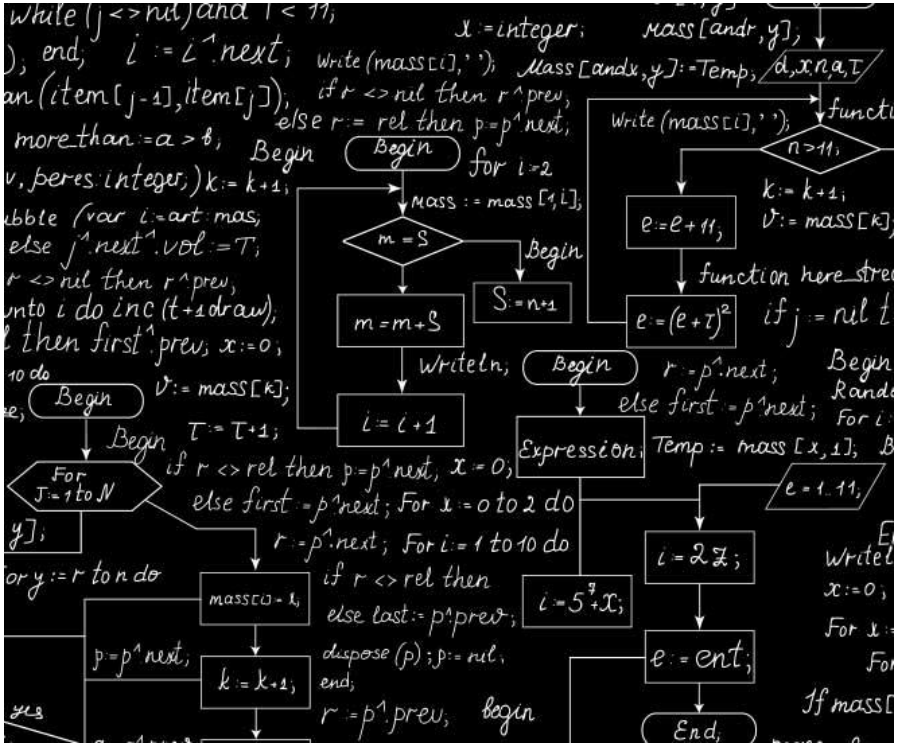
Sviluppo di un programma strutturato in C

Credits to P. Pinoli e Marcello M. Bersani

Algoritmi

- Prima di scrivere un programma per risolvere un problema, è necessario avere una comprensione approfondita del problema e un approccio di soluzione accuratamente pianificato
- La soluzione di qualsiasi problema informatico implica l'esecuzione di una serie di azioni in un ordine specifico
- Un **algoritmo** è una procedura per risolvere un problema, definendo le **azioni** da eseguire e l'**ordine** in cui queste azioni devono essere eseguite



Fonte: lindiceonline.com

Esempio di algoritmo

- Ecco un esempio di un "algoritmo" per preparare una tazza di caffè (americano):
  1. Riempire il bollitore con acqua
  2. Accendere il bollitore e aspettare che l'acqua raggiunga l'ebollizione
  3. Mettere una bustina di caffè solubile o una cialda nella tazza
  4. Versare l'acqua calda nella tazza
  5. Mescolare il caffè con un cucchiaino
  6. Aggiungere zucchero o latte, se desiderato
  7. Mescolare di nuovo e gustare il caffè
- Questo esempio mostra l'importanza dell'ordine delle azioni: versare l'acqua prima di mettere la bustina di caffè, per esempio, non porterebbe al risultato desiderato!
- Specificare l'ordine in cui le istruzioni devono essere eseguite in un programma informatico è chiamato **controllo del programma**

Pseudocodice

- Lo pseudocodice è un linguaggio artificiale **informale**
- Aiuta a sviluppare algoritmi prima di convertirli in C
- Aiuta a **"pensare"** un programma prima di scriverlo in un linguaggio di programmazione
- I computer non eseguono il pseudocodice
- Può essere scritto in qualsiasi editor di testo
- Spesso convertire un pseudocodice preparato con attenzione in C è semplice come sostituire un'istruzione del pseudocodice con il suo equivalente in C

Pseudocodice

- Lo pseudocodice descrive le azioni e le decisioni
- Le definizioni **non sono istruzioni eseguibili**: sono semplicemente messaggi per il compilatore
  - `int i = 0;`
  - indica al compilatore il tipo della variabile i, istruisce il compilatore a riservare spazio in memoria per la variabile e la inizializza a 0
  - Non esegue un'azione quando il programma viene eseguito, come input, output, una calcolazione o un confronto
- Alcuni programmatori non includono le definizioni nel loro pseudocodice

Programmare in C

- le istruzioni in un programma sono eseguite una dopo l'altra nell'ordine in cui sono scritte
- Questa semplice modalità è chiamata **esecuzione sequenziale**
- Varie istruzioni in C permettono di specificare come istruzione successiva da eseguire una diversa da quella successiva nella sequenza
- Questa modalità è chiamata **trasferimento del controllo**

