

Controllo nei programmi

Iterazione in C

- I programmi spesso utilizzano l'iterazione, cioè dei cicli, per eseguire ripetutamente un insieme di istruzioni finché una certa condizione è soddisfatta
- Nell'iterazione controllata da **contatore**, si utilizza una variabile per tenere traccia del numero di volte in cui il ciclo deve essere eseguito; il ciclo termina quando questa variabile raggiunge un valore predefinito.
- L'iterazione controllata da **sentinella** è utile quando non si conosce il numero esatto di iterazioni da effettuare; in questo caso, un valore speciale (sentinella) indica la fine dei dati da processare.
 - Il valore sentinella viene utilizzato per segnalare che non ci sono più dati da leggere e deve essere unico rispetto ai dati normali per evitare ambiguità.

Iterazioni con contatore

- L'iterazione controllata da contatore necessita di:
 - una **variabile** di controllo con un nome specifico
 - un **valore iniziale** assegnato alla variabile di controllo
 - un **incremento o un decremento** applicato alla variabile di controllo a ogni iterazione
 - una **condizione** che verifica il valore della variabile di controllo per stabilire se il ciclo deve proseguire o terminare

Iterazioni con contatore

```
1 // fig04_01.c
2 // Iterazione controllata da contatore.
3 #include <stdio.h>
4
5 int main(void) {
6     int counter = 1; // inizializzazione
7     while (counter <= 5) { // condizione di iterazione
8         printf("%d ", counter);
9         ++counter; // incremento
10    }
11    puts("");
12 }
```

1 2 3 4 5

Fonte: Deitel & Deitel

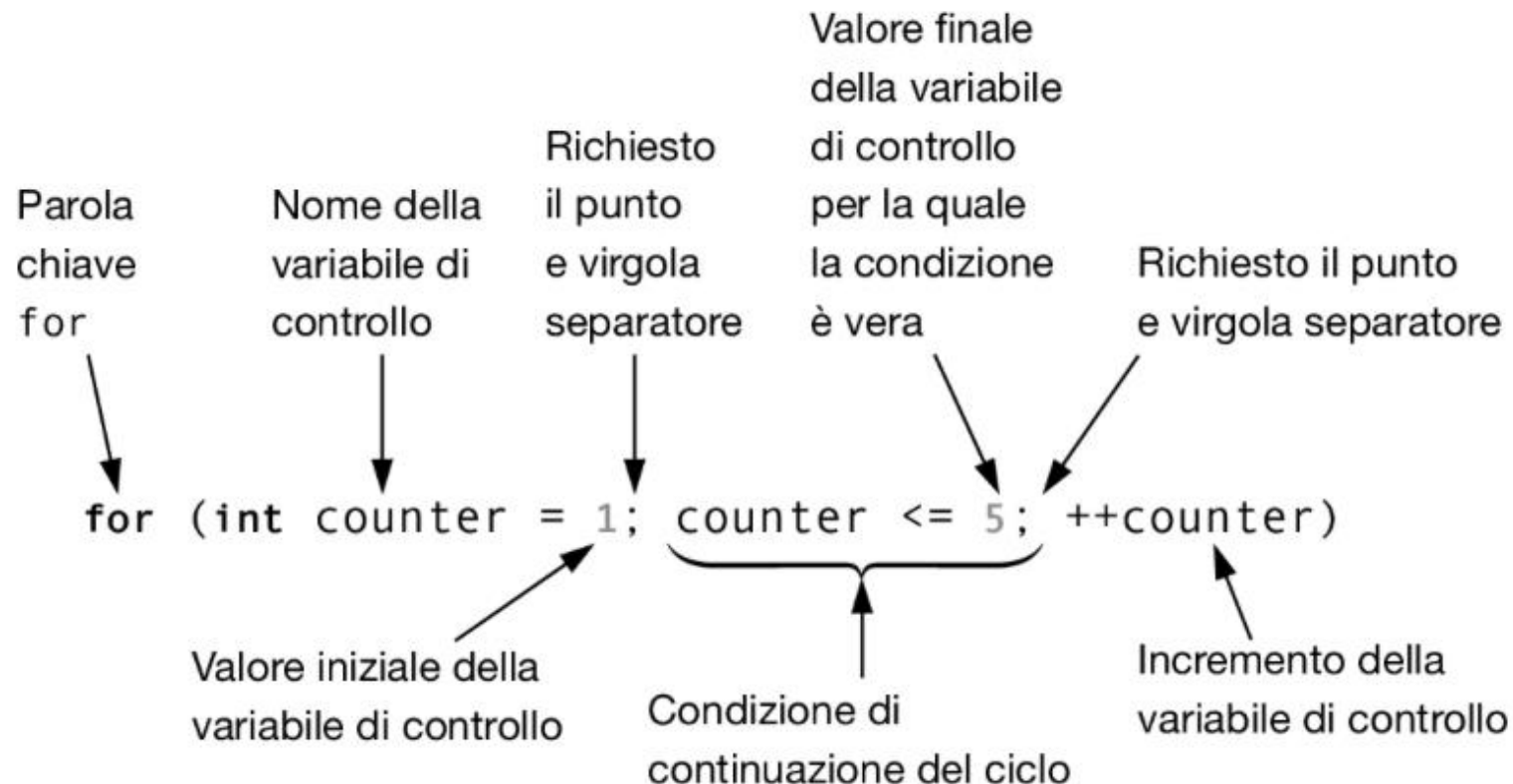
- I valori in virgola mobile possono essere approssimati e portare a test imprecisi
- Sempre controllare i cicli usando valori numerici interi

