
                 a += c;
               }// fine blocco 2
          printf("%d",b);
        }// fine blocco 1
   printf("%d",a);
}
```

visibilità di *b*

Blocchi annidati e variabili

```
#include <stdio.h>
int main(){
   int a = 10;
        {// inizio blocco 1
          int b = 5;
          a += b;
               {// inizio blocco 2
                 int c = 2;
                 b += c;
                 a += c;
               }// fine blocco 2
          printf("%d",b);
        }// fine blocco 1
   printf("%d",a);
}
```

visibilità di *c*

If annidati

- Poiché l'*if* è una istruzione C al pari di tutte le altre, è possibile che il corpo di una istruzione *if* sia un'altra istruzione *if*
- Si parla in questo caso di *if* annidati
- Esempio: modifichiamo il codice dell'esempio precedente in modo da stampare la metà di *a* solo nel caso in cui esso sia positivo

If annidati: esempio

```
#include <stdio.h>

int main(){
   int a;
   printf("inserisci un intero\n");
   scanf("%d", &a);
   if (a%2 == 0)
     if(a > 0){
       int b = a/2;
       printf("La metà del numero inserito è %d\n", b);
   }

}
```

- La seconda istruzione *if* verrà eseguita solo se la condizione della prima è vera

If annidati: esempio

```
#include <stdio.h>

int main(){
   int a;
   printf("inserisci un intero\n");
   scanf("%d", &a);
   if (a%2 == 0)
     if(a > 0){
       int b = a/2;
       printf("La metà del numero inserito è %d\n", b);
     }

}
```

- Nota: il secondo *if* non è racchiuso tra graffe
- Questa è infatti un'unica istruzione

L'istruzione if-else: sintassi

- La sintassi dell'istruzione *if-else* è la seguente

  if (⟨*condizione*⟩)

  ⟨*istruzione 1*⟩

  else

  ⟨*istruzione 2*⟩
- ⟨*condizione*⟩ è un'espressione di tipo intero, detta condizione dell'*if*
  - tipicamente essa sarà un predicato
- ⟨*istruzione 1*⟩ è una qualunque istruzione C (anche un blocco) detta corpo dell'*if*
- ⟨*istruzione 2*⟩ è una qualunque istruzione C (anche un blocco) detta corpo dell'*else*

L’istruzione if-else: semantica

if (⟨*condizione*⟩)

⟨*istruzione 1*⟩

else

⟨*istruzione 2*⟩

- la ⟨*condizione*⟩ viene valutata
- se è vera (diverso da zero) viene eseguito il corpo dell’*if*
- se è falsa (uguale a zero) viene eseguito il corpo dell’*else*

Istruzione if-else: esempio

- Vogliamo riscrivere il programma per stampare il minore tra due numeri utilizzando l'istruzione *if-else*
- Ricordiamo che il programma legge due numeri interi *a* e *b* da tastiera e stampa il minore dei due. Se i numeri sono uguali, stampa indifferentemente il valore di uno dei due

Istruzione if-else: esempio

```
#include <stdio.h>
int main(){
   int a, b;
   printf("inserisci due interi\n");
   scanf("%d%d", &a, &b);
   int min;
   if (b<a)
       min = b;
   else
       min = a;
   printf("Il  minimo tra i due valori è %d\n", min);
}
```

- Se *b* è più piccolo di *a*, il valore di *b* viene assegnato a *min*
- Altrimenti a *min* viene assegnato il valore di *a*

Codice alternativo

```
#include <stdio.h>
int main(){
   int a, b;
   printf("inserisci due interi\n");
   scanf("%d%d", &a, &b);
   int min;
   if (b<a)
       printf("Il  minimo tra i due valori è %d\n", b);
   else
       printf("Il  minimo tra i due valori è %d\n", a);
}
```

- Se *b* è più piccolo di *a*, viene stampato *b*
- Altrimenti viene stampato *a*

Cascate di if-else

- Scriviamo un programma che legge un intero *n* da tastiera:
  - decide se il numero è pari o dispari
  - se è pari stampa il risultato della divisione di *n* per 2
  - se è dispari verifica se *n* è divisibile per 3
  - se lo è stampa il risultato della divisione di *n* per 3
  - se non lo è verifica se *n* è divisibile per 5
  - se lo è stampa il risultato della divisione di *n* per 5
  - se non lo è stampa un messaggio opportuno

Cascate di if-else

```
#include <stdio.h>

int main(){
    int n;
    printf("Inserisci un intero\n");
    scanf("%d",&n);

    if (n%2==0){
          printf("Num. pari\n");
          printf("Quoziente %d/2: %d\n", n, n/2);
    }else
          if (n%3==0){
                    printf("Num. dispari divisibile per 3\n");
                    printf("Quoziente %d/3: %d\n", n, n/3);
          }else
                    if (n%5==0){
                        printf("Num. dispari divis. per 5 ma non per 3\n");
                        printf("Quoziente %d/5: %d\n", n, n/5);
                    }else
                    printf("Num. dispari non divis. per 3 o per 5\n");
}
```

Indentazione alternativa