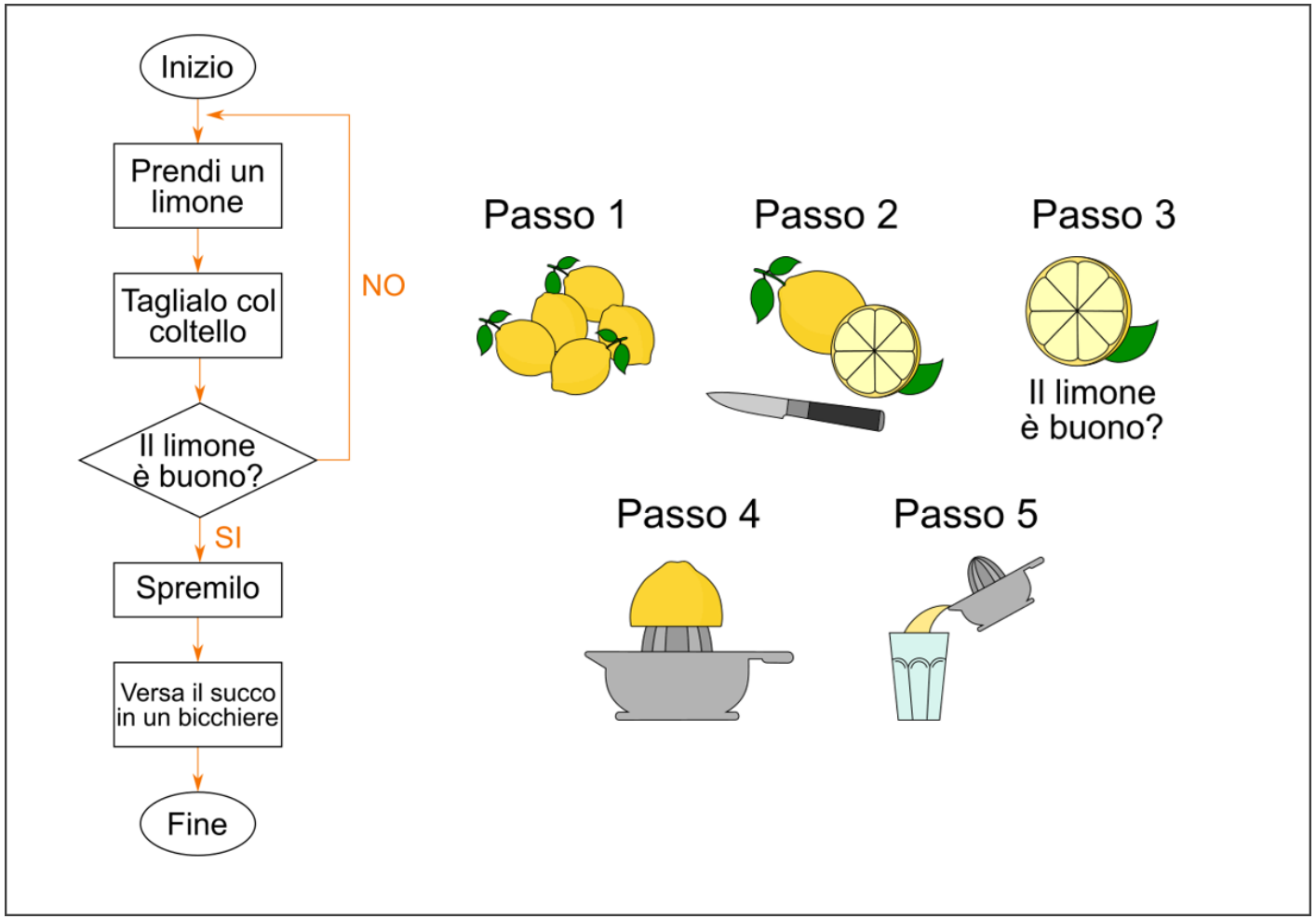
Strutture di controllo

- Programmazione strutturata: eliminazione del **goto**
- Böhm e Jacopini (1966) hanno dimostrato che tutti i programmi possono essere scritti usando solo tre **strutture di controllo**:
  - **struttura sequenziale**
  - **struttura di selezione**
  - **struttura di iterazione**
- La struttura sequenziale è semplice: il computer esegue le istruzioni in C una dopo l'altra nell'ordine in cui sono scritte, a meno che non venga specificato diversamente.

Diagramma di flusso

- Un diagramma di flusso è una **rappresentazione visiva** di un algoritmo o di una sua parte
- Questi diagrammi utilizzano **simboli specifici** come rettangoli, rombi, rettangoli arrotondati e cerchietti, collegati da frecce note come linee di flusso
- I diagrammi di flusso sono utili per **sviluppare e illustrare algoritmi**, anche se molti programmatori preferiscono utilizzare lo pseudocodice
- I diagrammi di flusso aiutano a visualizzare chiaramente il funzionamento delle **strutture di controllo**

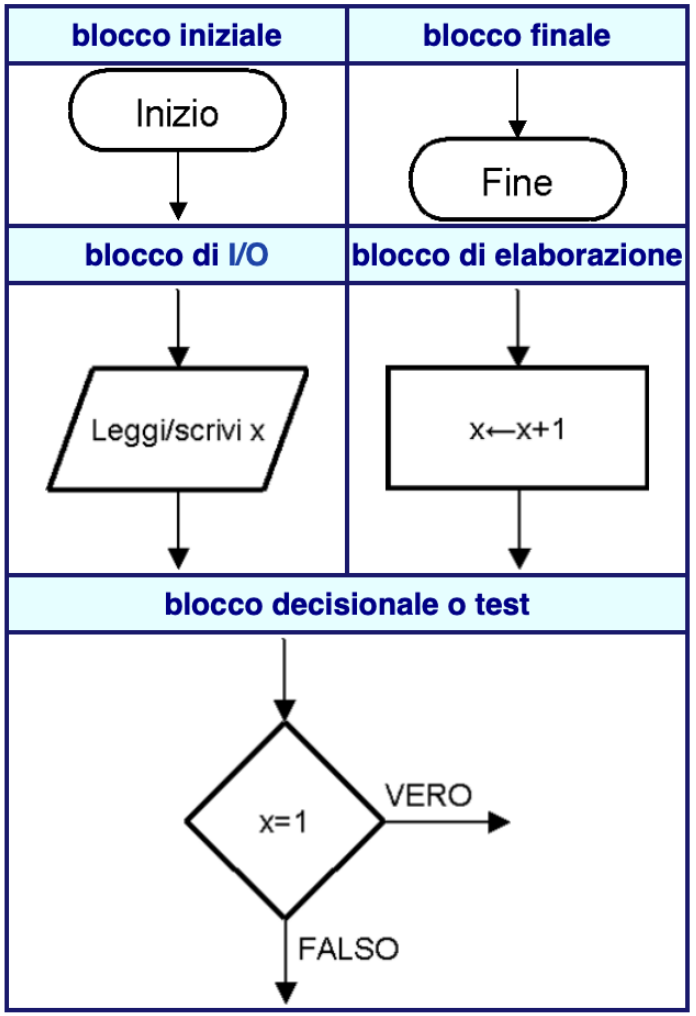
Diagramma di flusso



Fonte: dietroschermo.wordpress.com

Blocchi elementari

- **Blocco iniziale**: Indica l'inizio di un algoritmo o processo
- **Blocco finale**: Segna la conclusione di un algoritmo o processo
- **Blocco di I/O**: Utilizzato per operazioni di input e output, come leggere dati dall'utente o visualizzare risultati
- **Blocco di elaborazione**: Rappresenta le operazioni di calcolo o manipolazione dei dati
- **Blocco decisionale o condizionale**: Impiega un rombo per rappresentare una scelta o condizione logica che determina il percorso successivo del flusso del programma



