

Elementi Di Informatica E Programmazione

Prof. Andrea Loreggia



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

Istruzione switch



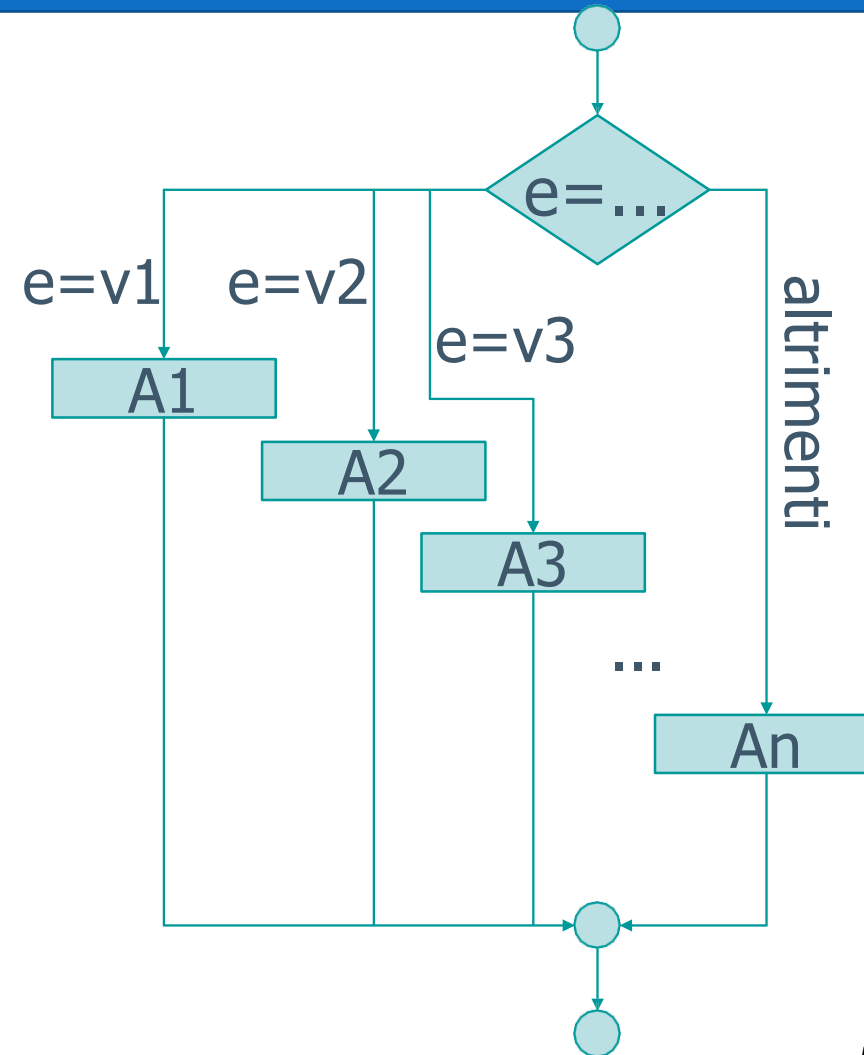
- Quando occorre compiere una sequenza di scelte, in funzione del valore di una variabile, occorre una catena di if-else
- Lo stesso risultato si può ottenere in forma più compatta mediante l'istruzione switch

```
if( mese == 1 )
    printf("Gennaio\n") ;
else if( mese == 2 )
    printf("Febbraio\n") ;
else if( mese == 3 )
    printf("Marzo\n") ;
else if( mese == 4 )
    printf("Aprile\n") ;
else if( mese == 5 )
    printf("Maggio\n") ;
.....
else if( mese == 9 )
    printf("Settembre\n") ;
else if( mese == 10 )
    printf("Ottobre\n") ;
else if( mese == 11)
    printf("Novembre\n") ;
else if( mese == 12 )
    printf("Dicembre\n") ;
else
    printf("MESE ERRATO!\n") ;
```

Sintassi istruzione switch



```
switch ( e )  
{  
    case v1:  
        A1 ;  
        break ;  
  
    case v2:  
        A2 ;  
        break ;  
  
    case v3:  
        A3 ;  
        break ;  
    .....  
    default:  
        An ;  
}
```



Precisazioni (1/2)



- L'espressione `e` può essere una variabile oppure un'espressione aritmetica
 - Il tipo di dato deve essere `int`, `char` o `enum`
- Ciascun caso è identificato da una costante
 - L'espressione `e` viene confrontata con il valore delle costanti `v1...vn`
 - Il tipo di dato deve essere compatibile
- Ciascun caso è delimitato da `case...break`
 - Non vi sono parentesi graffe `{...}`

Precisazioni (2/2)



- I casi possono apparire in qualsiasi ordine
 - Devono essere tutti diversi
- Verrà selezionato al più un caso
- Il caso default viene valutato se e solo se nessuno degli altri casi è stato considerato
 - Opzionale, ma sempre consigliato

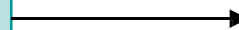
- Il significato di `break` è di portare l'esecuzione del programma fino al termine del costrutto `switch`
 - "Salta alla chiusa graffa": `}`
- In assenza di `break`, l'esecuzione proseguirebbe attraverso il caso successivo
 - Né il prossimo `case`, né eventuali parentesi graffe, possono fermare l'esecuzione lineare

Casi multipli



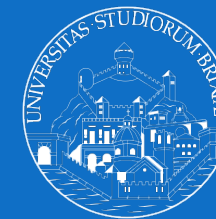
- Potrebbe essere necessario eseguire lo stesso codice in corrispondenza di diversi valori dell'espressione
- È possibile accomunare più casi, indicandoli consecutivamente

```
switch( ora )  
{  
    case 12:  
        pranzo = 1 ;  
        break;  
    case 13:  
        pranzo = 1 ;  
        break;  
}
```



```
switch( ora )  
{  
    case 12:  
    case 13:  
        pranzo = 1 ;  
        break;  
}
```

Esempio



```
switch( mese )
{
    case 1:
        printf("Gennaio\n") ;
        break ;
    case 2:
        printf("Febbraio\n") ;
        break ;
    case 3:
        printf("Marzo\n") ;
        break ;
    case 4:
        printf("Aprile\n") ;
        break ;
    .....
    case 12:
        printf("Dicembre\n") ;
        break ;
    default:
        printf("MESE ERRATO!\n") ;
}
```



mesi3.c

- Catene di istruzioni `if-else if-...-else`
- Annidamento delle istruzioni o condizioni Booleane complesse
- Uso di variabili logiche per tenere traccia delle condizioni incontrate
- Istruzione `switch` per sostituire alcuni tipi di catene `if-else if`
- Uso di `else` e `default` per catturare condizioni anomale



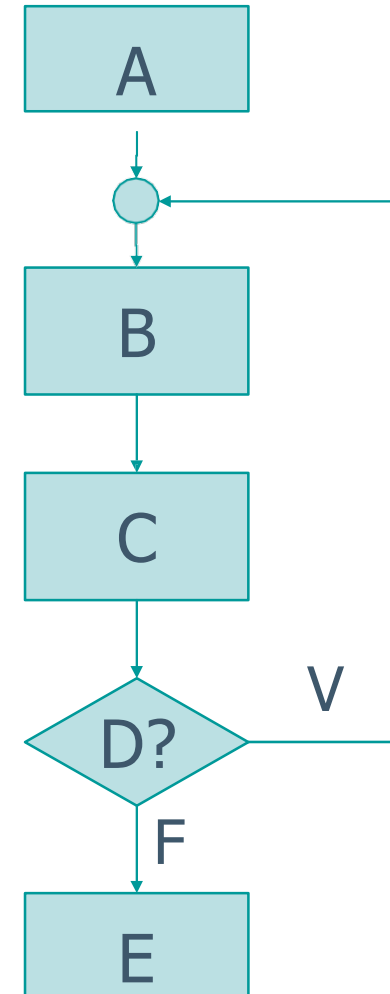
- Analizzare sempre tutti i casi possibili prima di iniziare a scrivere il programma
- Abbondare con le parentesi graffe Curare l'indentazione
- Aggiungere commenti in corrispondenza della clausola else e della graffa di chiusura

Flusso di esecuzione ciclico



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- È spesso utile poter **ripetere** alcune parti del programma più volte
- Nel diagramma di flusso, corrisponde a “tornare indietro” ad un blocco precedente
- Solitamente la ripetizione è controllata da una condizione booleana



Flusso di esecuzione ciclico

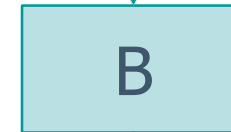


UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

Prima del
ciclo



Istruzioni
che vengono
ripetute



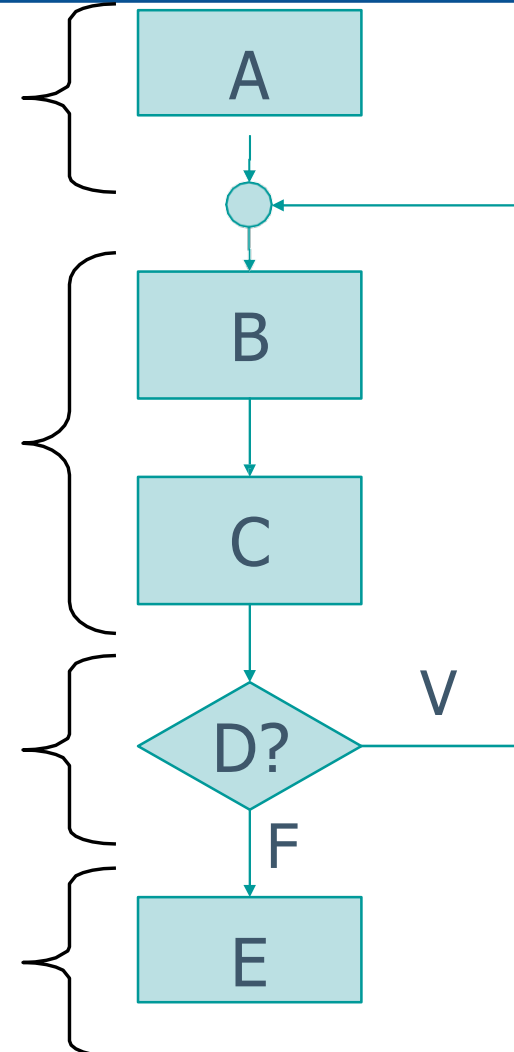
Condizione
di ripetizione



V

F

Dopo il ciclo



Errore frequente

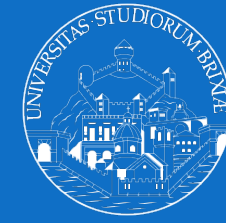


UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



- Ogni ciclo porta in sé il rischio di un grave errore di programmazione: il fatto che il ciclo venga ripetuto indefinitamente, senza mai uscire
- Il programmatore deve garantire che ogni ciclo, dopo un certo numero di iterazioni, venga terminato
 - La condizione booleana di controllo dell'iterazione deve divenire falsa

Istruzioni eseguibili ed eseguite



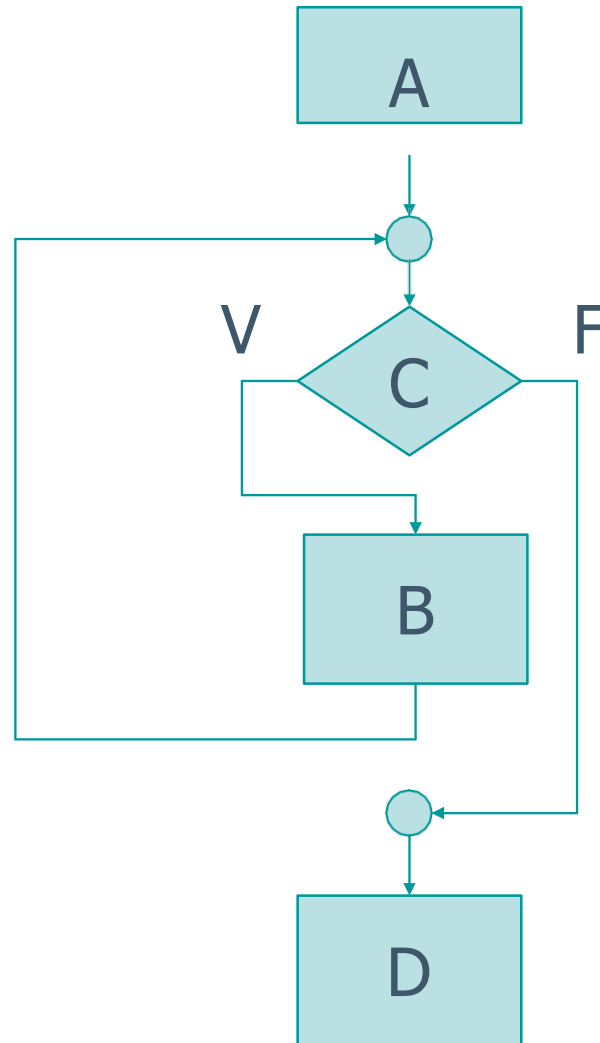
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Istruzioni eseguibili
- Le istruzioni che fanno parte del programma Corrispondono alle istruzioni del sorgente C
- Istruzioni eseguite
- Le istruzioni effettivamente eseguite durante una specifica esecuzione del programma
- Dipendono dai dati inseriti
- Nel caso di scelte, alcune istruzioni eseguibili non verranno eseguite
- Nel caso di cicli, alcune istruzioni eseguibili verranno eseguite varie volte