```
#include <stdio.h>

int main(){
   int n;
   printf("Inserisci un intero\n");
   scanf("%d",&n);
   if (n%2==0){
        printf("Num. pari\n");
        printf("Quoziente %d/2: %d\n", n, n/2);
   }else if (n%3==0){
        printf("Num. dispari divisibile per 3\n");
        printf("Quoziente %d/3: %d\n", n, n/3);
   }else if (n%5==0){
        printf("Num. dispari divis. per 5 ma non per 3\n");
        printf("Quoziente %d/5: %d\n", n, n/5);
   }else
        printf("Num. dispari non divis. per 3 o per 5\n");
}
```

Altre istruzioni condizionali

- Il C mette a disposizione un'altra istruzione condizionale:
  - switch
- e un operatore simile all'if-else
  - operatore condizionale (o operatore ternario)

- Non discuteremo i dettagli di tali istruzioni

Istruzioni iterative

- Le istruzioni iterative consentono di ripetere più volte l'esecuzione di una sequenza di istruzioni all'interno di un metodo
- Un'istruzione iterativa è anche chiamata ciclo
- Il C mette a disposizione tre diversi tipi di istruzioni iterative:
  - while
  - do-while
  - for

L’istruzione while

- Supponiamo di voler scrivere un programma che, letto un intero *n* da tastiera, stampi tutti i numeri pari da *0* a *n*

Un’idea per la soluzione

- Generiamo tutti i numeri da *0* a *n* e per ognuno controlliamo se è pari. Partiamo da *i=0* e ripetiamo i seguenti passi:
  1. controlla se *i<n+1*: se no termina, altrimenti prosegui
  2. controlla se *i* è pari: se sì stampa *i*
  3. incrementa *i* di un’unità
  4. torna al passo 1
- Questa strategia può essere realizzata con l’istruzione *while*

L'istruzione while

- La sintassi dell'istruzione *while* è la seguente

  while (⟨*condizione*⟩)

  ⟨*istruzione*⟩

- ⟨*condizione*⟩ è un'espressione di tipo intero, detta condizione del *while*
  - tipicamente essa sarà un predicato
- ⟨*istruzione*⟩ è una qualunque istruzione C (anche un blocco) detta corpo del *while*

L’istruzione while: semantica

while (⟨*condizione*⟩)

⟨*istruzione*⟩

- la ⟨*condizione*⟩ viene valutata; se è falsa (pari a 0) il corpo non viene eseguito e si passa oltre
- se invece la ⟨*condizione*⟩ è vera (diversa da zero) si esegue il corpo del *while*
- al termine dell’esecuzione del corpo si torna a valutare la ⟨*condizione*⟩ e si itera il comportamento precedente

Esempio

```
#include <stdio.h>
int main(){
   int n;
   printf("Inserisci un intero\n");
   scanf("%d", &n);
   int i = 0;
   while (i<n+1){
        if (i%2==0)
              printf("%d\n", i);
        i++;
   }
}
```

- La variabile *i* viene inizializzata con il valore *0* e viene incrementata di una unità alla fine di ogni iterazione del ciclo *while*
- Si continua a ripetere il ciclo finché il valore di *i* si mantiene non superiore a *n*
- Quando *i* assume il valore *n+1*, il ciclo termina.

L’istruzione while: commenti

- È necessario fare in modo che la condizione del *while* prima o poi diventi falsa
- Se così non fosse, il corpo del ciclo verrebbe ripetuto all’infinito (loop infinito)
- Le istruzioni del corpo del *while* devono modificare la condizione
  - nell’esempio precedente l’istruzione *i*++ modifica il valore di *i* e fa sì che *i* diventi ad un certo punto pari a *n+1*

L’istruzione while: commenti

- Il corpo del *while* potrebbe anche non essere eseguito mai
- Ciò accade se la condizione è falsa la prima volta che viene valutata

Esempio: codice alternativo