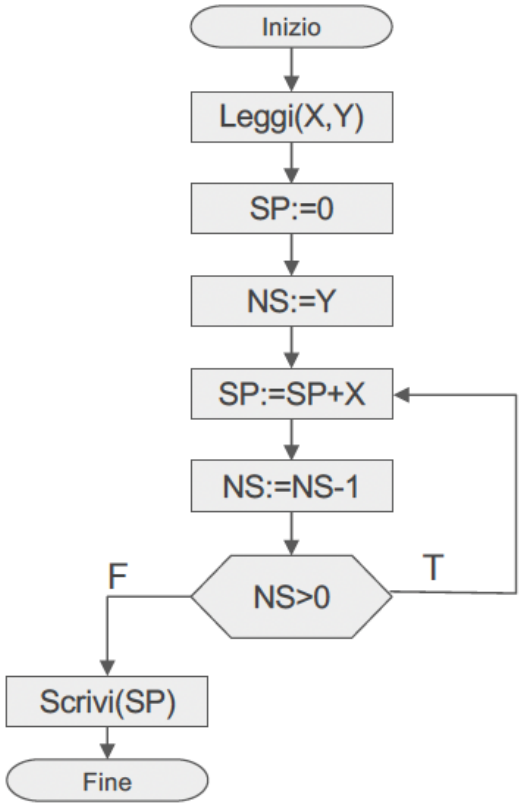
Fonte: Wikipedia

Regole di composizione

- La composizione dei blocchi base determina il flusso di controllo dell'algoritmo
- Un solo blocco di **inizio** ed almeno un blocco di **fine**
- Ogni blocco di inizio ed esecuzione genera al più un flusso in uscita
- Ogni blocco, ad eccezione di inizio, deve essere raggiunto da almeno un flusso
- Ogni blocco condizionale genera **due flussi:**
  - Uno (T) eseguito quando la condizione è verificata, l'altro (F) quando è falsificata
- In ogni esecuzione il flusso da eseguire è unico e determinato dalla condizione e dai valori delle variabili

Esempio: prodotto di due interi

- Problema: Prodotto di due interi X,Y
- **Vincoli:**
  - non è possibile usare il prodotto
- Sono disponibili le seguenti **operazioni di base**
  - Somma e sottrazione
  - Relazione d'ordine (<,>,=, etc.)
- Algoritmo: sommare X per Y volte

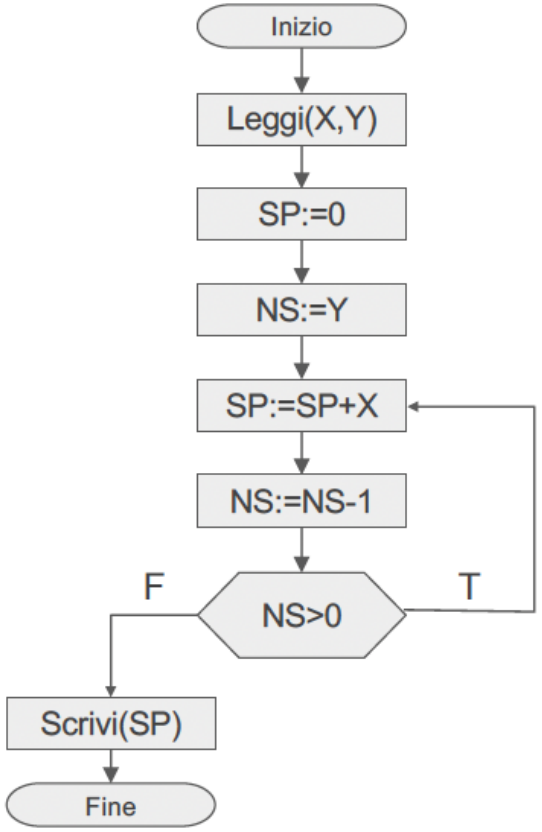


$$X \times Y = \underbrace{X + X + \ldots + X}_{Y \text{ volte}}$$

Fonte: Marcello M. Bersani

Algoritmo e pseudocodice (do-while)

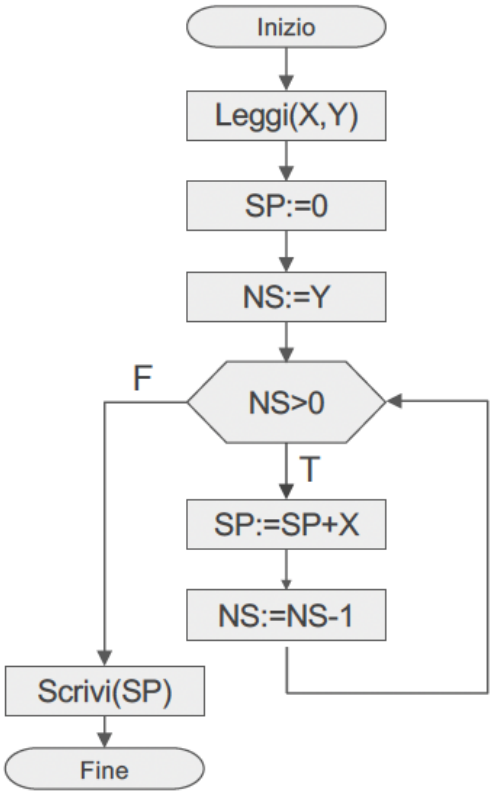
```
Variabili intere X,Y,SP,NS
read(X,Y)
SP := 0
NS := Y
do
    SP := SP+X
    NS := NS-1
while (NS>0)
write(SP)
```



Fonte: Marcello M. Bersani

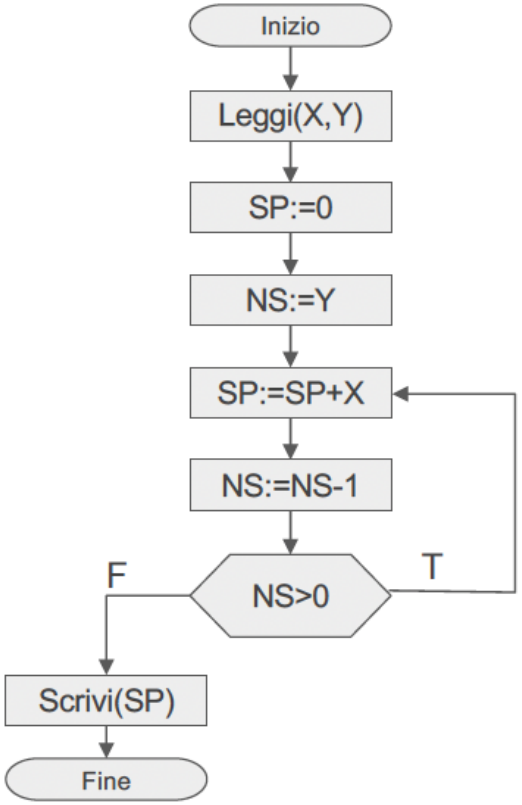
Algoritmo e pseudocodice (while)

```
Variabili intere X,Y,SP,NS
read(X,Y)
SP := 0
NS := Y

while (NS>0)
    SP := SP+X
    NS := NS-1

write(SP)
```