Iterazioni con istruzione for

```
1  // fig04_02.c
2  // Iterazione controllata da contatore con l'istruzione for.
3  #include <stdio.h>
4
5  int main(void) {
6      // inizializzazione, condizione dell'iterazione e incremento
7      // sono tutti inclusi nell'intestazione dell'istruzione for.
8      for ( int counter = 1 ; counter <= 5 ; ++counter) {
9          printf("%d ", counter);
10     }
11     puts(""); // stampa un newline
12 }
```

```
1 2 3 4 5
```

Iterazioni con istruzione for



Osservazioni sul `for`

- La variabile di controllo esiste solo fino al termine del ciclo
- Provare ad accedere al di fuori del ciclo (dopo la parentesi di chiusura) è un errore di compilazione
- Attenzione a usare `<=` o `<` nella condizione (errori di tipo off-by-one)
 - Per stampare i valori da 1 a 5, per esempio, la condizione di continuazione del ciclo può essere `counter <= 5` oppure `counter < 6`

Formato dell'istruzione `for`

```
for (inizializzazione; condizioneDiContinuazioneDelCiclo; incremento) {  
    istruzione  
}
```

- **inizializzazione**: definisce la variabile di controllo del ciclo e ne assegna il valore iniziale
- **condizioneDiContinuazioneDelCiclo**: stabilisce se il ciclo deve continuare a essere eseguito
- **incremento**: aggiorna il valore della variabile di controllo dopo ogni iterazione, in modo che la condizione del ciclo diventi falsa al momento opportuno
- I due punti e virgola nell'intestazione del `for` sono obbligatori.
- Se la condizione di continuazione del ciclo è falsa all'inizio, il corpo del `for` non viene eseguito e il programma passa direttamente all'istruzione successiva.

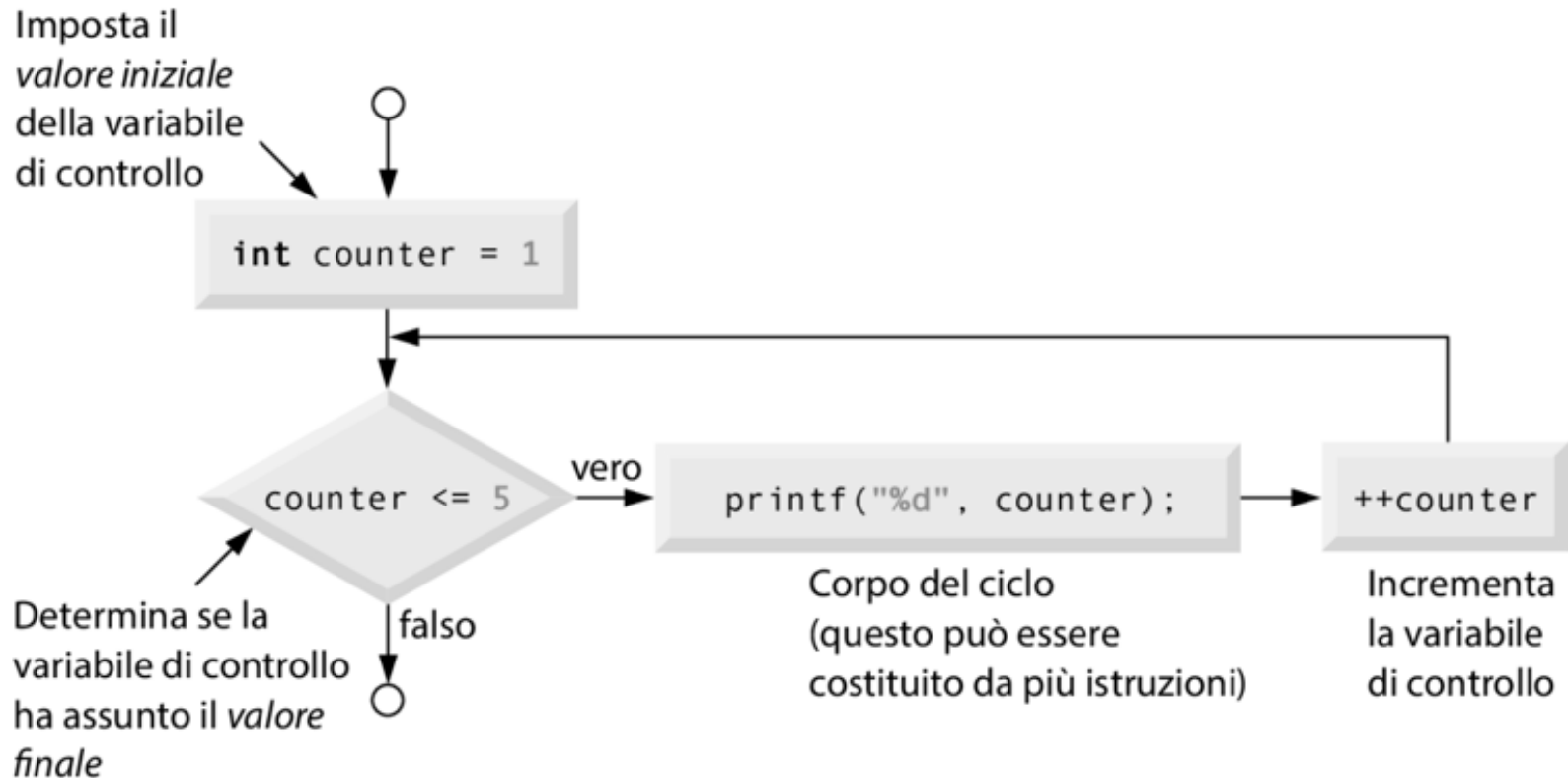
Cicli infiniti

- Un ciclo infinito si verifica quando la condizione per continuare il ciclo non diventa **mai falsa**
- Per evitare cicli infiniti:
 - Assicurarsi che non ci sia un punto e virgola subito dopo l'istestazione di un'istruzione `while`
 - In un ciclo controllato da **contatore**, garantire che la variabile di controllo venga incrementata o decrementata, in modo che la condizione del ciclo diventi falsa
 - In un ciclo controllato da **sentinella**, assicurarsi che il valore sentinella venga inserito alla fine