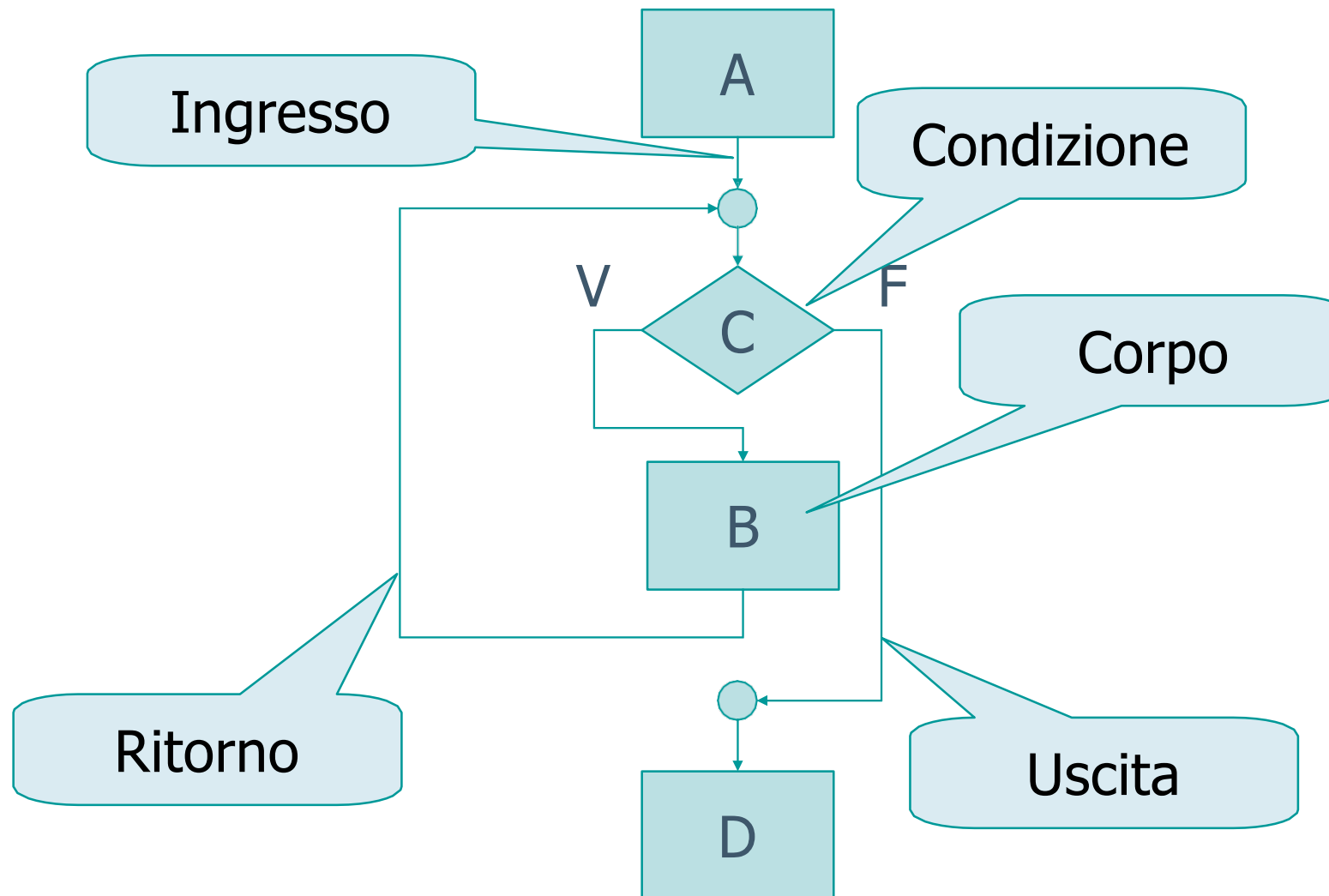
Notazione grafica (while)



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



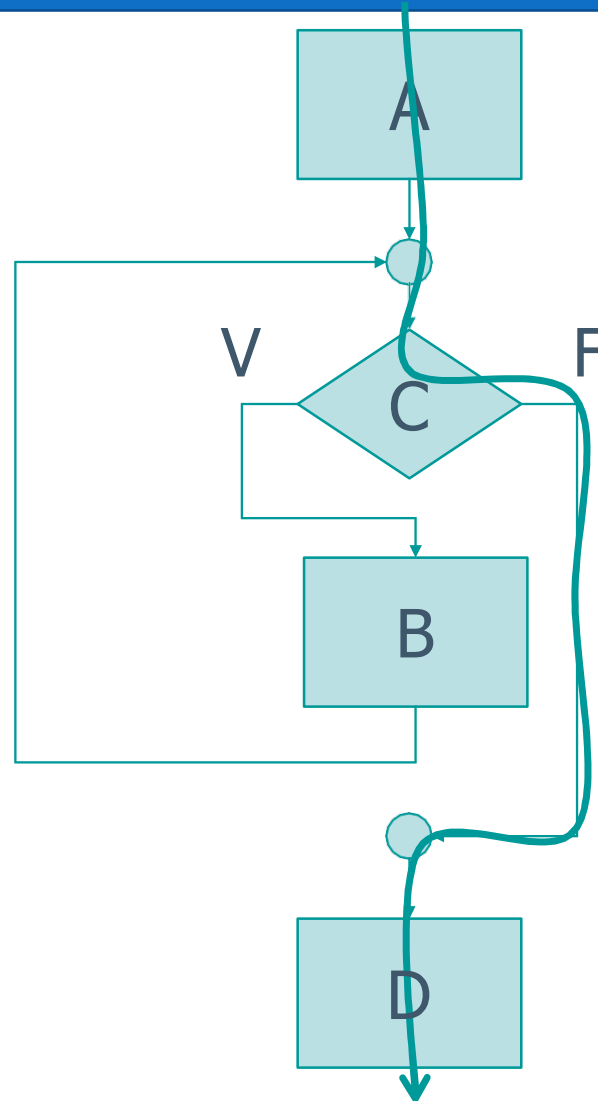
Notazione grafica (while)



Flussi di esecuzione



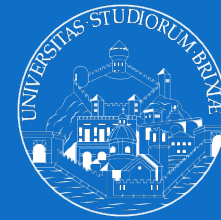
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



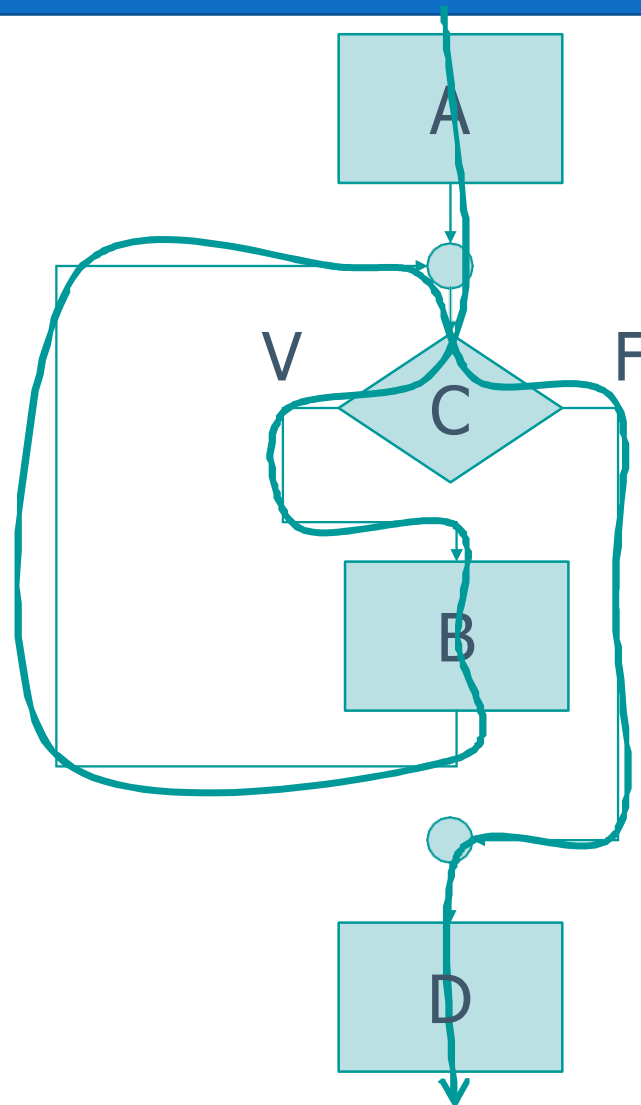
Zero iterazioni

A
C → Falso
D

Flussi di esecuzione



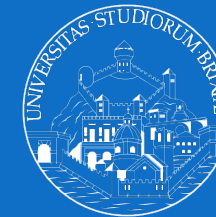
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



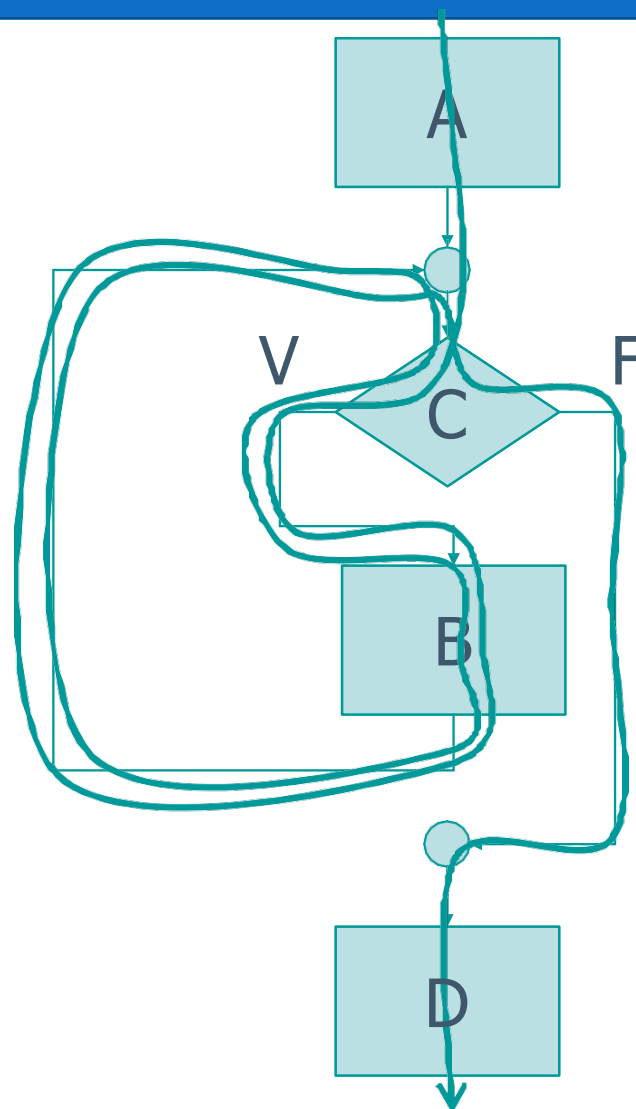
Una iterazione

A
C → Vero
B
C → Falso
D

Flussi di esecuzione



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



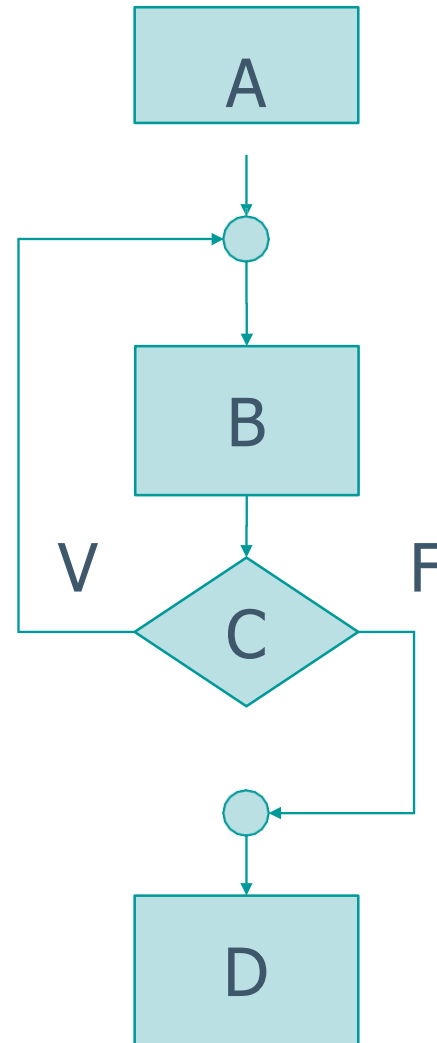
Due iterazioni

A
C → Vero
B
C → Vero
B
C → Falso
D

Notazione grafica (do-while)