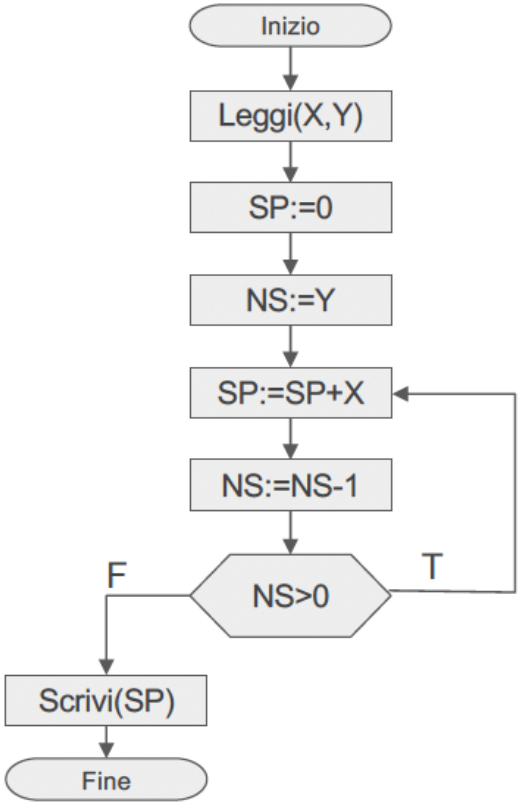


Fonte: Marcello M. Bersani

Quiz

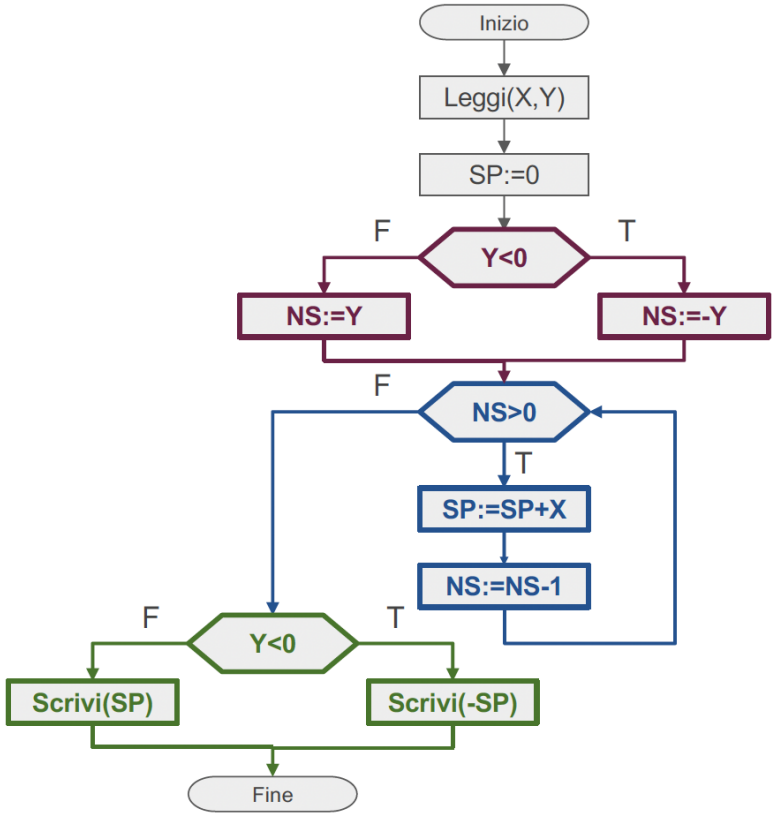


Quiz



Algoritmo e pseudocodice raffinato

```
Variabili intere X,Y,SP,NS
read(X,Y)
SP := 0
if (Y<0) then NS:=-Y
         else NS:=Y

while (NS>0)
        SP := SP+X
    NS := NS-1

if (Y<0) then write(-SP)
         else write(SP)
```



Fonte: Marcello M. Bersani

Programmazione strutturata in C

- Il linguaggio C dispone di **sette istruzioni di controllo**: **sequenza**, tre tipi di **selezione** (`if, if-else, switch`), e tre tipi di **iterazione** (`while, do-while, for`)
- Ogni programma in C è composto dalla **combinazione** di queste istruzioni di controllo, selezionate in base all'algoritmo da implementare
- Le istruzioni di controllo possono essere **"accatastate"** collegando il punto di uscita di una con il punto di ingresso della successiva, oppure annidate, permettendo di costruire qualsiasi programma con semplicità

Strutture di selezione in C

- Il linguaggio C offre tre tipi di **strutture di selezione** sotto forma di istruzioni:
  - L'istruzione di **selezione semplice** `if` esegue un'azione (o un gruppo di azioni) solo se una condizione è vera
  - L'istruzione di **selezione doppia** `if-else` esegue un'azione (o un gruppo di azioni) se la condizione è vera e un'altra azione (o un altro gruppo di azioni) se la condizione è falsa
  - L'istruzione di **selezione multipla** `switch` esegue una delle molte azioni possibili in base al valore di un'espressione.

Strutture di iterazione in C

- Il linguaggio C offre tre tipi di **strutture di iterazione** sotto forma di istruzioni:
  - `while` esegue il blocco di codice solo se la condizione è vera, controllandola prima di ogni iterazione
  - `do-while` esegue il blocco di codice almeno una volta e poi controlla la condizione alla fine di ogni iterazione
  - `for` è utilizzata per le iterazioni che richiedono una variabile di controllo e un numero definito di esecuzioni
- Queste istruzioni permettono di eseguire blocchi di codice ripetutamente fino al soddisfacimento di una condizione.

Intervallo

Fonte: PlaygroundAI

Selezione semplice: istruzione `if`

- L'operatore `if` valuta una condizione logica specificata tra parentesi tonde
- Se la condizione è vera (TRUE), il blocco di codice all'interno delle parentesi graffe {} viene eseguito
- La sintassi base è
  - `if (condizione) { /* codice */ }` dove "condizione" è un'espressione che restituisce un valore booleano
- Le variabili usate nella condizione devono essere dichiarate e inizializzate prima dell'uso dell'operatore `if`
- I commenti nel codice possono essere aggiunti per chiarire la logica dell'operatore `if` e il suo scopo