Modificare la variabile di controllo

- L'espressione di incremento nell'istruzione `for` in C funziona come **un'istruzione separata** eseguita alla fine del corpo del ciclo
- Pertanto, le seguenti espressioni sono equivalenti se utilizzate come incremento in un'istruzione `for`:
 - `counter = counter + 1`
 - `counter += 1`
 - `++counter`
 - `counter++`
- L'incremento nell'espressione può anche essere **negativo** (in questo caso diventa un decremento e il ciclo procede all'indietro)
- Meglio non modificare la variabile di controllo internamente!


Diagramma di flusso



Esempi (credits to chatGPT)

1. Ciclo per stampare numeri da 1 a 10

c


 Copia codice

```
for (int i = 1; i <= 10; i++) {  
    printf("%d\n", i);  
}
```

Questo ciclo stampa i numeri da 1 a 10 incrementando la variabile `i` ad ogni iterazione.

2. Ciclo per calcolare la somma dei primi N numeri naturali

c

 Copia codice


```
int N = 5;  
int somma = 0;  
  
for (int i = 1; i <= N; i++) {  
    somma += i;  
}  
  
printf("La somma dei primi %d numeri naturali è %d\n", N, somma);
```

Questo ciclo calcola la somma dei primi N numeri naturali.

Esempi (credits to chatGPT)

3. Ciclo per contare all'indietro da 10 a 1

c


 Copia codice

```
for (int i = 10; i >= 1; i--) {  
    printf("%d\n", i);  
}
```

Questo ciclo stampa i numeri da 10 a 1 decrementando la variabile `i` ad ogni iterazione.

4. Ciclo per stampare solo i numeri pari da 1 a 20

c

 Copia codice


```
for (int i = 2; i <= 20; i += 2) {  
    printf("%d\n", i);  
}
```

Questo ciclo stampa solo i numeri pari da 1 a 20 incrementando `i` di 2 ad ogni iterazione.

Esempi (credits to chatGPT)

5. Ciclo senza condizione di continuazione (ciclo infinito)

c

 Copia codice

```
for (int i = 0; ; i++) {  
    printf("Questo è un ciclo infinito: %d\n", i);  
    if (i == 10) {  
        break; // Rompe il ciclo dopo 10 iterazioni  
    }  
}
```

Questo è un ciclo infinito, che si interrompe quando `i` raggiunge 10 grazie all'istruzione `break`.

Esercizio: interesse composto

Una persona investe \$1000,00 in un conto corrente che frutta il 5% di interesse. Supponendo che l'intero interesse resti depositato nel conto, calcolate e stampate la quantità di denaro nel conto alla fine di ogni anno per 10 anni. Usate la seguente formula per determinare queste quantità:

$$a = p(1 + r)^n$$

Dove p è la quantità iniziale di denaro investita (cioè, il capitale, che qui è \$1000.00), r è il tasso annuale di interesse (per esempio, 0.05 per 5%), n è il numero degli anni, che qui è 10, e a è la quantità di denaro in deposito alla fine dell'anno n .

Esercizio: interesse composto

```
1 // fig04_04.c
2 // Calcolo dell'interesse composto.
3 #include <stdio.h>
4 #include <math.h>
5
6 int main(void) {
7     double principal = 1000.0; // capitale iniziale
8     double rate = 0.05; // tasso di interesse annuale
9
10    // stampa le intestazioni delle colonne della tabella
11    printf("%4s%21s\n", "Year", "Amount on deposit");
12
13    // calcola la quantità in deposito per ognuno dei dieci anni
14    for (int year = 1; year <= 10; ++year) {
15
16        // calcola la nuova quantità per l'anno specificato
17        double amount = principal * pow( 1.0 + rate, year);
18
19        // stampa una riga della tabella
20        printf("%4d%21.2f\n", year, amount);
21    }
22 }
```

Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.63
4	1215.51
5	1276.28
6	1340.10
7	1407.10
8	1477.46
9	1551.33
10	1628.89

- C non ha una funzione esponenziale
- È necessario includere l'intestazione `<math.h>` per utilizzare la funzione `pow`
- Senza questa intestazione, il programma non funzionerebbe correttamente perché il **linker** non troverebbe la funzione `pow`

Funzione pow

- La funzione `pow` accetta due argomenti di tipo `double`, mentre la variabile `year` è di tipo `int`
- Il file `math.h` contiene informazioni che indicano al compilatore di **convertire** la variabile `year` in un tipo `double` temporaneo prima di chiamare la funzione `pow`
- Queste informazioni sono fornite nel **prototipo** della funzione `pow`

Syntax

One of the following:

```
pow(double base, double exponent);
```

Parameter Values

Parameter	Description
<i>base</i>	Required. The base of the power operation.
<i>exponent</i>	Required. The exponent of the power operation.

Technical Details

Returns: A `double` value representing the result of a power operation.

Fonte: W3Schools

Funzioni di `<math.h>`

C Math Functions

The `<math.h>` library has many functions that allow you to perform mathematical tasks on numbers.