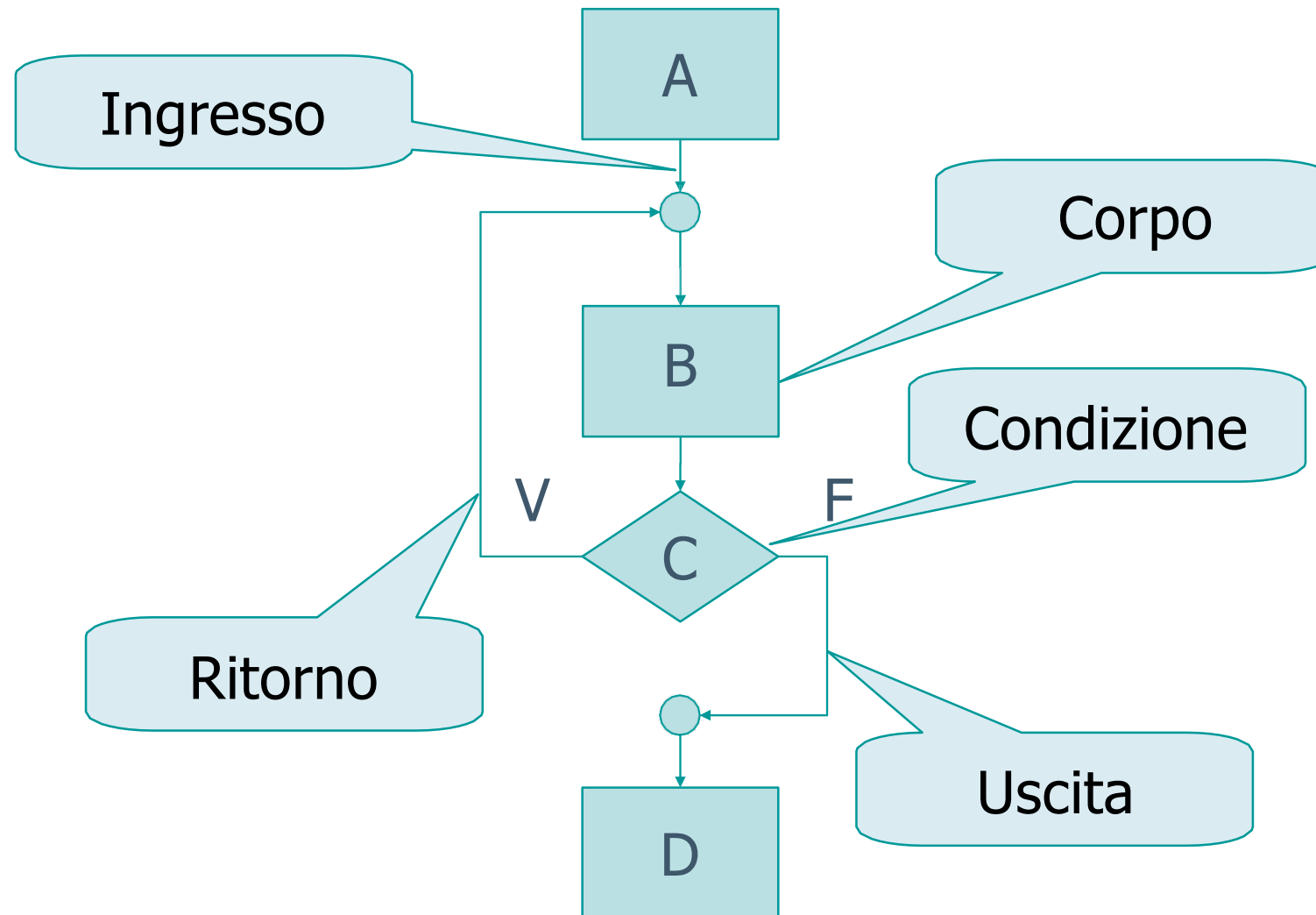
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



Notazione grafica (do-while)



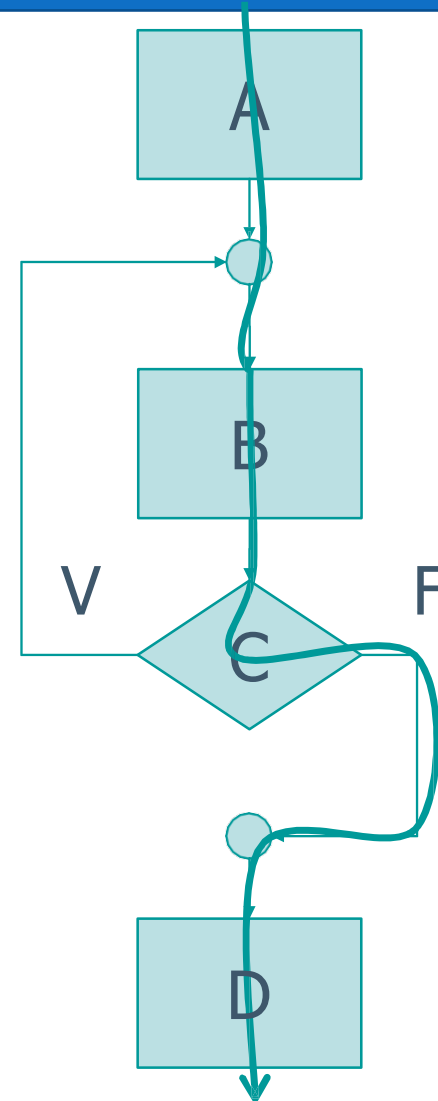
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



Flussi di esecuzione



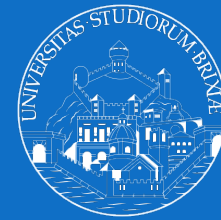
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



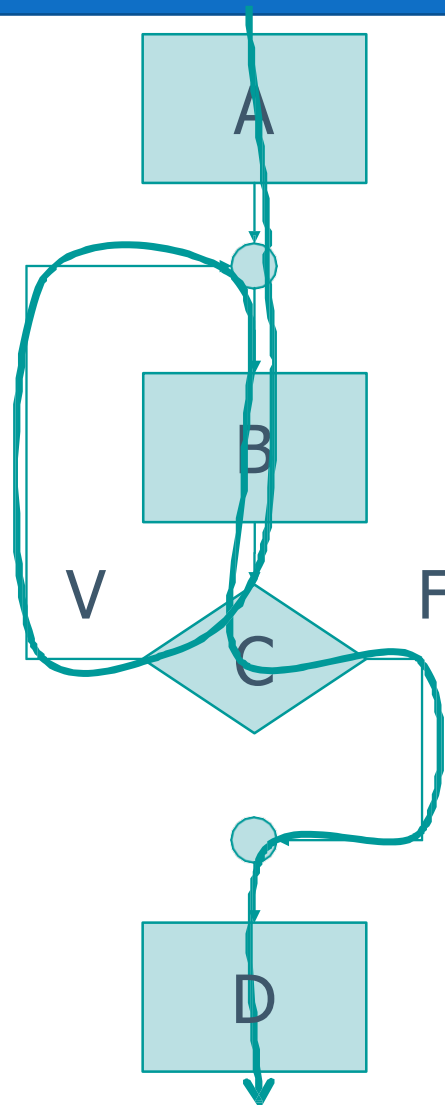
Una iterazione

A
B
C → Falso
D

Flussi di esecuzione



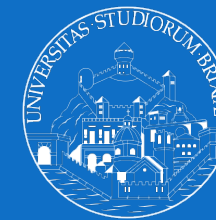
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



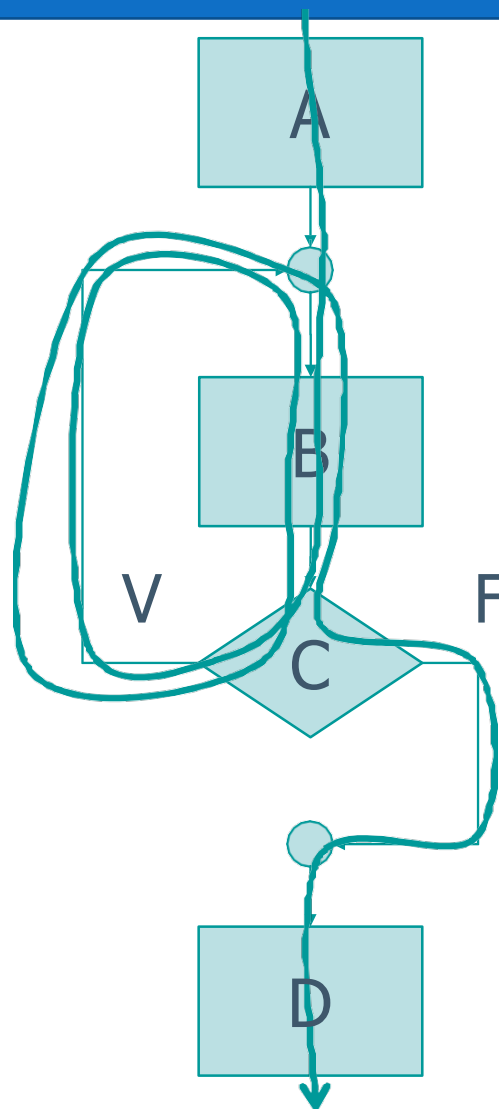
Due iterazioni

A
B
C → Vero
B
C → Falso
D

Flussi di esecuzione



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



Tre iterazioni

A
B
C → Vero
B
C → Vero
B
C → Falso
D

- Nello strutturare un ciclo occorre garantire:
 - Che il ciclo possa terminare
 - Che il numero di iterazioni sia quello desiderato
- Il corpo centrale del ciclo può venire eseguito più volte:
 - La prima volta lavorerà con variabili che sono state inizializzate al di fuori del ciclo
 - Le volte successive lavorerà con variabili che possono essere state modificare nell'iterazione precedente
 - Garantire la correttezza sia della prima, che delle altre iterazioni

Anatomia di un ciclo (1/5)



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Conviene concepire il ciclo come 4 fasi
 - Inizializzazione
 - Condizione di ripetizione
 - Corpo
 - Aggiornamento

Anatomia di un ciclo (2/5)



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Conviene concepire il ciclo come 4 fasi
 - Inizializzazione
 - Assegnazione del valore iniziale a tutte le variabili che vengono lette durante il ciclo (nel corpo o nella condizione)
 - Condizione di ripetizione
 - Corpo
 - Aggiornamento

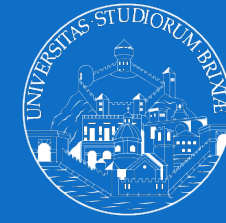
Anatomia di un ciclo (3/5)



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

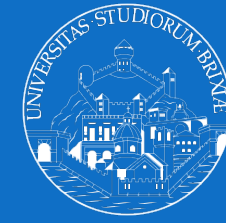
- Conviene concepire il ciclo come 4 fasi
 - Inizializzazione
 - Condizione di ripetizione
 - Condizione, di solito inizialmente vera, che al termine del ciclo diventerà falsa
 - Deve dipendere da variabili che saranno modificate all'interno del ciclo (nel corpo o nell'aggiornamento)
 - Corpo
 - Aggiornamento

Anatomia di un ciclo (4/5)



- Conviene concepire il ciclo come 4 fasi
 - Inizializzazione
 - Condizione di ripetizione
 - Corpo
 - Le istruzioni che effettivamente occorre ripetere
 - Sono lo scopo per cui il ciclo viene realizzato
 - Posso usare le variabili inizializzate
 - Posso modificare le variabili
 - Aggiornamento

Anatomia di un ciclo (5/5)



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

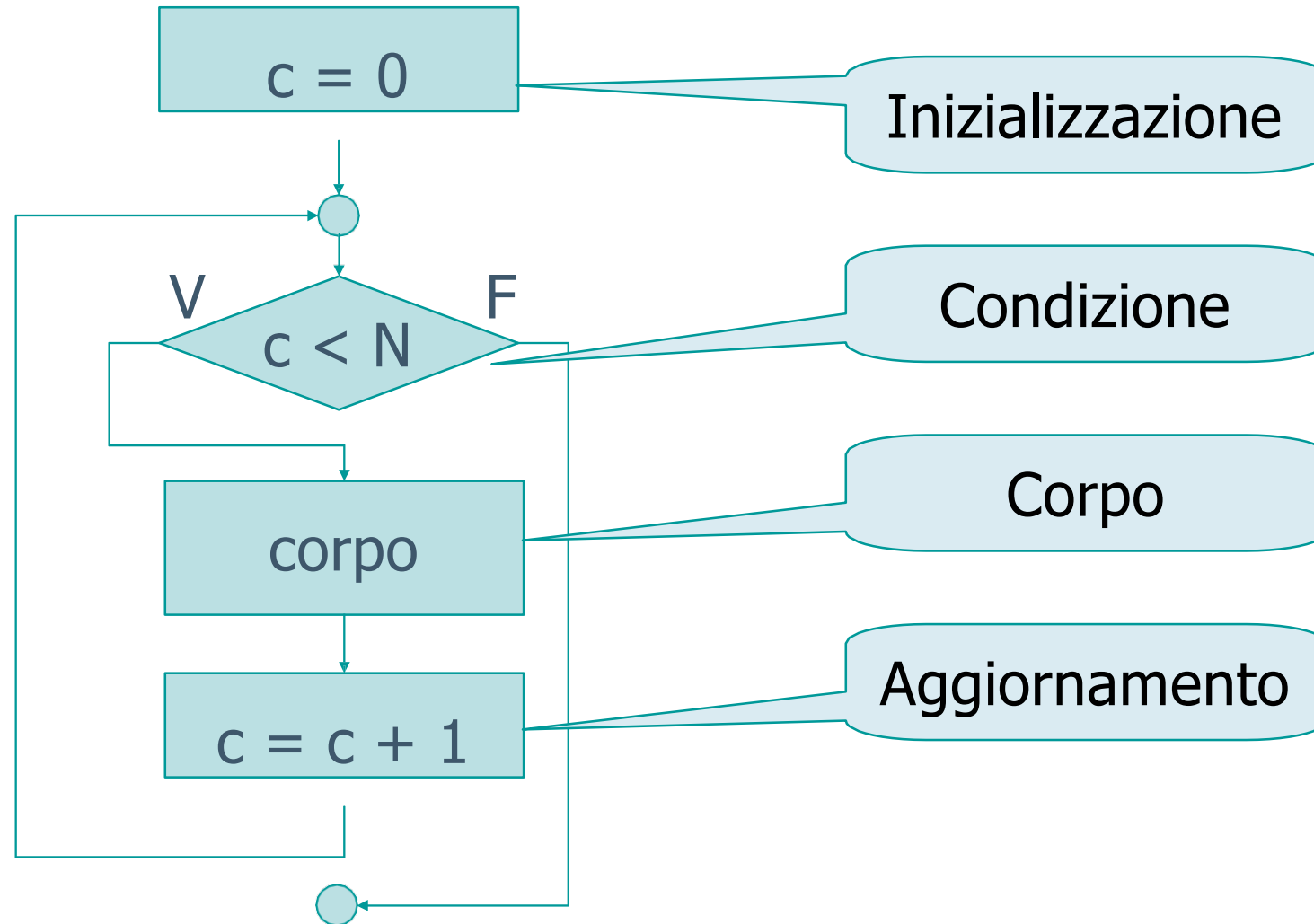
- Conviene concepire il ciclo come 4 fasi
 - Inizializzazione
 - Condizione di ripetizione
 - Corpo
 - Aggiornamento
 - Modifica di una o più variabili in grado di aggiornare il valore della condizione di ripetizione
 - Tengono "traccia" del progresso dell'iterazione