Selezione semplice: istruzione `if`

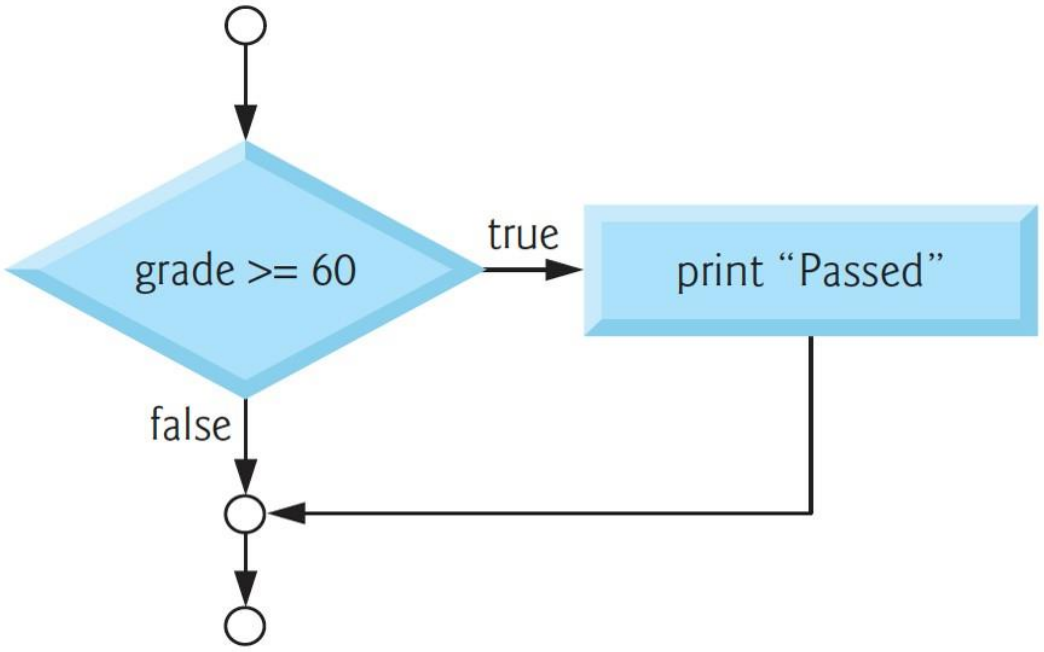
- Se il voto dello studente è superiore a 18,
  stampa «Passato»
- In C:

```
1.  // la variabile voto deve essere definita prima di questa riga
2.  if (voto >= 18) {
3.      printf("Passato");
4.  } // end if
```

- L'indentazione nella seconda riga è **opzionale**, ma altamente **raccomandata**.
  - Aiuta a evidenziare la struttura intrinseca dei programmi
  - Il compilatore ignora i caratteri di spaziatura utilizzati per l'indentazione e il posizionamento verticale, come spazi, tabulazioni e nuove righe
  - l'uso corretto dell'indentazione migliora la leggibilità e la manutenzione del codice.

Selezione semplice: istruzione `if`

- Blocco elementare più importante (decisione)
- Il rombo contiene la condizione da controllare



Fonte: Deitel & Deitel

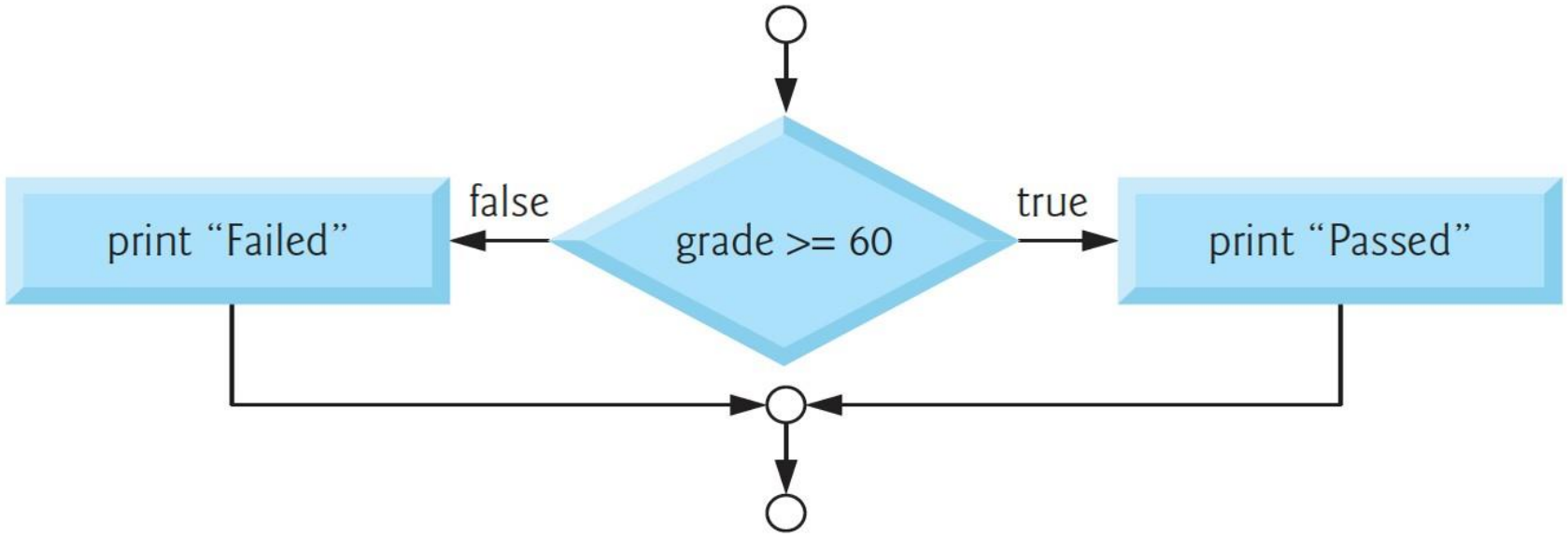
Selezione doppia: istruzione `if-else`

- Se il voto dello studente è superiore a 18
  stampa «Passato»
- **Altrimenti**
  stampa «Non passato»
- In C:

```
1.  // la variabile voto deve essere definita prima di questa
    riga
2.  if (voto >= 18) {
3.      printf("Passato");
4.  }
5.  else {
6.      printf(“Non passato”);
7.  } // fine if-else
```

- Anche il corpo di «Altrimenti» viene indentato

Selezione doppia: istruzione `if-else`



Fonte: Deitel & Deitel

Selezione doppia: operatore condizionale ? :

- Unico operatore **ternario** in C
  - Il primo operando è una condizione
  - Il secondo operando è il valore dell'espressione condizionale se la condizione è *vera*
  - Il terzo è il valore dell'espressione condizionale se la condizione è *falsa*
- Stampa condizionale:

```
printf( (voto >= 18) ? "Passato" : "Non passato");
```

- Assegnazione condizionale:

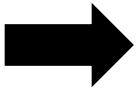
```
int a = 10, b = 20;
int max = (a > b) ? a : b; // Se a è maggiore di b,
max è a; altrimenti, max è b.
```