```
#include <stdio.h>
int main(){
   int n;
   printf("Inserisci un intero\n");
   scanf("%d", &n);
   int i = 0;
   while (i<n+1){
        printf("%d\n", i);
        i+=2;
   }
}
```

- La variabile *i* viene inizializzata con il valore *0* e viene incrementata di due unità ad ogni iterazione del ciclo *while*
- In questo modo *i* assume soltanto i valori pari
- Si continua a ripetere il ciclo finché il valore di *i* si mantiene non superiore a *n*
- Quando *i* assume un valore superiore a *n*, il ciclo termina.

L'istruzione do-while: sintassi

- L'istruzione *do-while* è simile all'istruzione *while*
- La sintassi dell'istruzione *do-while* è la seguente

  do

  ⟨*istruzione*⟩

  while (⟨*condizione*⟩)
- ⟨*condizione*⟩ è un'espressione di tipo intero, detta condizione del *do-while*
  - tipicamente essa sarà un predicato
- ⟨*istruzione*⟩ è una qualunque istruzione C (anche un blocco) detta corpo del *do-while*

L’istruzione do-while: semantica

do

⟨*istruzione*⟩

while (⟨*condizione*⟩)

- viene eseguito il corpo una prima volta
- viene poi valutata la ⟨*condizione*⟩
- se è falsa (uguale a zero) l’esecuzione del *do-while* termina
- se invece la ⟨*condizione*⟩ è vera (diversa da zero) si torna ad eseguire il corpo e si itera il comportamento precedente

L’istruzione do-while: commenti

- Così come per il *while*, il corpo del *do-while* viene eseguito fintanto che la condizione rimane vera
- La differenza è che, nel caso del *do-while*, il corpo viene eseguito almeno una volta

Esempio

```
#include <stdio.h>
int main(){
   int n;
   printf("Inserisci un intero\n");
   scanf("%d", &n);
   int i = 0;
   do{
       printf("%d\n", i);
       i+=2;
   } while (i<n+1);
}
```

- Poiché il primo valore (cioè *0*) è pari, la prima iterazione deve essere eseguita

L’istruzione for: sintassi

- L’istruzione *for* è abbastanza diversa dalle due precedenti
- La sintassi dell’istruzione *for* è la seguente

  for (⟨*inizializzazione*⟩;⟨*condizione*⟩; ⟨*aggiornamento*⟩)

  ⟨*istruzione*⟩
- ⟨*inizializzazione*⟩ è un’espressione di qualunque tipo (anche l’invocazione di una funzione *void*):
  - quasi sempre è un’istruzione di assegnazione
- ⟨*condizione*⟩ è un’espressione di tipo intero
  - tipicamente essa sarà un predicato

L’istruzione for: sintassi

- L’istruzione *for* è abbastanza diversa dalle due precedenti
- La sintassi dell’istruzione *for* è la seguente

  for (⟨*inizializzazione*⟩;⟨*condizione*⟩; ⟨*aggiornamento*⟩)

  ⟨*istruzione*⟩
- ⟨*aggiornamento*⟩ è un’espressione di qualunque tipo (anche l’invocazione di una funzione *void*):
  - quasi sempre è un incremento o decremento di una variabile
- ⟨*istruzione*⟩ è una qualunque istruzione C (anche un blocco)

L’istruzione for: sintassi

- ⟨*inizializzazione*⟩, ⟨*condizione*⟩ e ⟨*aggiornamento*⟩ sono dette rispettivamente inizializzazione, condizione e aggiornamento del *for*
- ⟨*istruzione*⟩ è detta, come al solito, corpo del *for*

L’istruzione for: semantica

- Quando si esegue un’istruzione *for*, vengono eseguite le seguenti azioni:
  1. viene eseguita l’inizializzazione
  2. viene valutata la condizione; se essa è falsa (uguale a zero) l’esecuzione del *for* termina
  3. se la condizione è vera (diversa da zero) viene eseguito il corpo
  4. una volta eseguito il corpo viene eseguito l’aggiornamento e si torna al punto 2

L’istruzione for: commenti

- In base a quanto detto, l’inizializzazione viene eseguita una ed una sola volta in ogni caso, prima di qualunque iterazione
- dopo di ciò ogni iterazione consisterà di:
  - verifica condizione
  - esecuzione del corpo
  - aggiornamento

Esempio

```
#include <stdio.h>
int main(){
   int n;
   printf("Inserisci un intero\n");
   scanf("%d", &n);
   int i;
   for(i = 0; i<n+1; i+=2)
       printf("%d\n", i);
}
```

- La variabile *i* viene inizializzata a *0* prima di qualunque iterazione
- Ad ogni iterazione viene valutato se *i* è minore o uguale a *n*
- in caso positivo *i* viene stampata e incrementata di due

Esempio: commenti

- Il codice precedente è molto compatto
  - in particolare, poiché l'incremento della variabile *i* avviene con l'istruzione di aggiornamento, il corpo è costituito dalla sola istruzione di stampa
- L'istruzione *for* precedente può essere letta come: "*Per i che va da 0 a n, stampa il valore di i e poi incrementalo di 2 unità*"
- In effetti l'istruzione *for* viene tipicamente utilizzata per far assumere ad una variabile un numero discreto di valori in un intervallo fissato

Esempio: versione alternativa