Function	Description
<u>acos(x)</u>	Returns the arccosine of x, in radians
<u>acosh(x)</u>	Returns the hyperbolic arccosine of x
<u>asin(x)</u>	Returns the arcsine of x, in radians
<u>asinh(x)</u>	Returns the hyperbolic arcsine of x
<u>atan(x)</u>	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
<u>atan2(y, x)</u>	Returns the angle theta from the conversion of rectangular coordinates (x, y) to polar coordinates (r, theta)
<u>atanh(x)</u>	Returns the hyperbolic arctangent of x
<u>cbrt(x)</u>	Returns the cube root of x
<u>ceil(x)</u>	Returns the value of x rounded up to its nearest integer
<code>copysign(x, y)</code>	Returns the first floating point x with the sign of the second floating point y
<u>cos(x)</u>	Returns the cosine of x (x is in radians)

Stampare output numerico

- La specifica di conversione `%21.2f` è utilizzata per stampare il valore della variabile `amount`
 - Il numero 21 indica la **larghezza** del campo, specificando che il valore sarà stampato in 21 posizioni di carattere
 - La parte `.2` specifica la **precisione**, ossia il numero di cifre decimali da visualizzare
- Se il numero di caratteri stampati è inferiore alla larghezza del campo, il valore sarà allineato a destra con spazi vuoti all'inizio
- Più dettagli sulla formattazione dell'output nelle prossime lezioni

```
14         // stampa una riga della tabella
15         printf("%4d%21.2f\n", year, amount);
16     }
17 }
```

Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.63
4	1215.51
5	1276.28
6	1340.10
7	1407.10
8	1477.46
9	1551.33
10	1628.89

Fonte: Deitel & Deitel

Precisione e requisiti di memoria dei numeri in virgola mobile

- Il tipo `float` richiede generalmente **quattro byte di memoria** e offre circa *7 cifre significative*
- Il tipo `double` richiede solitamente **otto byte di memoria** e fornisce circa *15 cifre significative*, offrendo quindi il **doppio** della precisione rispetto ai `float`
- In C, i valori in virgola mobile sono trattati come **tipo double di default**: "letterali in virgola mobile" (floating-point)
- C include anche il tipo `long double`, che generalmente è memorizzato in **12 o 16 byte di memoria**
- Lo [standard del C](#) stabilisce le dimensioni minime per ciascun tipo

Tipi di variabili in C

Type specifier	Equivalent type	Width in bits by data model				
		C standard	LP32	ILP32	LLP64	LP64
char	char	at least 8	8	8	8	8
signed char	signed char					
unsigned char	unsigned char					
short	short int	at least 16	16	16	16	16
short int						
signed short						
signed short int						
unsigned short						
unsigned short int						
int	int	at least 16	16	32	32	32
signed						
signed int						
unsigned	unsigned int	at least 32	32	32	32	64
unsigned int						
long	long int					
long int						
signed long						
signed long int						
unsigned long	unsigned long int	at least 64	64	64	64	
unsigned long int						
long long	long long int (C99)					
long long int						
signed long long						
signed long long int						
unsigned long long	unsigned long long int (C99)					
unsigned long long int						

Errori di rappresentazione

- In aritmetica convenzionale, i numeri in virgola mobile spesso derivano da divisioni, come nel caso di $10 / 3$, che produce un risultato di 3,3333333... con la sequenza di 3 che si ripete all'infinito
- I computer allocano solo una **quantità fissa di spazio** per memorizzare i valori in virgola mobile, quindi il valore memorizzato è solo **un'approssimazione**
- I numeri in virgola mobile sono soggetti a un **errore di rappresentazione**, il che significa che non possono essere rappresentati esattamente
 - ad esempio, usarli per confronti di uguaglianza può portare a risultati errati
- I numeri in virgola mobile sono ampiamente usati in applicazioni che richiedono valori di misura

Errori di rappresentazione: esempio

- Supponiamo di avere due importi: 14.234 (che viene arrotondato a 14.23 per la stampa) e 18.673 (arrotondato a 18.67)
- La somma di questi importi produce 32.907, che viene arrotondata a 32.91 per la stampa
- Il risultato della stampa sarà: $14.23 + 18.67 = 32.91$
- Tuttavia, se si sommano i valori stampati ($14.23 + 18.67$), il risultato è 32.90 **invece di 32.91!**
- Questo esempio illustra come i numeri in virgola mobile possano portare a discrepanze nei risultati a causa di problemi di precisione

Intervallo



Fonte: PlaygroundAI

Selezione multipla con `switch`

- Abbiamo già visto le istruzioni di selezione singola `if` e di selezione doppia `if-else`
- Quando un algoritmo richiede di testare una variabile o espressione contro più valori interi, e intraprendere azioni diverse per ciascuno di essi, si utilizza una **selezione multipla**
- Il C offre l'istruzione di selezione multipla `switch` per gestire questo tipo di processo decisionale
- L'istruzione `switch` è composta da:
 - Una serie di etichette `case`, ciascuna associata a un valore specifico
 - Un **caso default opzionale**, che gestisce i valori che non corrispondono a nessuna delle etichette `case`
 - **Istruzioni** da eseguire per ciascuna etichetta `case`