Istruzioni annidate `if-else`

- Le istruzioni annidate permettono di verificare più condizioni inserendo una `if-else` dentro un'altra `if-else`

```
if (voto >= 28) {
   printf("A");
} // end if
else {
   if (voto >= 25) {
      printf("B");
   } // end if
   else {
      if (voto >= 21) {
         printf("C");
      } // end if
      else {
         if (voto >= 18) {
            printf("D");
         } // end if
         else {
            printf("F");
         } // end else
      } // end else
   } // end else
} // end else
```

di solito si scrive così



```
if (voto >= 28) {
   printf("A");
} // fine di if
else if (voto >= 25) {
   printf("B");
} // fine di else if
else if (voto >= 21) {
  printf("C");
} // fine di else if
else if (voto >= 18) {
  printf("D");
} // fine di else if
else {
  printf("F");
} // fine di else
```

Istruzioni composte

- Per includere più istruzioni nel corpo di un `if`, è necessario racchiuderle tra parentesi graffe `{ }`
- Un insieme di istruzioni racchiuso tra parentesi graffe è chiamato **istruzione composta** o **blocco**.
- Un'istruzione composta può essere usata ovunque nel programma in cui è permessa una singola istruzione
- Esempio:

```
1.  // la variabile voto deve essere definita prima di questa riga
2.  if (voto >= 18) {
3.      printf("Passato");
4.  }
5.  else {
6.      printf(“Non passato.”);
7.      printf(“Mi dispiace!”);
8.  } // fine if-else
```

Istruzioni composte – errori

- Le parentesi graffe sono cruciali per includere correttamente tutte le istruzioni nel blocco `else`
  - Altrimenti l'istruzione viene eseguita a prescindere (**istruzione vuota**)

```
if (voto >= 18);
    printf("Passato");
```

- Per evitare errori logici come l'**else sospeso**, si consiglia sempre di racchiudere il corpo delle istruzioni di controllo tra parentesi graffe, anche quando contiene una singola istruzione
  - Non rilevati dal compilatore!
- Molti ambienti di sviluppo integrati e editor di codice completano **automaticamente** la parentesi graffa di chiusura quando digitate quella di apertura

Iterazione con `while`

- L'operatore `while` esegue un blocco di codice ripetutamente finché una condizione specificata rimane vera
  - Valuta la condizione
  - Se la condizione è vera, esegue il blocco di codice
  - Ritorna alla valutazione della condizione e ripete il ciclo
  - Termina quando la condizione diventa falsa
- In C:

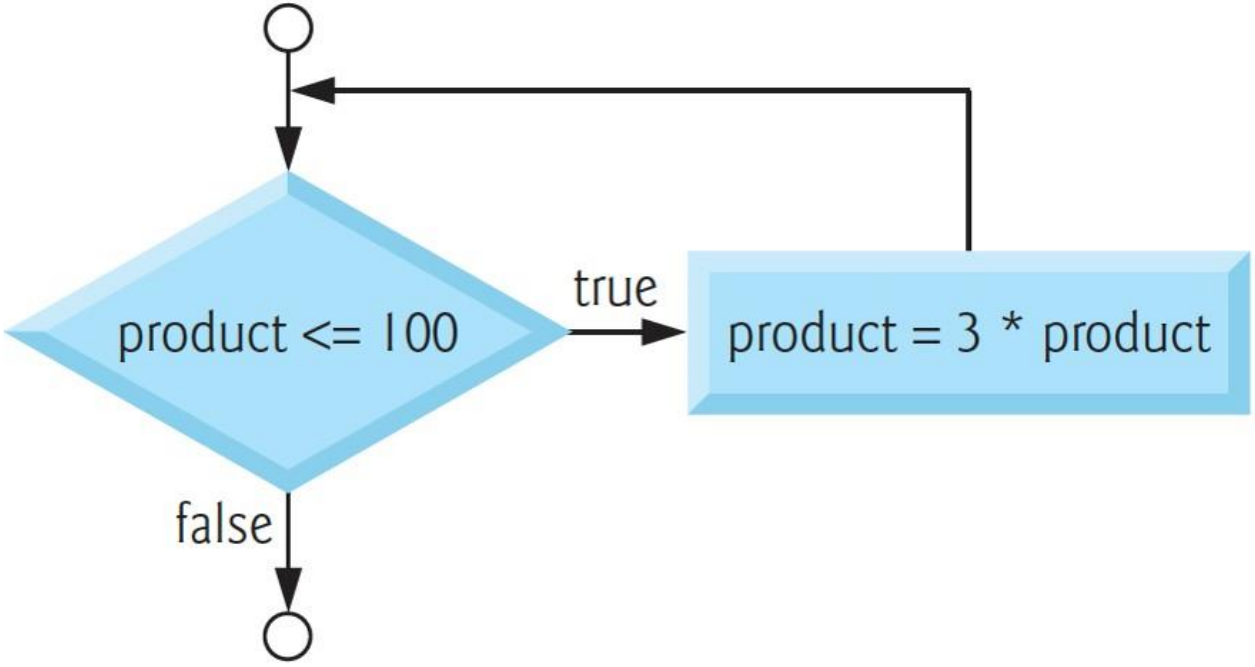
```
1. int i = 0;
2. while (i < 5) {
3.     printf("%d\n", i);
4.     i++;
5. }
```

Iterazione con `while`

- **Utilizzo:** ideale quando il numero di iterazioni non è noto a priori e dipende da una condizione dinamica
- **Attenzione:** assicurarsi che la condizione diventi falsa a un certo punto per evitare cicli infiniti
  - Nel corpo ci deve essere un'istruzione che modifichi la variabile su cui è posta la condizione
  - Oppure si può utilizzare `break` (prossima lezione!)

Iterazione con `while`

- Costruito con un blocco decisionale ed un istruzione che entra di nuovo nel blocco condizionale



Fonte: Deitel & Deitel

Esempio: media voti esami

- Una classe di dieci studenti fa un quiz. Avete a disposizione i voti (numeri interi compresi tra 0 e 100) per questo quiz
- Determinare la media della classe in riferimento al quiz
- Pseudocodice:

```
Imposta il totale uguale a zero
Imposta il contatore dei voti uguale a uno

Finché il contatore dei voti è minore o uguale a dieci
   Inserisci il voto successivo
   Somma il voto al totale
   Somma uno al contatore dei voti

Imposta la media della classe uguale al totale diviso per dieci
Stampa la media della classe
```

Fonte: Deitel & Deitel

Esempio: media voti esami

```c
// fig03_02.c
// Media di una classe con l'iterazione controllata da contatore.
#include <stdio.h>

// la funzione main inizia l'esecuzione del programma
int main(void) {
   // fase di inizializzazione
   int total = 0; // inizializza il totale dei voti a 0
   int  counter =  1 ; // numero del voto da inserire successivamente

    // fase di elaborazione
    while ( counter <=  10 ) { // ripeti 10 volte
       printf("%s", "Enter grade: "); // prompt per l'input
       int grade = 0; // valore del voto
       scanf("%d", &grade); // leggi il voto
       total = total + grade; // somma il voto al totale
       counter = counter +  1 ; // incrementa il contatore
    } // fine di while

    // fase di terminazione
    int average = total / 10; // divisione intera
    printf("Class average is %d\n", average); // stampa il risultato
 } // fine della funzione main
```

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

Fonte: Deitel & Deitel

Esempio: media voti esami

- Variabile usata per accumulare i voti (`total`)
  - Deve essere inizializzata a 0 altrimenti conterrà un valore «spazzatura»
  - Ultimo valore inizializzato in memoria
- Stessa cosa per la variabile usata come contatore (`counter`)
- La somma dei voti inseriti è 817, la media è 81.7
- Il programma ha stampato 81, perché?