- Cicli in cui il numero di iterazioni sia noto a priori, ossia prima di entrare nel ciclo stesso
- Solitamente si usa una variabile “contatore” L’aggiornamento consiste in un incremento o decremento della variabile
- Cicli in cui il numero di iterazioni non sia noto a priori, ma dipenda dai dati elaborati nel ciclo
- Solitamente si usa una condizione dipendente da una variabile letta da tastiera oppure calcolata nel corpo del ciclo
- Difficile distinguere il corpo dall’aggiornamento

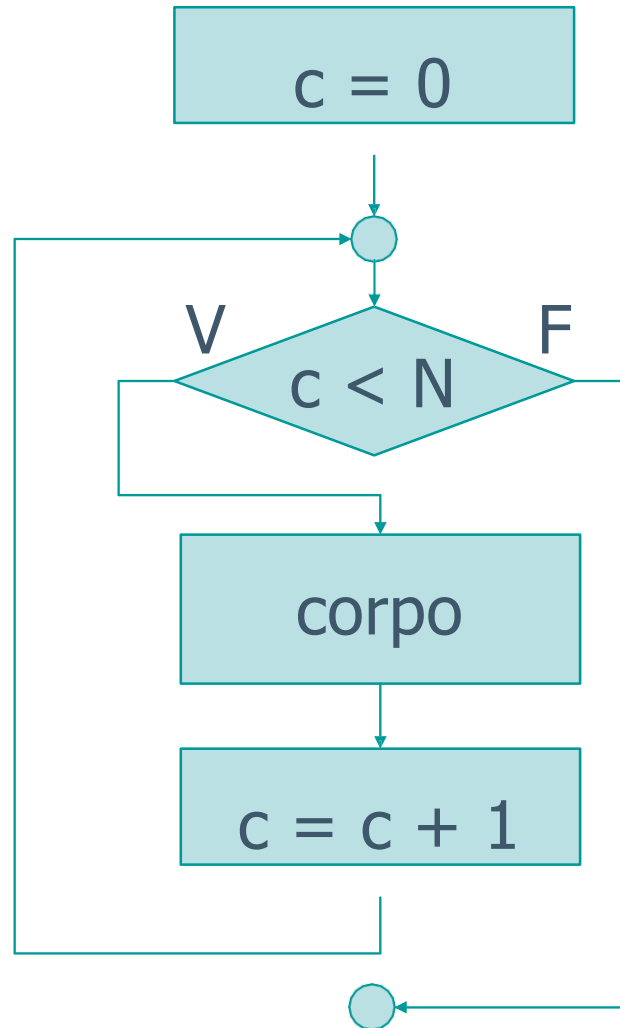
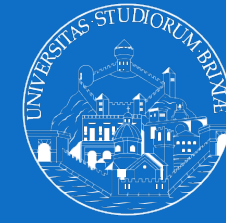
Cicli con iterazioni note



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

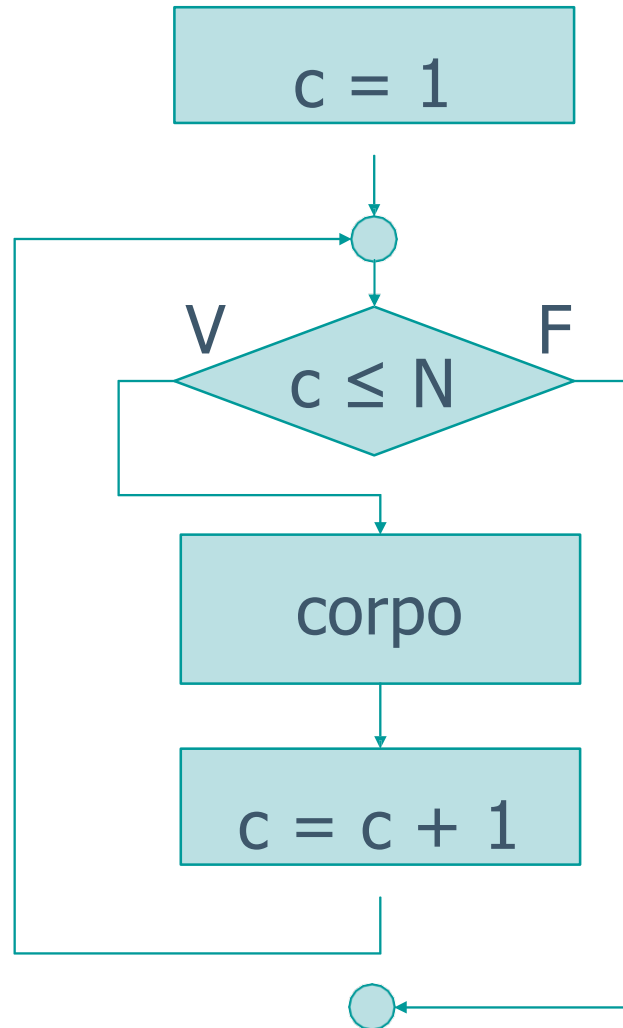


Cicli con iterazioni note



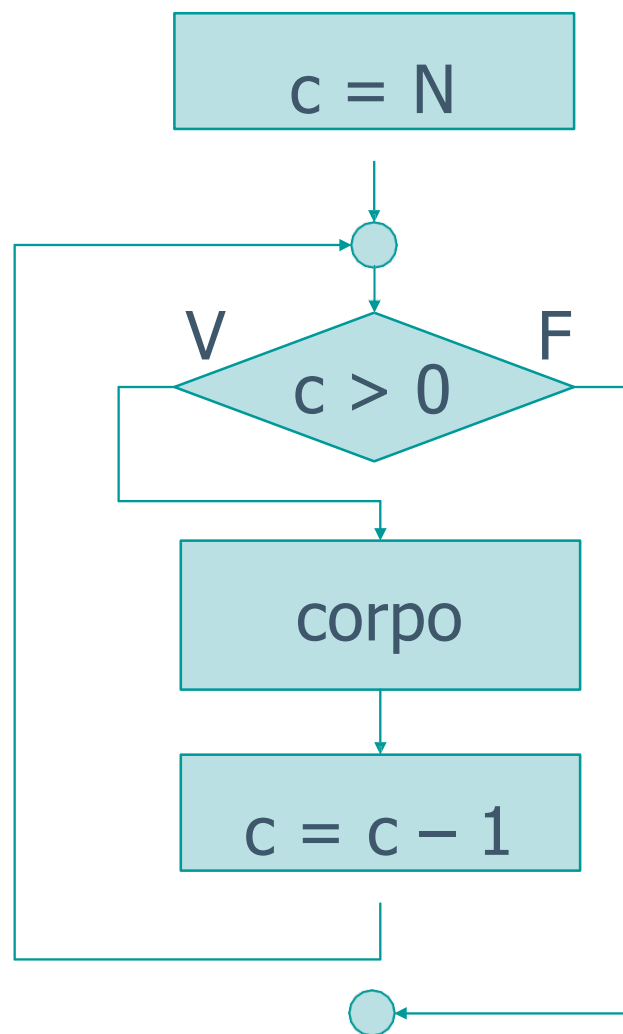
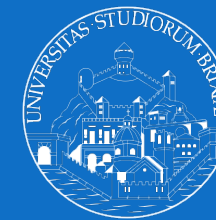
- Forma $0 \dots N-1$
- Prima iterazione:
 - $c=0$
- Ultima iterazione:
 - $c=N-1$
- Corpo ripetuto:
 - N volte
- Al termine del ciclo
 - $c=N$
 - condizione falsa

Cicli con iterazioni note



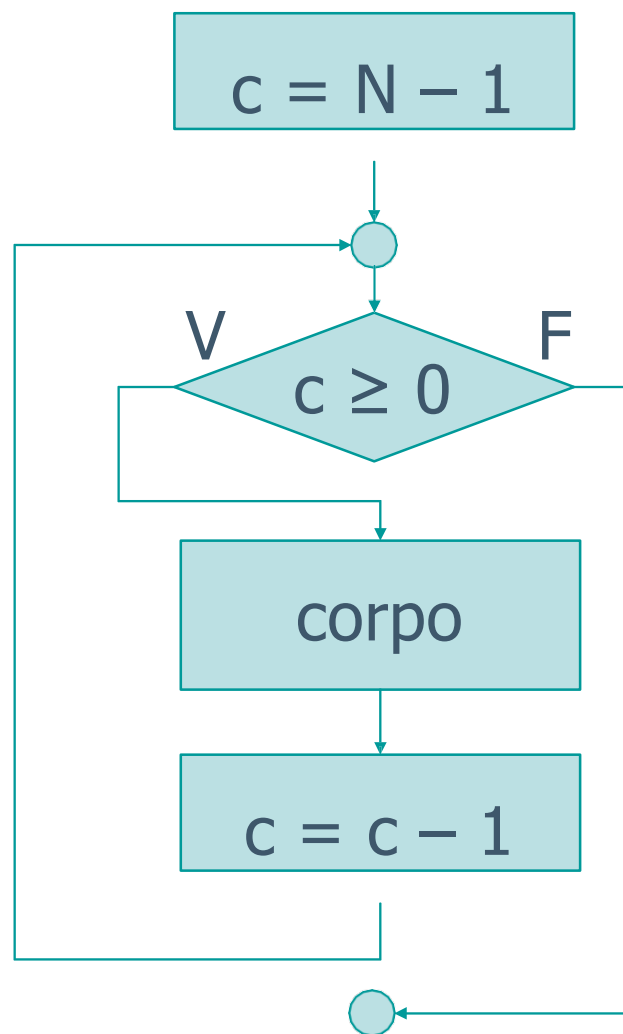
- Forma 1...N
- Prima iterazione:
 - $c=1$
- Ultima iterazione:
 - $c=N$
- Corpo ripetuto:
 - N volte
- Al termine del ciclo
 - $c=N+1$
 - condizione falsa

Cicli con iterazioni note



- Forma N...1
- Prima iterazione:
 - $c=N$
- Ultima iterazione:
 - $c=1$
- Corpo ripetuto:
 - N volte
- Al termine del ciclo
 - $c=0$
 - condizione falsa