Selezione multipla con switch

```
1 // fig04_05.c
2 // Conteggio di voti a lettera con switch.
3 #include <stdio.h>
4
5 int main(void) {
6     int aCount = 0;
7     int bCount = 0;
8     int cCount = 0;
9     int dCount = 0;
10    int fCount = 0;
11
12    puts("Enter the letter grades.");
13    puts("Enter the EOF character to end input.");
14    int grade = 0; // un voto
15
16    // ripeti finche' l'utente non immette la sequenza di end-of-file
17    while ((grade = getchar()) != EOF) {
18
19        // determina quale voto e' stato inserito
20        switch (grade) { // switch annidato nel while
21            case 'A': // il voto e' la lettera maiuscola A
22            case 'a': // o la lettera minuscola a
23                ++aCount;
24                break; // necessario per uscire dallo switch
25            case 'B': // il voto e' la lettera maiuscola B
26            case 'b': // o la lettera minuscola b
27                ++bCount;
28                break;
29            case 'C': // il voto e' la lettera maiuscola C
30            case 'c': // o la lettera minuscola c
31                ++cCount;
32                break;
33            case 'D': // il voto e' la lettera maiuscola D
34            case 'd': // o la lettera minuscola d
35                ++dCount;
36                break;
37            case 'F': // il voto e' la lettera maiuscola F
38            case 'f': // o la lettera minuscola f
39                ++fCount;
40                break;
41            case '\n': // ignora i newline,
42            case '\t': // le tabulazioni
43            case ' ': // e gli spazi in input
44                break;
45            default: // cattura tutti gli altri caratteri
46                printf("%s", "Incorrect letter grade entered.");
47                puts(" Enter a new grade.");
48                break; // opzionale; uscirà comunque dallo switch
49        } // fine di switch
50    } // fine di while
51}
```

```
47 // stampa il riepilogo dei risultati
48 puts("\nTotals for each letter grade are:");
49 printf("A: %d\n", aCount);
50 printf("B: %d\n", bCount);
51 printf("C: %d\n", cCount);
52 printf("D: %d\n", dCount);
53 printf("F: %d\n", fCount);
54 }
```

Enter the letter grades.
Enter the EOF character to end input.

a

b

c

C

A

d

f

C

E

Incorrect letter grade entered. Enter a new grade.

D

A

b

^Z _____ Non tutti i sistemi mostrano una rappresentazione del carattere EOF

Totals for each letter grade are:

A: 3

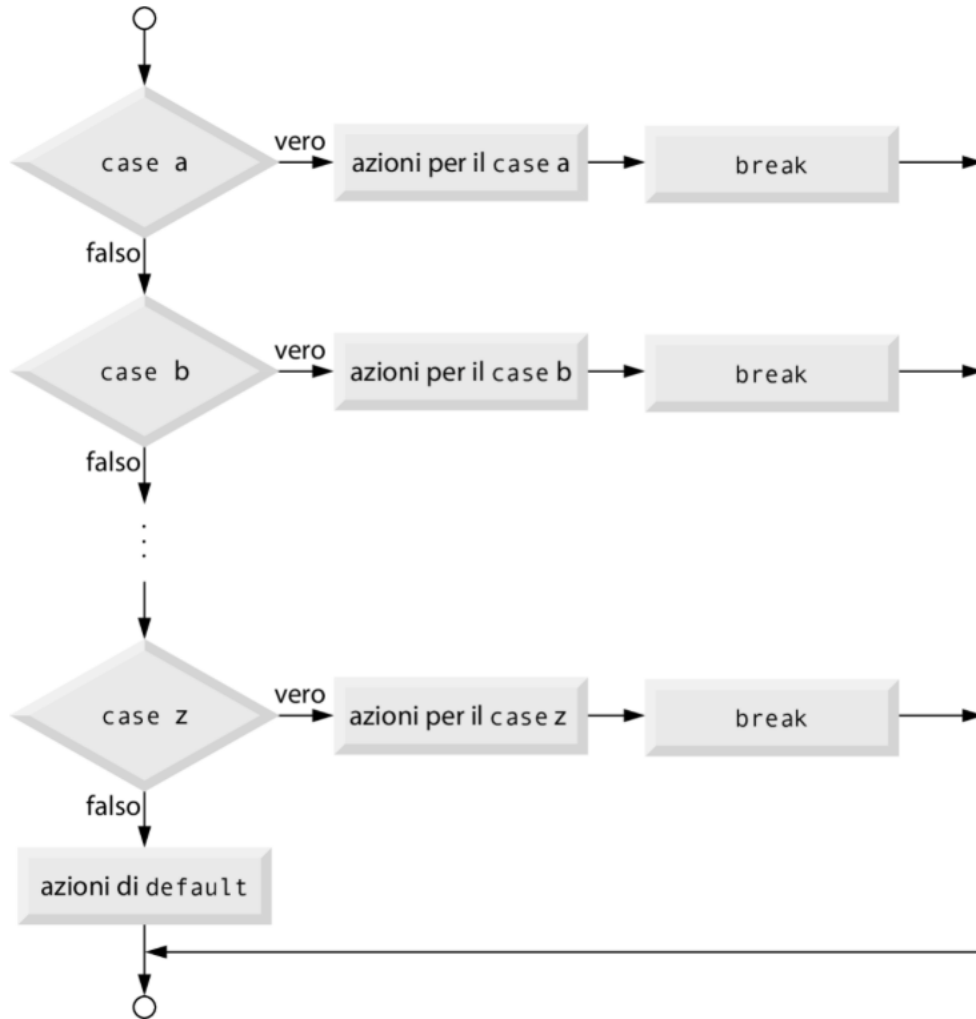
B: 2

C: 3

D: 2

F: 1

Diagramma di flusso



Dettagli dell'istruzione `switch`

- Utilizziamo l'istruzione `break` perché i vari case in un'istruzione `switch` verrebbero altrimenti eseguiti tutti insieme
- In mancanza di istruzioni `break`, ogni volta che si verificherà un confronto positivo verranno eseguite le istruzioni per tutti i case rimanenti
- Dimenticare un'istruzione `break` quando è necessaria in un'istruzione `switch` è un **errore logico**