```
#include <stdio.h>
int main(){
   int n;
   printf("Inserisci un intero\n");
   scanf("%d", &n);
   for(int i = 0; i<n+1; i+=2)
       printf("%d\n", i);
}
```

- In questo caso la variabile *i* viene definita all'interno del *for* (nell'inizializzazione)
- Così facendo la variabile *i* è visibile solo all'interno del ciclo *for*
  - nelle tre parti tra parentesi e nel corpo

Istruzione for: commenti

- Ciascuna delle tre parti del *for* è opzionale e può essere omessa
- Nelle parti di inizializzazione e aggiornamento è possibile scrivere un elenco di istruzioni separate da virgole
- Non entreremo nei dettagli di entrambi gli aspetti

Confronto tra le istruzioni iterative

- Abbiamo visto tre diversi tipi di istruzioni iterative...
- ...ed un problema che poteva essere risolto con tutte e tre
- Ci sono situazioni in cui è necessario l'uso di una specifica istruzione iterativa?

Equivalenza di istruzioni iterative

- L'istruzione *for* e l'istruzione *while* sono equivalenti

for (⟨*iniz.*⟩; ⟨*cond.*⟩; ⟨*aggior.*⟩)
    ⟨*istruzione*⟩

⟺

⟨*iniz.*⟩
while(⟨*cond.*⟩){
    ⟨*istruzione*⟩
    ⟨*aggior.*⟩
}

Equivalenza di istruzioni iterative

- Il *do-while* è meno flessibile delle altre due in quanto esegue il corpo almeno una volta
- Usando le istruzioni condizionali, è possibile superare tale vincolo

```
while (⟨condizione⟩)
    ⟨istruzione⟩
```

⇒

```
if (⟨condizione⟩){
    do
      ⟨istruzione⟩
    while(⟨condizione⟩)
}
```

```
do
   ⟨istruzione⟩
while (⟨condizione⟩)
```

⇒

```
⟨istruzione⟩
while(⟨condizione⟩)
    ⟨istruzione⟩
```

Equivalenza di istruzioni iterative

- In definitiva le tre istruzioni iterative sono equivalenti
- In linea di principio se ne potrebbe utilizzare una soltanto
- Tuttavia ci sono casi in cui usare un tipo di istruzione piuttosto che un’altra rende la scrittura del codice più semplice ed immediata
- La scelta del tipo di istruzione da utilizzare è questione di esperienza e gusto personale

Alcune indicazioni

- Vista la minor flessibilità del *do-while*, esso è meno usato delle altre due
  - si usa in quei casi in cui si vuole sfruttare la sua caratteristica di eseguire il corpo almeno una volta
- L'istruzione *for* viene preferita quando il numero di iterazioni da eseguire è noto (anche in forma parametrica) al momento della scrittura del codice
  - Ad esempio per far assumere ad una variabile una serie di valori in un intervallo noto (come nell'esempio visto)

Alcune indicazioni

- L'istruzione *while* viene preferita quando non è noto il numero di iterazioni all'atto della scrittura del codice
  - ad esempio il ciclo potrebbe essere ripetuto finché l'utente non immette un valore

Le istruzioni break e continue

- All'interno di un ciclo è possibile utilizzare le istruzioni *break* e *continue* per alterare la normale esecuzione del ciclo
- L'istruzione *break* causa la terminazione del ciclo più interno in cui si trova
- L'istruzione *continue* interrompe l'iterazione corrente e:
  - nel caso del *while* torna a valutare la condizione
  - nel caso del *for* esegue l'aggiornamento e poi torna a valutare la condizione

Istruzione break: esempio

```
#include <stdio.h>
int main(){
   int a;
   while(1){
        printf("inserisci un intero, 0 per finire \n");
        scanf("%d", &a);
        if (a == 0)
              break;
        printf("Numero letto: %d\n",a);
   }
}
```

- Il codice chiede ripetutamente all'utente di inserire un intero; se esso è 0 termina, altrimenti lo stampa
- Si noti che in questo caso il *break* interrompe un ciclo la cui condizione è sempre vera

Istruzione continue: esempio

```
#include <stdio.h>
int main(){
   printf("Inserisci un valore");
   int soglia;
   scanf("%d",&soglia);
   for (int i = 0; i<=100; i++){
       if (i<soglia)
              continue;
       printf("%d\n",i);
   }
}
```

- Il codice scandisce tutti i numeri da zero a 100 e stampa tutti quelli sopra la soglia inserita dall'utente

Break e continue: commenti

- Le istruzioni *break* e *continue* permettono di semplificare la scrittura del codice
- Tuttavia non sono indispensabili:
  - qualunque codice usi le istruzioni *break* e *continue* può essere riscritto senza di esse
- L'uso di tali istruzioni tende a destrutturare il codice rendendolo poco leggibile
- Per questo motivo si tende a farne un uso limitato o nullo