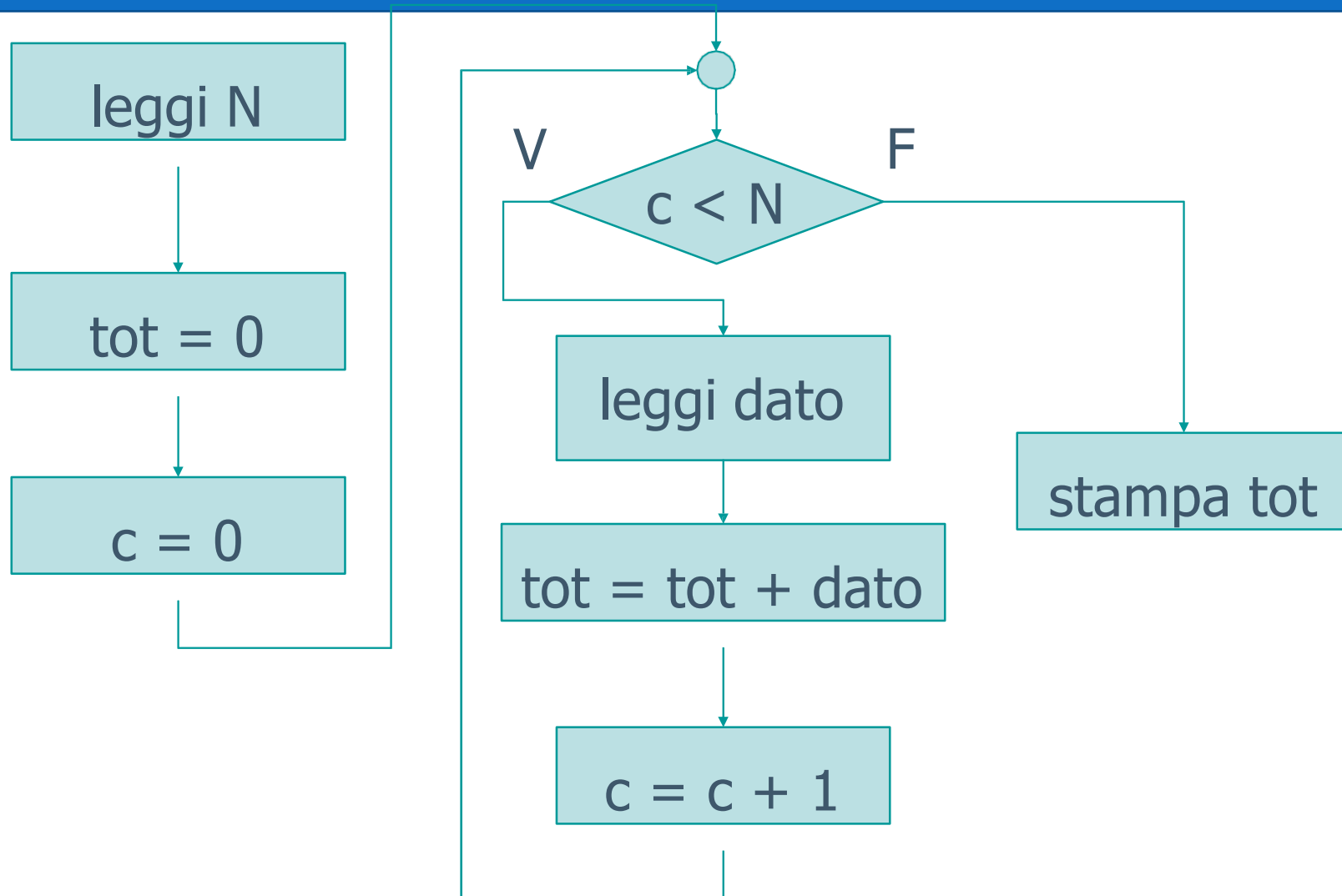
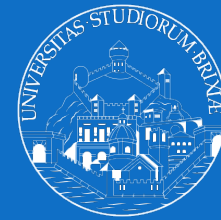
Cicli con iterazioni note



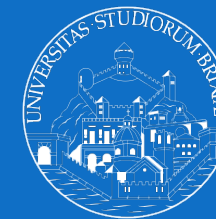
- Forma $N-1 \dots 0$
- Prima iterazione:
 - $c = N-1$
- Ultima iterazione:
 - $c = 0$
- Corpo ripetuto:
 - N volte
- Al termine del ciclo
 - $c = -1$
 - condizione falsa

- Acquisire da tastiera una sequenza di numeri interi e stamparne la somma.
- Il programma
 - inizialmente chiede all'utente quanti numeri intende inserire
 - in seguito richiede uno ad uno i dati
 - infine stampa la somma

Soluzione



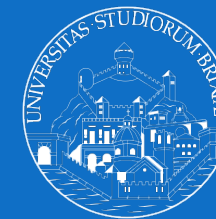
Cicli con iterazioni ignote



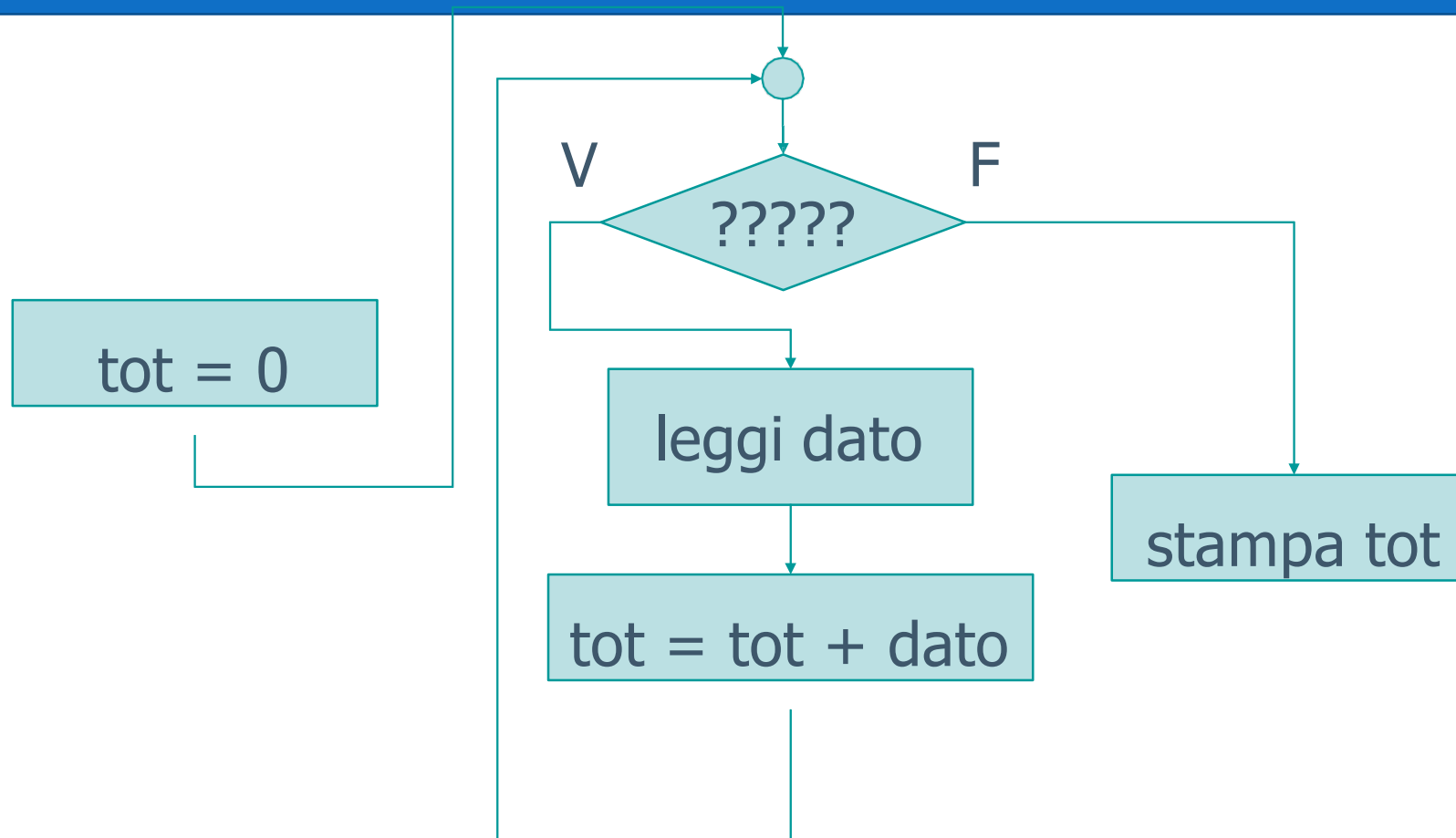
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Non esiste uno schema generale
- Esempio:
 - Acquisire da tastiera una sequenza di numeri interi e stamparne la somma.
 - Il termine della sequenza viene indicato inserendo un dato pari a zero.