Caso default

- È buona pratica includere sempre un caso **default** per gestire valori non esplicitamente testati, evitando che vengano ignorati
- Il caso default aiuta a gestire condizioni eccezionali e a garantire che tutte le possibilità siano considerate
- Le clausole case e default possono essere ordinate in qualsiasi modo all'interno di uno `switch`
- Tuttavia, è comune posizionare la clausola default per ultima
- Quando la clausola default è ultima, l'istruzione `break` finale non è necessaria
- Molti programmatori includono comunque il `break` per chiarezza e simmetria con gli altri casi

Leggere caratteri in input

- Nel programma, l'utente inserisce i voti a lettera degli studenti
- Nell'intestazione del ciclo `while`, l'assegnazione `grade = getchar()`
- viene eseguita prima del controllo della condizione:
`while ((grade = getchar()) != EOF)`
- La funzione `getchar()` legge un carattere dalla tastiera e lo memorizza nella variabile `grade`, che è di tipo `int`
- I caratteri sono normalmente memorizzati in variabili di tipo `char`, ma in C possono essere memorizzati in variabili di tipo intero poiché sono rappresentati come interi di un byte
- La funzione `getchar()` restituisce il carattere come `int`, permettendo di trattarlo sia come carattere che come valore intero

Caratteri e interi

- L'istruzione

```
printf("The character (%c) has the value %d.\n", 'a', 'a');
```

- stampa

```
The character (a) has the value 97.
```

- utilizzando le specifiche di conversione %c e %d per stampare il carattere 'a' e il suo valore intero.
- È possibile leggere i caratteri con `scanf` utilizzando la specifica di conversione %c
- Il numero intero 97 rappresenta il valore numerico del carattere 'a' nel computer

Assegnazioni come valori

- L'intera operazione di assegnazione ha un valore: ad esempio, l'espressione `grade = getchar()` restituisce il carattere letto da `getchar()` e lo assegna alla variabile `grade`
- Le assegnazioni possono essere usate per assegnare lo stesso valore a più variabili. Ad esempio:
 - `a = b = c = 0;`
 - Prima viene eseguita `c = 0`, poi `b` riceve il valore di `c = 0` (che è 0), e infine `a` riceve il valore di `b = (c = 0)` (anche questo è 0)
- Il valore dell'assegnazione viene confrontato con `EOF`, che indica la fine del file
- `EOF` è una costante simbolica definita in `<stdio.h>`, e normalmente ha il valore -1
- Viene usato come valore sentinella per indicare "non ci sono più dati da inserire"
- Se il valore assegnato a `grade` è uguale a `EOF`, il programma termina

Valore EOF

- I caratteri sono rappresentati con il tipo `int` nel programma perché `EOF` è un valore intero
- Utilizzare `EOF` al posto di `-1` rende i programmi più portabili
- Il C standard stabilisce che `EOF` è un valore intero negativo, ma non necessariamente `-1`. I valori di `EOF` possono variare a seconda del sistema
- La combinazione di tasti per inserire EOF (**end-of-file**) è dipendente dal sistema.
- Sui sistemi Linux/UNIX/macOS, l'indicatore EOF si inserisce digitando su una riga separata `Ctrl + d`
- Su altri sistemi, come Windows di Microsoft, l'indicatore EOF può essere inserito digitando `Ctrl + z` (seguito da `Invio`)

Ignorare i caratteri di formattazione

```
37     case '\n' : // ignora i newline,
38     case '\t' : // le tabulazioni
39     case ' ' : // e gli spazi in input
40         break ;
41     default : // cattura tutti gli altri caratteri
42         printf("%s", "Incorrect letter grade entered.");
43         puts(" Enter a new grade.");
44         break ; // opzionale; uscirà comunque dallo switch
45 } // fine di switch
```

- Il programma deve saltare caratteri di formattazione per garantire il corretto funzionamento del programma
- Premere `Invio` per inviare l'input inserisce un carattere `newline` nell'input
- Le istruzioni `case` precedenti nell'istruzione `switch` impediscono che il messaggio di errore nel caso **default** venga stampato ogni volta
- Ogni input causa **due iterazioni** del ciclo: una per il voto a lettera e una per il carattere `newline` (`\n`)

Espressioni costanti integrali

- Ogni case in un'istruzione switch può testare solo un'espressione costante integrale
- Un'espressione costante integrale è una combinazione di costanti di tipo carattere e costanti intere che rappresentano valori costanti interi
- Le costanti di tipo carattere sono rappresentate come un singolo carattere tra virgolette singole, ad esempio 'A'
- I caratteri devono essere racchiusi tra virgolette singole per essere riconosciuti come costanti di tipo carattere
 - Doppia virgoletta = stringhe
- Le costanti intere sono semplici valori numerici interi
- Nell'esempio discusso, abbiamo utilizzato costanti di tipo carattere per i case