Esempio: media voti esami (generalizzato)

- Come sviluppare un programma per il calcolo della media di una classe in grado di elaborare un numero **arbitrario** di voti?
- Utilizzo di un valore **sentinella** (o **flag**) per indicare la fine dell'inserimento dei dati in input
- In questo caso il programma utilizza una **iterazione indefinita**
- Il valore sentinella deve essere scelto in modo tale da non poter essere confuso con un valore di input valido
- Dato che i voti di un quiz sono numeri interi non negativi, -1 è un buon valore sentinella per questo problema
- Un esempio di sequenza di input per calcolare la media dei voti di una classe potrebbe essere: 95, 96, 75, 74, 89 e -1.
- Il programma calcolerà e stamperà la media solo per i voti 95, 96, 75, 74 e 89, escludendo il valore sentinella -1 dal calcolo.

Approccio top-down

- Per sviluppare un programma ben strutturato si parte da una definizione ad alto livello (**top**) delle istruzioni per risolvere il problema

```
Determina la media della classe per il quiz
```

- Processo di affinamento per suddividere in compiti più piccoli

```
Inizializza le variabili
Ricevi in ingresso, somma e conta i voti del quiz
Calcola e stampa la media della classe
```

- Primo affinamento semplice del programma in una struttura sequenziale

Approccio top-down

- Definiamo le seguenti variabili:
  - Una variabile per il totale corrente dei numeri
  - Una variabile per il conteggio dei numeri elaborati
  - Una variabile per memorizzare ogni voto inserito
  - Una variabile per calcolare e memorizzare la media
- Utilizziamo un ciclo iterativo con un valore **sentinella** per leggere i voti inseriti dall'utente uno alla volta
- Dopo aver inserito l'ultimo voto valido, l'utente inserirà il valore sentinella (e.g., `-1`)
- Il programma controllerà ogni valore inserito e terminerà il ciclo quando verrà immesso il valore sentinella
  - Utile segnalarlo nel prompt per leggere l'input dall'utente

Pseudocodice media voti esami (generalizzato)

```
Inizializza il totale a zero
Inizializza il contatore a zero

Ricevi in ingresso il primo voto (eventualmente la sentinella)
Finché l'utente non ha ancora inserito la sentinella
   Aggiungi il voto al totale corrente
   Aggiungi uno al contatore dei voti
   Ricevi in ingresso il voto successivo (eventualmente la sentinella)

Se il contatore non è uguale a zero
   Imposta la media uguale al totale diviso per il contatore
   Stampa la media
altrimenti
   Stampa "No grades were entered"
```

Fonte: Deitel & Deitel

- Abbiamo verificato la possibilità di **divisione per zero**, che può causare un arresto anomalo del programma (**errore irreversibile**)
- Importante stampare un messaggio di errore, invece di permettere che il programma si blocchi (**crashing**)

Struttura di un programma

- La maggior parte dei programmi può essere suddivisa in tre fasi principali:
  - **Fase di inizializzazione**: Imposta e inizializza le variabili necessarie per il funzionamento del programma
  - **Fase di elaborazione**: Accoglie i dati in ingresso e aggiorna le variabili del programma in base a tali dati
  - **Fase di chiusura**: Calcola e visualizza i risultati finali del programma
- Spesso sono necessarie **tante** fasi di affinamento
- Una volta terminata la scrittura dell'algoritmo in pseudocodice è molto semplice scrivere il programma in C!

Esempio: media voti esami (generalizzato)

```c
// fig03_04.c
// Media di una classe con iterazione controllata da sentinella.
#include <stdio.h>

// la funzione main inizia l'esecuzione del programma
int main(void) {
   // fase di inizializzazione
   int total = 0; // inizializza il totale
   int counter = 0; // inizializza il contatore del ciclo

   // fase di elaborazione
   // ricevi il primo voto dall'utente
   printf( "%s", "Enter grade, -1 to end: " ); // prompt per l'input
   int grade = 0; // valore del voto
   scanf( "%d", &grade); // leggi il voto dall'utente

   // ripeti finche' non viene letto il valore sentinella
   while (grade != -1) {
      total = total + grade; // aggiungi il voto al totale
      counter = counter + 1; // incrementa il contatore

      // ricevi il voto successivo dall'utente
      printf( "%s", "Enter grade, -1 to end: " ); // prompt per l'input
      scanf( "%d", &grade); // leggi il prossimo voto
   } // fine di while

   // fase di chiusura
   // se l'utente ha inserito almeno un voto
   if (counter != 0) {

      // calcola la media di tutti i voti inseriti
      double average = ( double ) total / counter; // evita il troncamento

      // stampa la media con la precisione di due cifre
      printf("Class average is %.2f\n", average);
   } // fine di if
   else { // se non sono stati inseriti voti, stampa un messaggio
      puts("No grades were entered");
   } // fine di else
} // fine della funzione main
```

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50

Enter grade, -1 to end: -1
No grades were entered
```

Attenzione al corpo del `while`!

Fonte: Deitel & Deitel

Variabili di tipo `double`

- Anche se si inseriscono solo voti interi, il calcolo della media può produrre un numero **decimale**, che non può essere rappresentato dal tipo `int`
- Le medie, come 7,2 o -93,5, contengono una parte frazionaria e sono rappresentabili con il tipo di dati `double`
- Normalmente il risultato di `total / counter` sarebbe un numero intero perché entrambe le variabili sono di tipo `int`
- Il programma utilizza il tipo di dati `double` per gestire i numeri decimali (**numeri in virgola mobile**) e l'operatore *cast* per assicurare che i numeri in virgola mobile siano usati nel calcolo della media