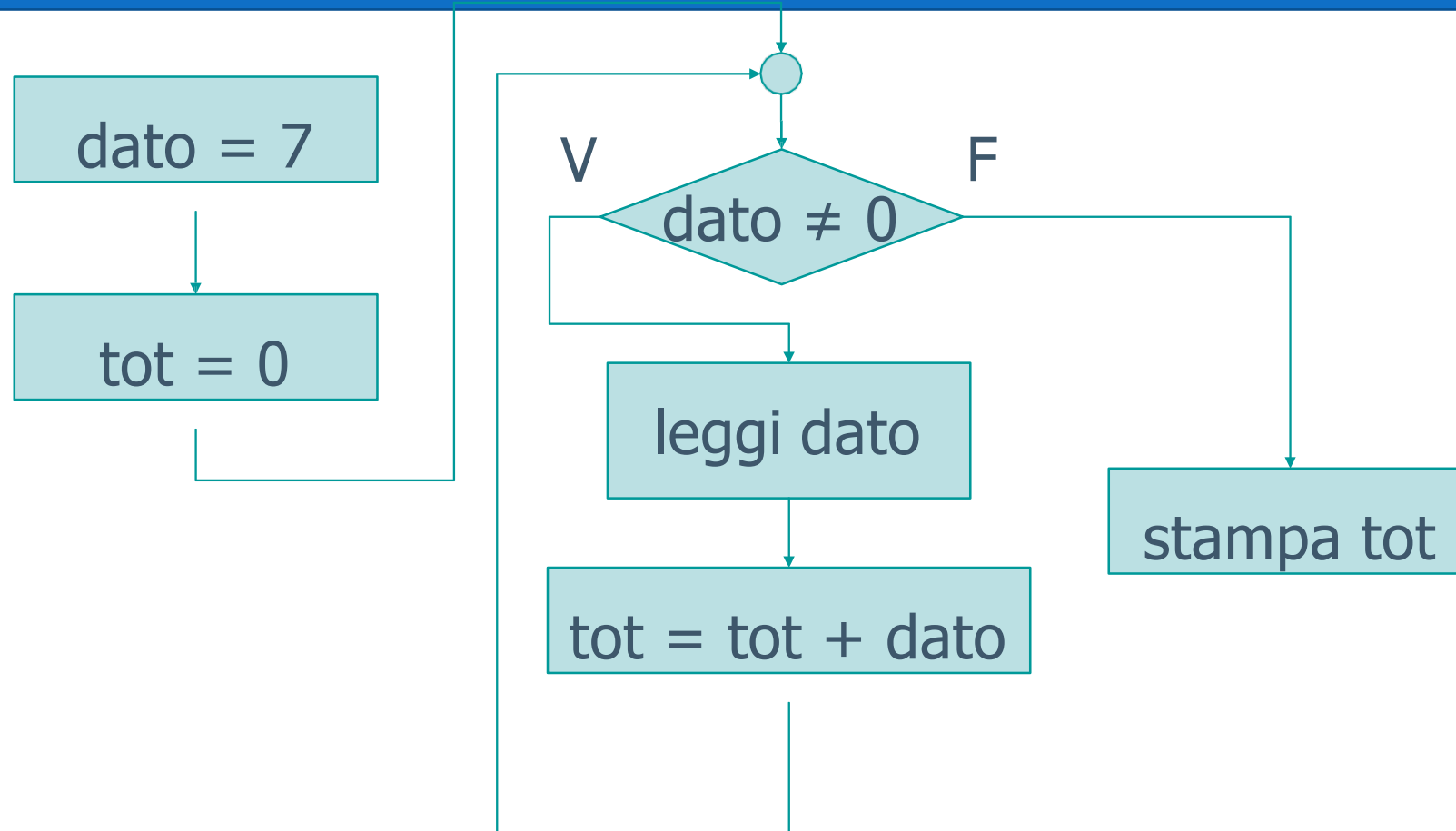
Soluzione parziale



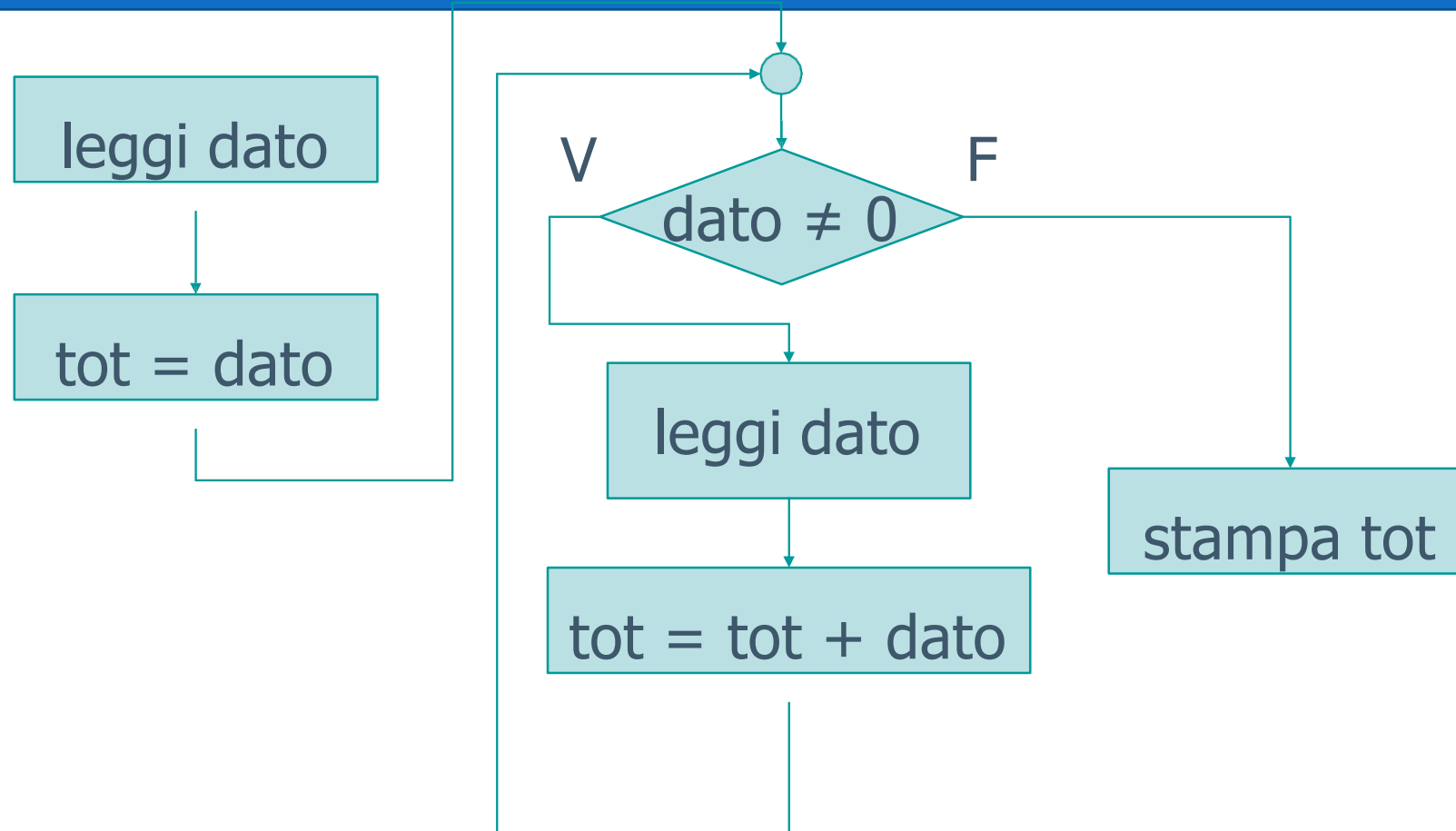
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



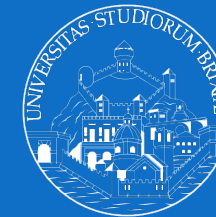
Soluzione 1



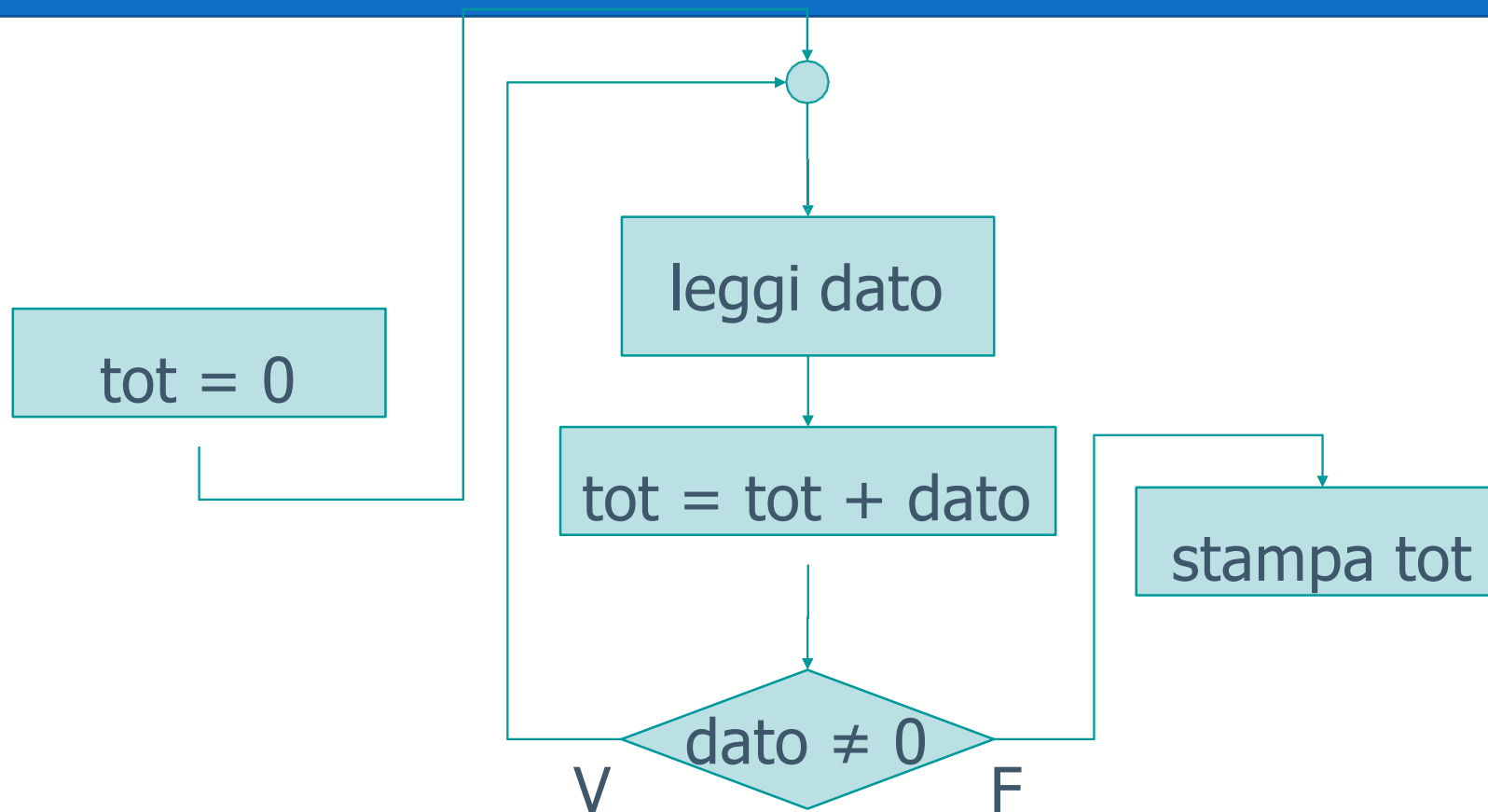
Soluzione 2



Soluzione 3



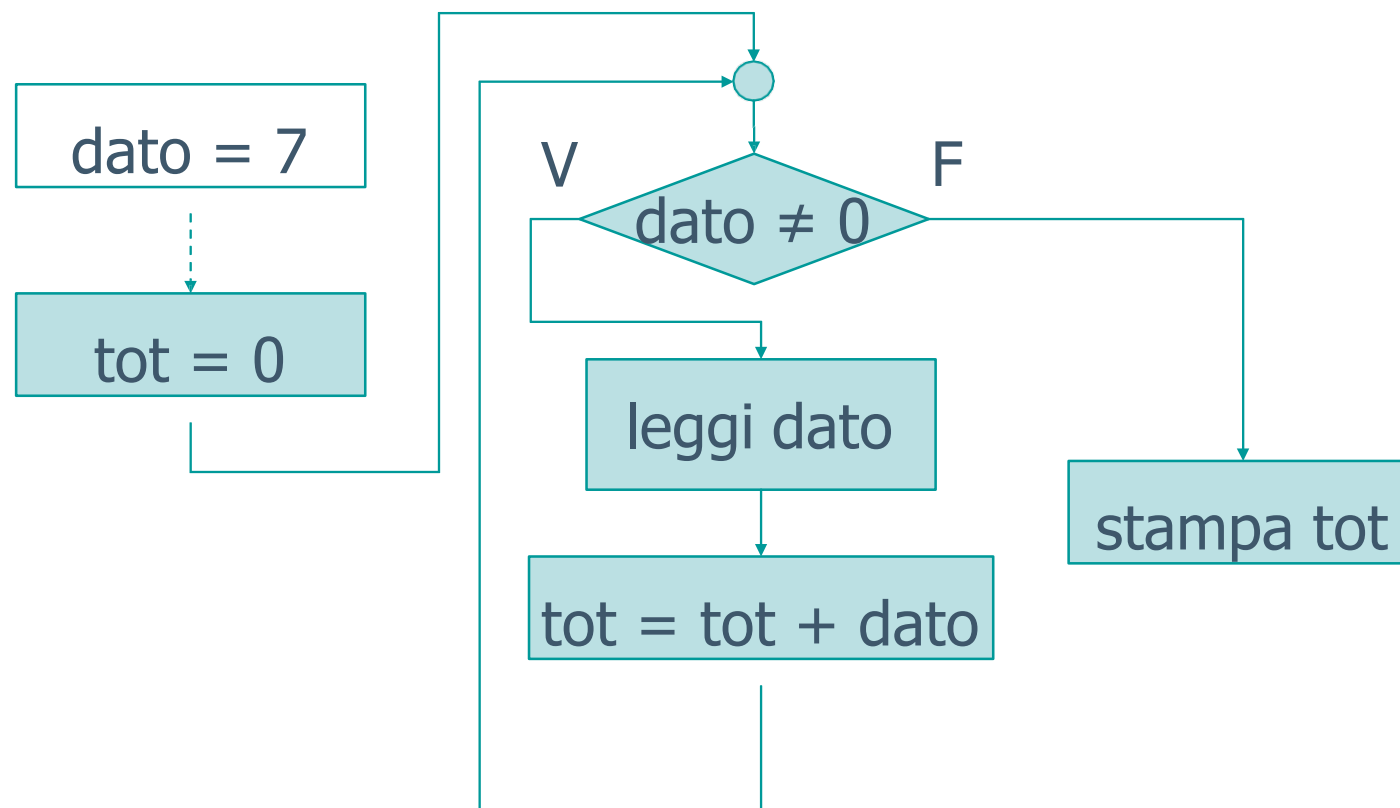
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



Errore frequente



- Dimenticare l'inizializzazione di una variabile utilizzata all'interno del ciclo