Iterazione con do-while

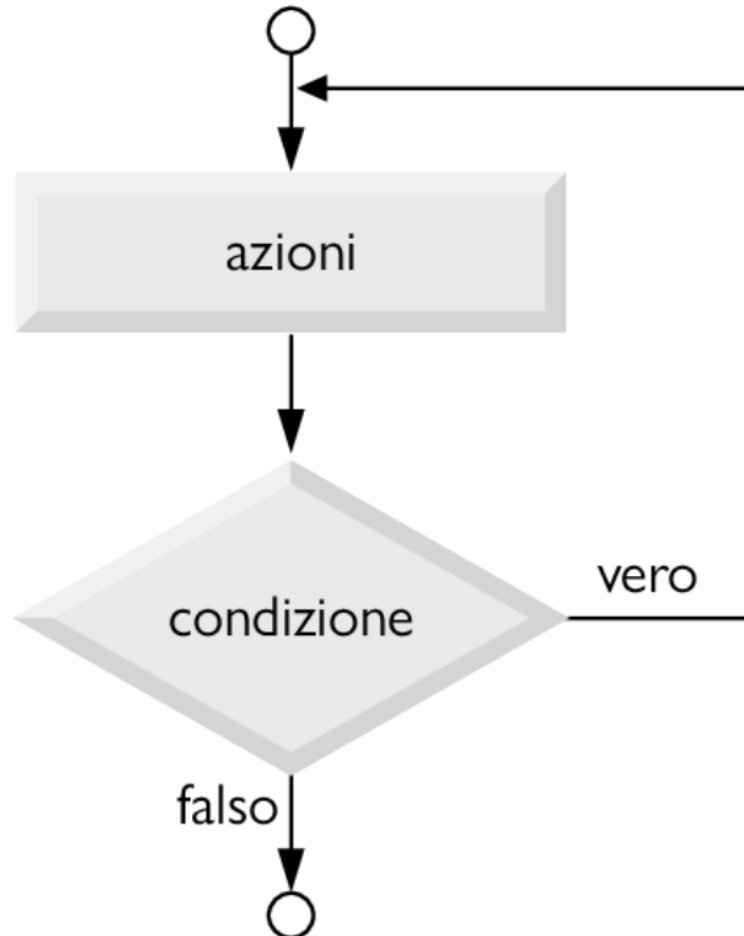
- Il ciclo viene eseguito **almeno una volta**

```
1 // fig04_06.c
2 // Uso dell'istruzione di iterazione do...while.
3 #include <stdio.h>
4
5 int main(void) {
6     int counter = 1; // inizializza il contatore
7     do {
8         printf( "%d ", counter);
9     } while ( ++counter <= 5 );
10 }
```

1 2 3 4 5

Fonte: Deitel & Deitel

Diagramma di flusso



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

- Istruzione **break**:
 - Termina prematuramente un ciclo o un'istruzione switch
 - Quando break viene eseguito, il controllo del programma esce immediatamente, saltando il resto del corpo del ciclo o dei case
 - Utilizzato per uscire da un ciclo quando una certa condizione è soddisfatta, o per terminare un'istruzione switch dopo aver gestito un caso specifico
- Istruzione **continue**:
 - Salta il resto del corpo del ciclo corrente e passa direttamente all'iterazione successiva del ciclo
 - Quando continue viene eseguito, il controllo del programma salta le istruzioni rimanenti nel ciclo e torna all'inizio del ciclo per la prossima iterazione
 - Utilizzato per saltare l'esecuzione di alcune istruzioni all'interno del ciclo basandosi su una condizione, e continuare con la prossima iterazione del ciclo

Esempio di break

```
1 // fig04_07.c
2 // Uso dell'istruzione break in un'istruzione for.
3 #include <stdio.h>
4
5 int main(void) {
6     int x = 1; // dichiarato qui per un utilizzo dopo il ciclo
7
8     // ripeti 10 volte
9     for (; x <= 10 ; ++x) {
10         // se x e' 5, termina il ciclo
11         if (x == 5) {
12             break ; // interrompi il ciclo solo se x e' 5
13         }
14
15         printf("%d ", x);
16
17         printf("\nBroke out of loop at x == %d\n", x);
18     }
```

```
1 2 3 4
Broke out of loop at x == 5
```

Esempio di continue

```
1  // fig04_08.c
2  // Uso dell'istruzione continue in un'istruzione for.
3  #include <stdio.h>
4
5  int main(void) {
6      // ripeti 10 volte
7      for (int x = 1; x <= 10; ++x) {
8          // se x e' 5, continua con la successiva iterazione del ciclo
9          if (x == 5) {
10             continue ; // salta il restante codice nel corpo del ciclo
11         }
12         printf("%d ", x);
13     }
14     puts("\nUsed continue to skip printing the value 5");
15 }
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```


Note su `break` e `continue`

- Alcuni programmatori evitano `break` e `continue` perché ritengono che **violino** le norme di programmazione strutturata
- Queste istruzioni possono essere sostituite con tecniche di programmazione strutturata, ma `break` e `continue` sono generalmente più veloci
- Migliorare le prestazioni può talvolta compromettere la leggibilità e la manutenibilità del codice, e viceversa.
- In situazioni che non richiedono prestazioni elevate, concentratevi innanzitutto sulla semplicità e correttezza del codice
- Ottimizzate il codice per la velocità e la compattezza solo se necessario, dopo aver assicurato che sia semplice e corretto

Esempio di conversione di `break` (credits to ChatGPT)

```
#include <stdio.h>

int main() {
    int number;
    int found = 0;

    while (1) { // Ciclo infinito
        printf("Enter a number (0 to stop): ");
        scanf("%d", &number);

        if (number == 0) {
            break; // Uscita anticipata quando il numero è 0
        }

        printf("You entered: %d\n", number);
    }

    printf("Loop terminated.\n");
    return 0;
}
```

`break`

```
#include <stdio.h>

int main() {
    int number;
    int should_continue = 1; // Variabile per controllare il ciclo

    while (should_continue) {
        printf("Enter a number (0 to stop): ");
        scanf("%d", &number);

        if (number == 0) {
            should_continue = 0; // Imposta la variabile per terminare il ciclo
        } else {
            printf("You entered: %d\n", number);
        }
    }