```
// calcola la media di tutti i voti inseriti
double  average = ( double ) total / counter; // evita il troncamento
```

Operatore cast

- La divisione tra due numeri interi produce un risultato intero **troncato** della parte frazionaria
  - per ottenere un risultato in virgola mobile, si utilizza l'operatore cast unario (`double`) per convertire temporaneamente il totale in un valore double
- L'operatore cast (double) crea una copia temporanea in virgola mobile di `total`, mentre `counter` rimane un valore intero, consentendo una divisione in virgola mobile

Operatore cast

- In C, per le espressioni aritmetiche con tipi di dati diversi, il compilatore esegue una conversione implicita per uniformare i tipi
  - In un'operazione con un `int` e un `double`, la variabile `int` viene convertita in `double`
- Gli operatori **cast** sono formati mettendo il nome del tipo tra parentesi e sono un operatore unario che si applica a un solo operando; hanno la stessa precedenza e associatività degli altri operatori unari e precedono gli operatori moltiplicativi come `*, / e %`

Formattare numeri in virgola mobile

```
// stampa la media con la precisione di due cifre
printf("Class average is %.2f\n", average);
```

- La specifica di conversione `%.2f` in `printf` è usata per formattare con due cifre decimali dopo il punto
- La lettera **f** indica che il valore è un numero in virgola mobile e **.2** specifica la precisione, cioè il numero di cifre decimali visualizzate
- Senza specificare la precisione, %f usa la precisione predefinita di **6 cifre decimali**, equivalente a %.6f
- Esempi di stampa con precisione:
  - `printf("%.2f\n", 3.446);` stampa 3.45
  - `printf("%.1f\n", 3.446);` stampa 3.4

Formattare numeri in virgola mobile

- I numeri in virgola mobile non sono sempre precisi al 100%, ma sono adeguati per molte applicazioni pratiche
- Ad esempio, una temperatura corporea di 98,6 gradi Fahrenheit può in realtà essere 98,599947321064 3; rappresentare il valore come 98,6 è sufficiente per la maggior parte dei casi
- La divisione di numeri, come 10 diviso 3, produce una sequenza infinita di decimali (3,3333…); i computer memorizzano solo un'approssimazione a causa dello spazio limitato
- La rappresentazione dei numeri in virgola mobile è **approssimativa** e potrebbe portare a risultati imprecisi
- **Evitare di confrontare** valori in virgola mobile per uguaglianza diretta, a causa delle possibili imprecisioni nella rappresentazione

Intervallo

Fonte: PlaygroundAI

Operatori di assegnazione

- In C, esistono operatori di assegnazione che semplificano le espressioni di assegnazione. Ad esempio:
  - L'operazione `c = c + 3` può essere abbreviata utilizzando l'operatore di assegnazione additiva +=, scrivendo `c += 3`
- L'operatore += aggiunge il valore dell'espressione a destra dell'operatore al valore della variabile a sinistra dell'operatore e poi memorizza il risultato nella variabile a sinistra

Esempi di assegnazione

```
int c = 3, d = 5, e = 4, f = 6, g = 12;
```

Assignment operator	Sample expression	Explanation	Assigns
+=	c += 7	c = c + 7	10 to c
-=	d -= 4	d = d - 4	1 to d
*=	e *= 5	e = e * 5	20 to e
/=	f /= 3	f = f / 3	2 to f
%=	g %= 9	g = g % 9	3 to g

Fonte: Deitel & Deitel

Operatori di incremento e decremento

- Gli operatori unari ++ e −− permettono di aumentare o diminuire il valore di una variabile di 1

Operator	Sample expression	Explanation
`++`	`++a`	Increment `a` by `1`, then use the new value of `a` in the expression in which `a` resides.
`++`	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by `1`.
`--`	`--b`	Decrement `b` by `1`, then use the new value of `b` in the expression in which `b` resides.
`--`	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by `1`.

Fonte: Deitel & Deitel

Esempio di pre- e post-incremento

```c
// fig03_07.c
// Preincremento e postincremento.
#include <stdio.h>

// la funzione main inizia l'esecuzione del programma
int main(void) {
   // illustrazione del postincremento
   int c = 5; // assegna 5 a c
   printf("%d\n", c); // stampa 5
   printf("%d\n", c++); // stampa 5 e poi postincrementa
    printf("%d\n\n", c); // stampa 6

    // illustrazione del preincremento
    c = 5; // assegna 5 a c
    printf("%d\n", c); // stampa 5
    printf("%d\n", ++c); // preincrementa e poi stampa 6
    printf("%d\n", c); // stampa 6
 } // fine della funzione main
```

```
5
5
6
5
6
6
```

Fonte: Deitel & Deitel

Esempio di operatori

- Le tre **assegnazioni**:
  - `passes = passes + 1;`
  - `failures = failures + 1;`
  - `student = student + 1;`
- Si possono scrivere con gli operatori di **assegnazione**:
  - `passes += 1;`
  - `failures += 1;`
  - `student += 1;`
- Con gli operatori di **pre-incremento**:
  - `++passes;`
  - `++failures;`
  - `++student;`
- Con gli operatori di **post-incremento**:
  - `passes++;`
  - `failures++;`
  - `student++;`

Attenzione all’ordine

- Quando si utilizza un operatore di incremento (o decremento) su una variabile in un'istruzione che coinvolge solo quella variabile, le forme di pre- e post- hanno lo stesso effetto
- In **espressioni più complesse**, la differenza tra preincremento e postincremento (e predecremento e postdecremento) diventa evidente.
  - La forma prefissa (++x, --x) modifica il valore prima dell'uso, mentre la forma postfissa (x++, x--) modifica il valore dopo l'uso
- Gli operatori di incremento e decremento possono essere applicati solo a variabili singole, non a espressioni.
  - Ad esempio, ++(x + 1) è un **errore di sintassi**
- Il linguaggio C generalmente **non specifica l'ordine** di valutazione degli operandi di un operatore.
  - Per evitare errori, è consigliabile usare gli operatori ++ e -- solo in istruzioni che coinvolgono una sola variabile

Recap

- Un algoritmo è una procedura composta da un insieme di azioni eseguite in un ordine prestabilito
- Lo pseudocodice è un linguaggio informale che aiuta a sviluppare algoritmi, simile al linguaggio naturale e non un vero linguaggio di programmazione
- Le istruzioni in un programma di solito vengono eseguite in sequenza. Tuttavia, alcune istruzioni in C permettono il trasferimento del controllo a un'altra istruzione, facilitando la programmazione strutturata, che migliora la chiarezza e la facilità di modifica
- I diagrammi di flusso visualizzano algoritmi e le istruzioni di controllo possono essere annidate o accatastate

Recap

- Le strutture di selezione permettono di scegliere tra linee alternative di azioni. Il simbolo di decisione (rombo) rappresenta una condizione che può essere vera o falsa, influenzando il percorso del flusso del programma
- Le istruzioni `if-else` possono essere annidate per gestire più casi e si usano blocchi di parentesi graffe per racchiudere gruppi di istruzioni
- L'istruzione `while` esegue un'azione ripetutamente finché una condizione è vera. Quando la condizione diventa falsa, il ciclo termina e il controllo passa alla prima istruzione dopo il ciclo