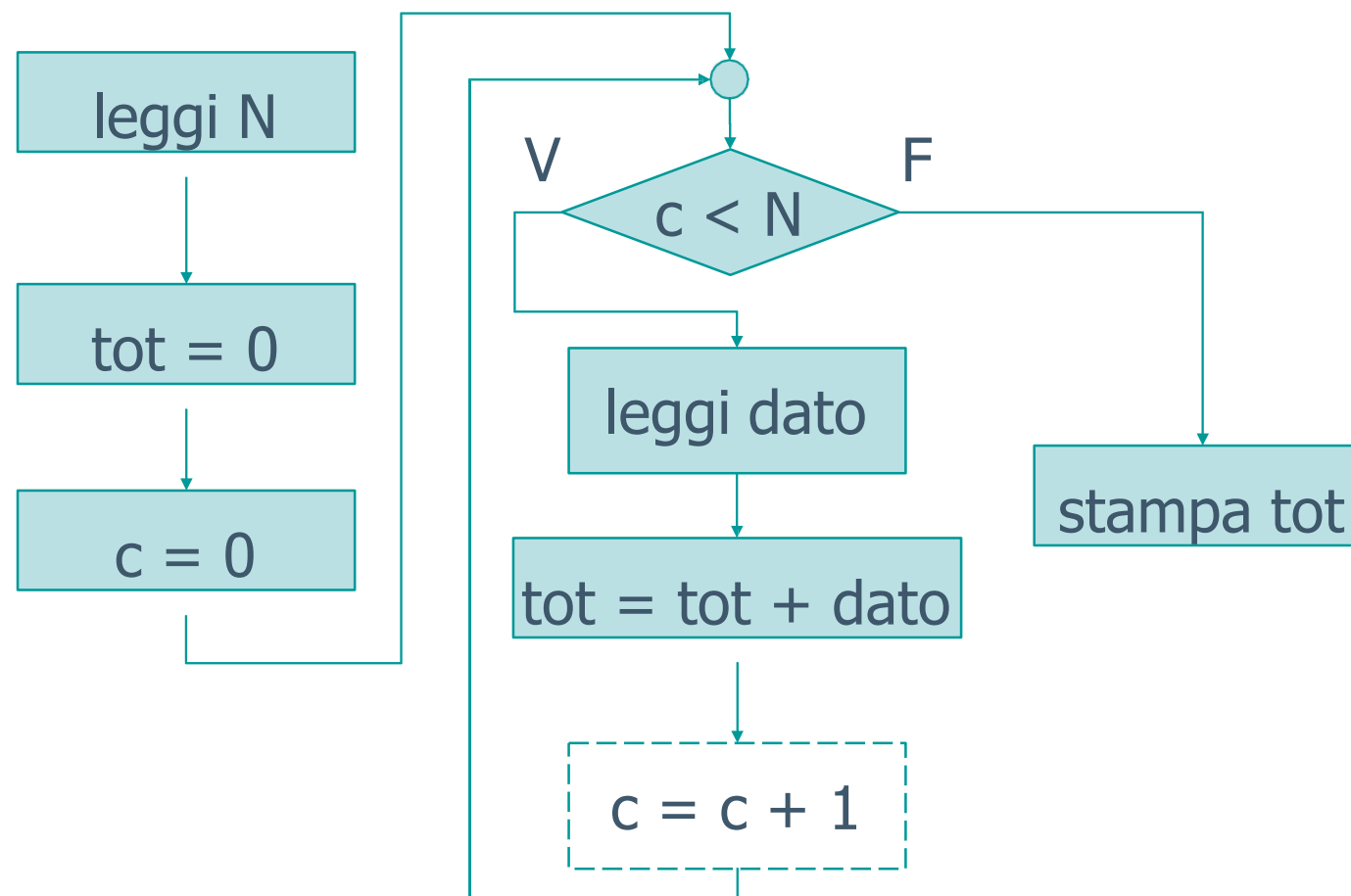


Errore frequente

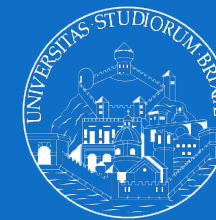


UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

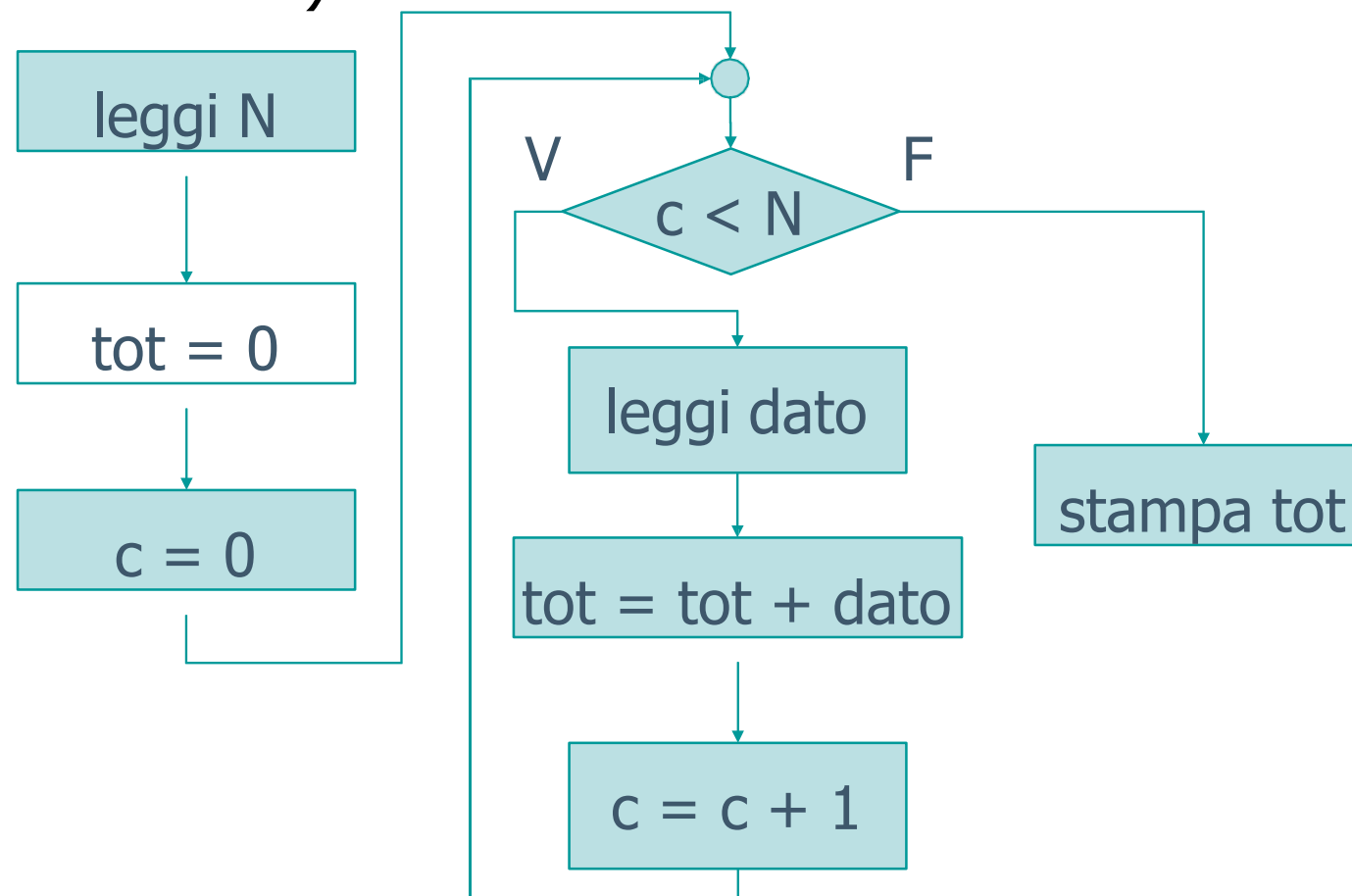
- Dimenticare l'incremento della variabile contatore



Errore frequente



- Dimenticare di inizializzare le altre variabili (oltre al contatore)



Istruzione while



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Sintassi dell'istruzione
- Esercizio "Media aritmetica"
- Esecuzione del programma
- Cicli while e annidati
- Esercizio "Quadrato"

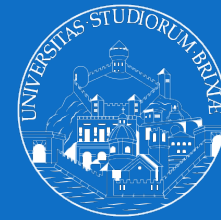
Istruzioni di ripetizione in C



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

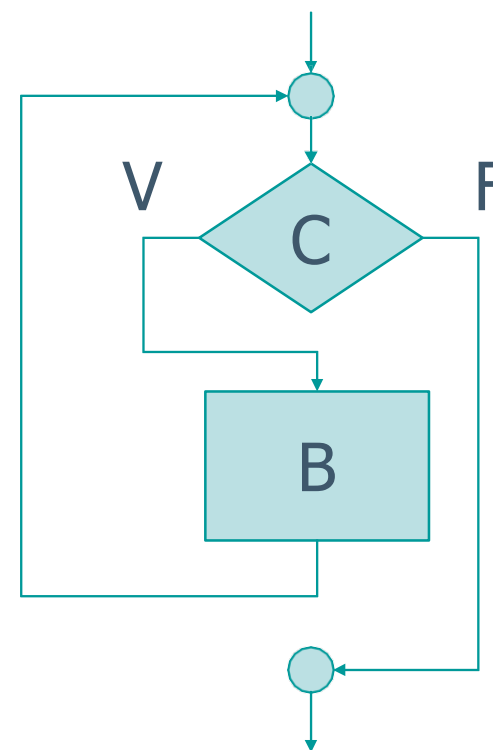
- Nel linguaggio C esistono tre distinte istruzioni di iterazione
 - `while`
 - `do-while`
 - `for`
- La forma più generale è l'istruzione di tipo `while`
- L'istruzione `do-while` si usa in taluni contesti (es. controllo errori di input)
- L'istruzione `for` è usatissima, soprattutto per numero di iterazioni noto

Istruzione while



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

```
while ( C )  
{  
    B ;  
}
```



```
while ( C )  
{  
    B ;  
}
```

1. Valuta la condizione **C**
2. Se **C** è falsa, salta completamente l'iterazione e vai all'istruzione che segue la }
3. Se **C** è vera, esegui una volta il blocco di istruzioni **B**
4. Al termine del blocco **B**, ritorna al punto 1. per rivalutare la condizione **C**

Numero di iterazioni note



```
int i, N ;  
  
i = 0 ;  
while ( i < N )  
{  
    /* Corpo dell'iterazione */  
    ...  
  
    i = i + 1 ;  
}
```

Esempio



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

Esempio



```
int i, n ;  
float f ;  
.... /* leggi n */  
  
...  
  
i = 2 ;  
f = 1.0 ;  
while ( f i* <= n )  
{ i = i + 1 ;  
}  
printf("Fattoriale di %d = %f\n",  
      n, f);
```

- Nel caso in cui il corpo del `while` sia composto di una sola istruzione, si possono omettere le parentesi graffe
 - Non succede quasi mai

```
while ( C )  
{  
    B ;  
}
```



```
while ( C )  
    B ;
```

Annidamento di cicli



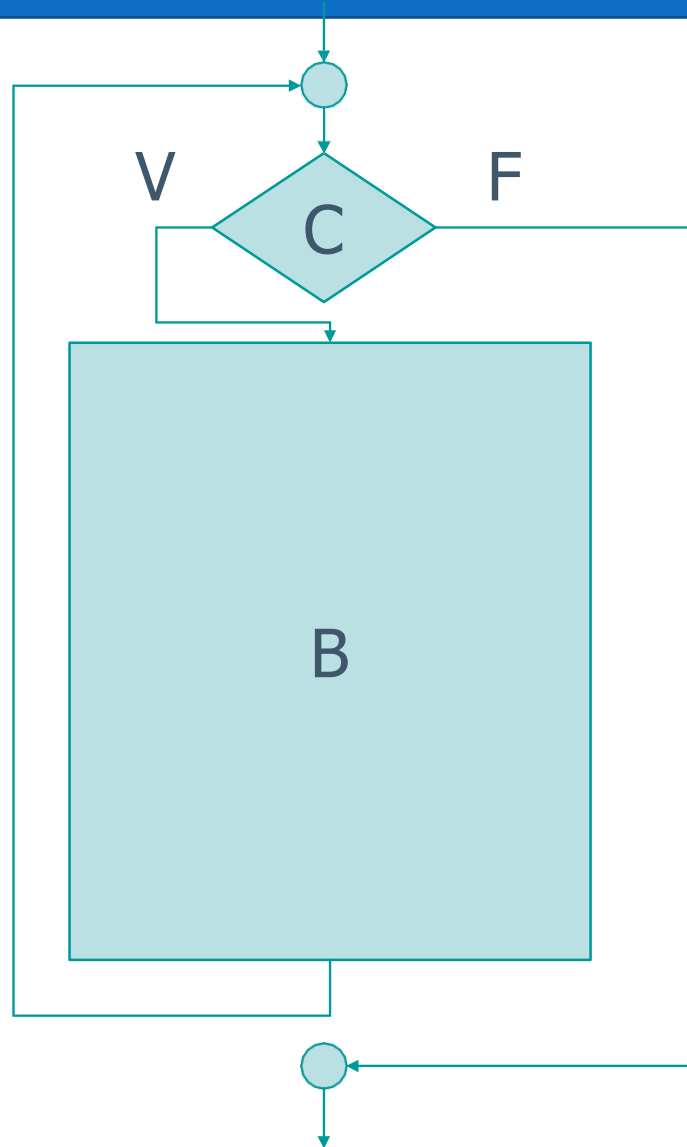
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- All'interno del corpo del ciclo while è possibile racchiudere qualsiasi altra istruzione C
- In particolare, è possibile racchiudere un'istruzione while all'interno di un'altra istruzione while
- In tal caso, per ogni singola iterazione del ciclo while più esterno, vi saranno tutte le iterazioni previste per il ciclo più interno

Cicli while annidati



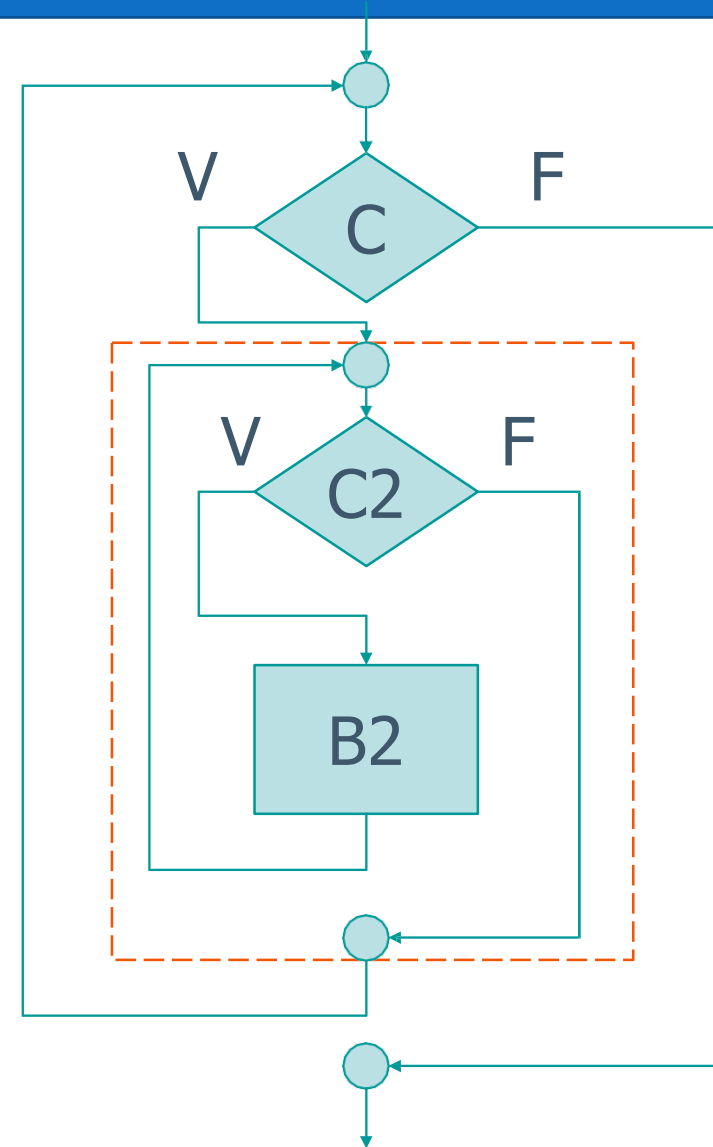
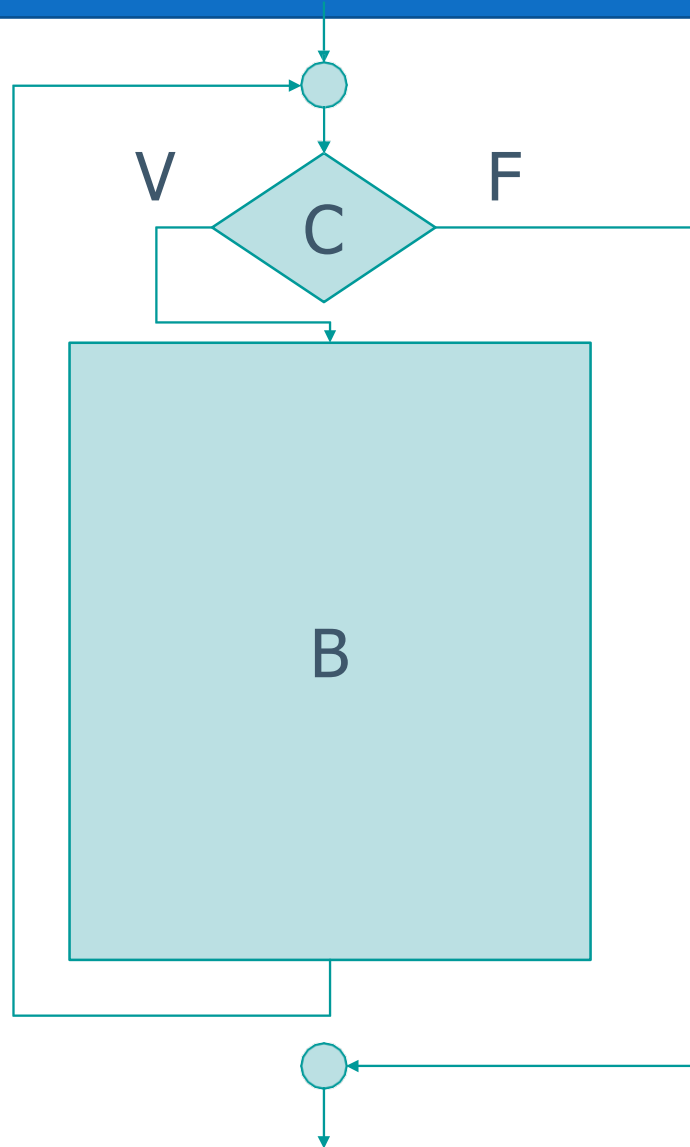
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