    printf("Loop terminated.\n");
    return 0;
}
```

`if-else`

Operatori logici in C

- **AND logico (&&):**

- Restituisce true (vero) solo se entrambe le espressioni operanti sono vere
- Esempio: `if (a > 0 && b < 10)` verifica che a sia maggiore di 0 e b sia minore di 10

- **OR logico (||):**

- Restituisce true se almeno una delle espressioni operanti è vera
- Esempio: `if (a > 0 || b < 10)` verifica che a sia maggiore di 0 o b sia minore di 10

- **NOT logico (!):**

- Inverte il valore logico dell'espressione operante. Se l'espressione è vera, restituisce false, e viceversa
- Esempio: `if (!(a > 0))` verifica se a non è maggiore di 0 (cioè, se a è minore o uguale a 0)

Tabella di verità di &&

espressione1	espressione2	espressione1 && espressione2
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

Tabella di verità di ||

espressione1	espressione2	espressione1 espressione2
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

Tabella di verità di !

- Nella maggior parte dei casi si può evitare di usarlo riscrivendo la condizione.
- `if (! (grade == sentinelValue))`
- **è equivalente a**
- `if (grade != sentinelValue)`

espressione	! espressione
0	1
nonzero	0

Operatori logici in C

- Gli operatori logici hanno una **precedenza inferiore** rispetto agli operatori aritmetici e relazionali, ma **superiore** agli operatori di assegnazione.
 - `a > 0 && b < 10 || c == 5`
 - l'operatore `&&` viene valutato prima di `||`
- Gli operatori logici `&&` e `||` usano la valutazione **cortocircuitata** (short-circuit)
 - ciò significa che il secondo operando non viene valutato se il risultato può essere determinato dal primo operando.
 - `if (a == 0 || (b / a > 1))`
 - se `a` è 0, il secondo operando `(b / a > 1)` non viene valutato per evitare la divisione per zero.
- Gli operatori logici sono comunemente utilizzati per combinare espressioni booleane e per controllare flussi condizionali nel codice

I tipi booleani in C

- Il tipo `_Bool` è il tipo di dato booleano standard del C, che può assumere solo due valori: 0 (falso) e 1(vero)
- In una condizione, 0 è considerato come falso, mentre qualsiasi valore diverso da zero è considerato vero
- Aggiungere la libreria `<stdbool.h>` include definizioni per `bool`, `true` e `false`.
 - `bool` è un'abbreviazione per `_Bool`, `true` è equivalente a 1, e `false` è equivalente a 0.
 - Durante la preelaborazione, gli identificatori `bool`, `true` e `false` vengono sostituiti rispettivamente con `_Bool`, 1, e 0

Associatività e precedenza operatori in C

Operatori	Associatività	Tipo
<code>++</code> (postfisso) <code>--</code> (postfisso)	da destra a sinistra	postfisso
<code>+</code> <code>-</code> <code>!</code> <code>++</code> (prefisso) <code>--</code> (prefisso) <code>(tipo)</code>	da destra a sinistra	unario
<code>*</code> <code>/</code> <code>%</code>	da sinistra a destra	moltiplicativo
<code>+</code> <code>-</code>	da sinistra a destra	additivo
<code><</code> <code><=</code> <code>></code> <code>>=</code>	da sinistra a destra	relazionale
<code>==</code> <code>!=</code>	da sinistra a destra	di uguaglianza
<code>&&</code>	da sinistra a destra	AND logico
<code> </code>	da sinistra a destra	OR logico
<code>?:</code>	da destra a sinistra	condizionale
<code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code>	da destra a sinistra	di assegnazione
<code>,</code>	da sinistra a destra	virgola

Confondere uguaglianza e assegnazione

- Gli operatori `==` (uguaglianza) e `=` (assegnazione) sono frequentemente scambiati per errore
- L'uso di `=` in una condizione `if` può causare l'assegnazione di un valore alla variabile e una condizione sempre vera, generando **errori logici** senza errori di compilazione
- Esempio:
 - **Corretto:** `if (payCode == 4)` verifica se `payCode` è uguale a 4.
 - **Errato:** `if (payCode = 4)` assegna 4 a `payCode` e la condizione è sempre vera
- Può avvenire anche in istruzioni semplici (e.g., `x == 1;`)
 - il valore viene perso
 - molti compilatori generano un avvertimento riguardo a una tale istruzione

In coppia:



Quale non è un'istruzione di iterazione?

- a) continue
- b) for
- c) while
- d) do ... while

Risposta: **1**

Quale istruzione è usata per saltare il resto del corpo di un'istruzione di iterazione e procedere con la prossima iterazione del ciclo?

- a) skip
- b) proceed
- c) continue
- d) jump

Risposta: **2**

In coppia:



Che valori della variabile di controllo x vengono stampati ?
(rispondere con i valori separati da uno spazio, e.g., "2 8 99")

```
for (int x = 2; x <= 13; x += 2) {  
    printf("%d\n", x);  
}
```



Risposta:

1

Clicca sullo schermo di proiezione per avviare la domanda

```
for (int x = 5; x <= 22; x += 7) {  
    printf("%d\n", x);  
}
```

Risposta:

2

```
for (int x = 3; x <= 15; x += 3) {  
    printf("%d\n", x);  
}
```

Recap

- Iterazione in C con contatore e sentinella
 - Istruzione `for`
 - Istruzione `do-while`
- Selezione multipla con istruzione `switch`
- Stampare output numerico e precisione dei numeri in virgola mobile
- Istruzioni `break` e `continue`
- Operatori logici in C

La programmazione strutturata

- La programmazione strutturata favorisce la chiarezza e la semplicità del codice con 3 forme di controllo:
 - **Sequenza:** esecuzione lineare delle istruzioni
 - **Selezione:** decisioni basate su condizioni
 - **Iterazione:** ripetizione delle istruzioni
- L'istruzione `if` è sufficiente per implementare tutte le forme di selezione (incluso `if-else` e `switch`)
- L'istruzione `while` è sufficiente per implementare tutte le forme di iterazione (incluso `do-while` e `for`)
- La programmazione strutturata, utilizzando solo queste tre forme di controllo e le loro combinazioni, supporta una progettazione chiara e semplice dei programmi