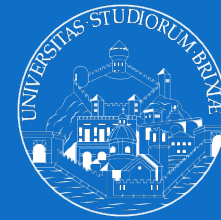
Cicli while annidati



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

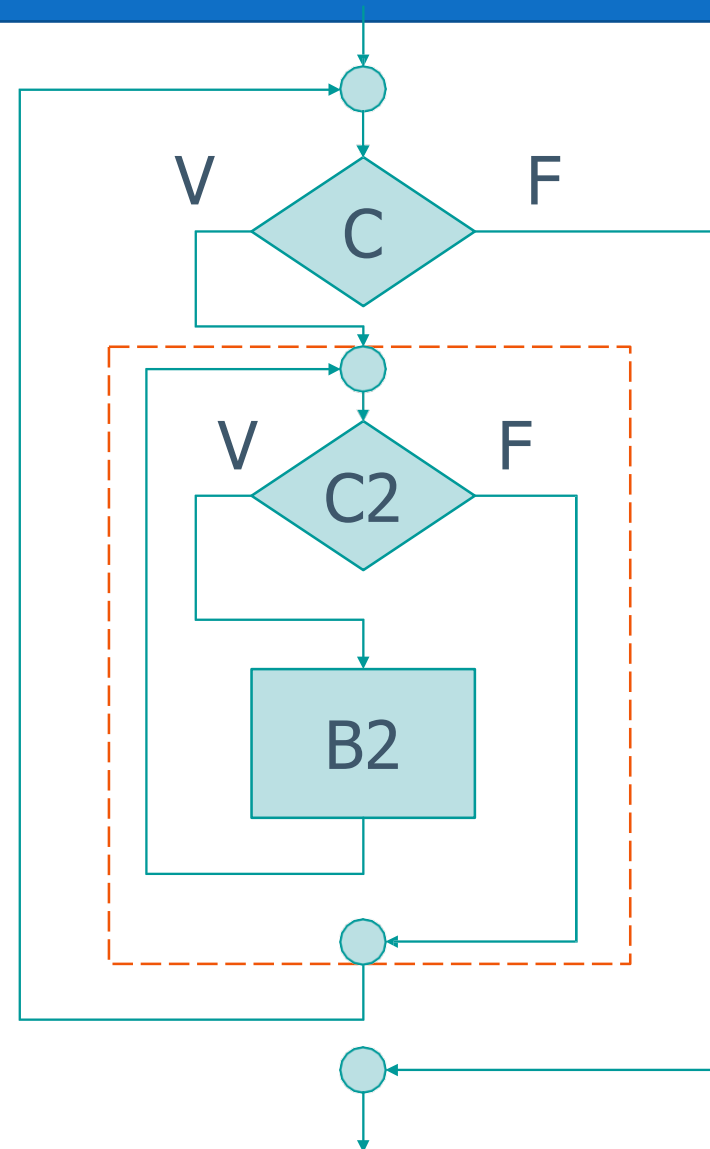


Cicli while annidati



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

```
while( c )  
{  
    while( c2 )  
    {  
        B2 ;  
    }  
}
```



Esempio




```
i = 0 ;  
while( i < N )  
{  
    j = 0 ;  
    while( j < N )  
    {  
        printf("i=%d - j=%d\n", i, j);  
  
        j = j + 1 ;  
    }  
  
    i = i + 1 ;  
}
```

Esempio



```
i = 0 ;  
while( i < N )  
{  
    j = 0 ;  
    while( j < N )  
    {  
        printf("i=%d - j=%d\n", i, j);  
        j = j + 1 ;  
    }  
    i = i + 1 ;  
}
```



```
i=0 - j=0  
i=0 - j=1  
i=0 - j=2  
i=1 - j=0  
i=1 - j=1  
i=1 - j=2  
i=2 - j=0  
i=2 - j=1  
i=2 - j=2
```

Schemi ricorrenti nei cicli



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Contatori
- Accumulatori
- Flag
- Esistenza e universalità

- Spesso in un ciclo è utile sapere
 - Quante iterazioni sono state fatte
 - Quante iterazioni rimangono da fare
 - Quale numero di iterazione sia quella corrente
- Per questi scopi si usano delle “normali” variabili intere, dette **contatori**
 - Inizializzate prima del ciclo
 - Incrementate/decrementate ad ogni iterazione
 - Oppure incrementate/decrementate ogni volta che si riscontra una certa condizione

Contatori



```
int i  N ;  
i = 0 ;  
while ( i < N )  
{  
    printf("Iterazione %d\n", i+1) ;  
    i = i + 1 ;  
}
```

- Sintassi dell'istruzione
- Operatori di autoincremento
- Cicli for annidati

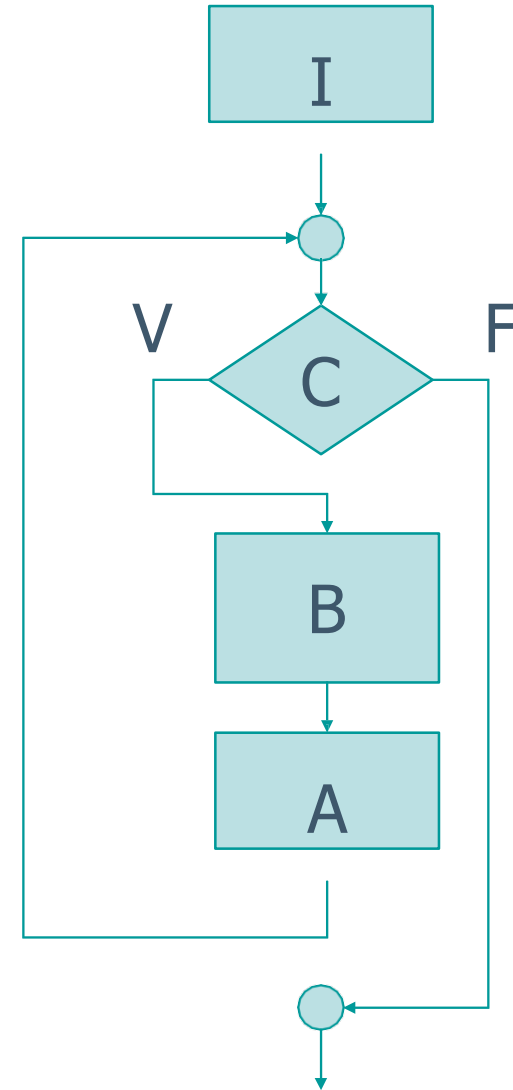
- L'istruzione fondamentale è while
- La condizione solitamente valuta una variabile di controllo
- Occorre ricordarsi l'inizializzazione della variabile di controllo
- Occorre ricordarsi di aggiornare (incrementare, ...) la variabile di controllo
- L'istruzione for rende più semplice ricordare queste cose

Istruzione for

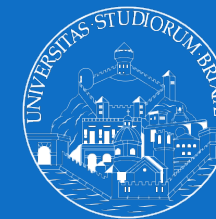


UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

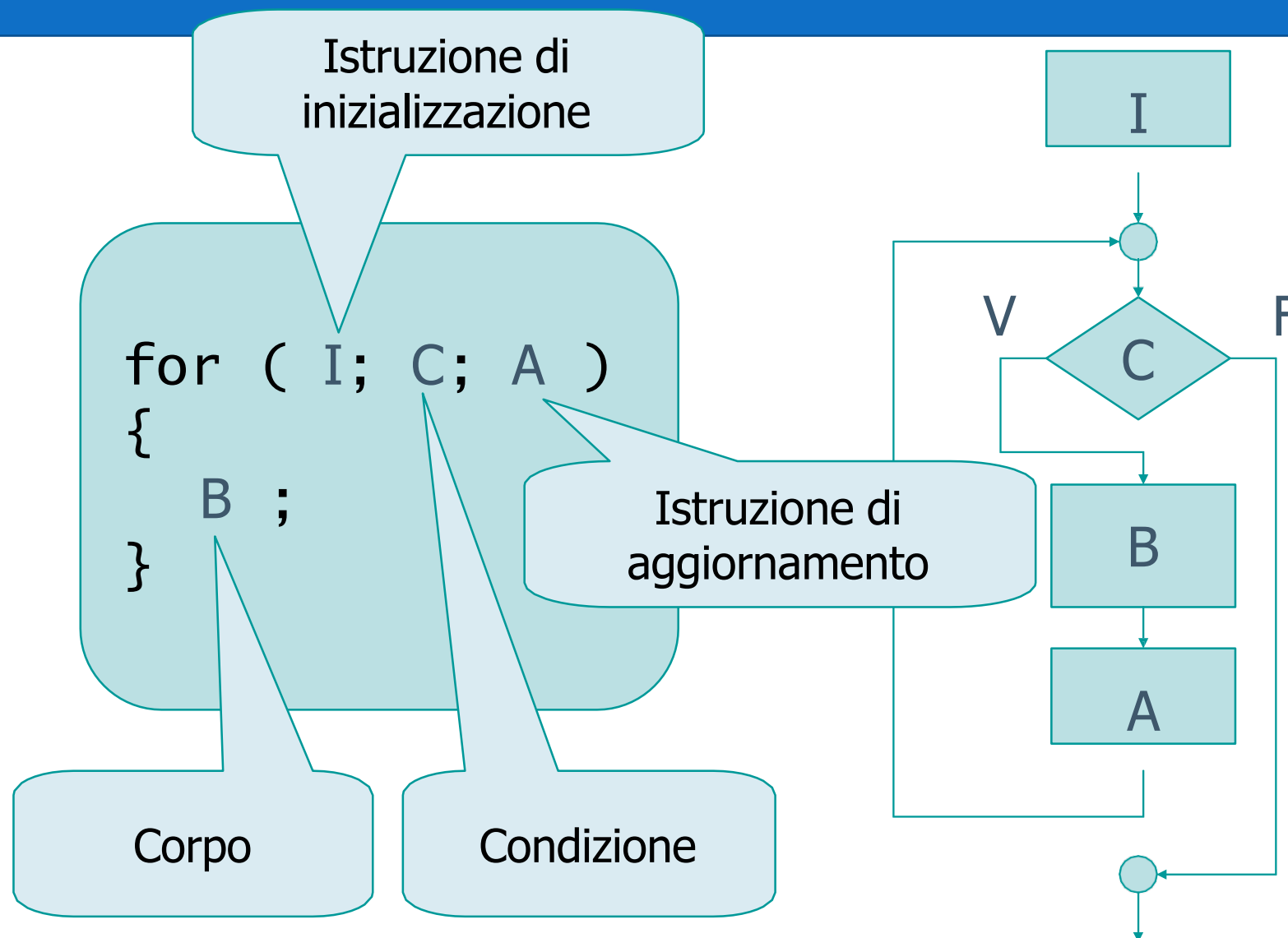
```
for ( I; C; A )  
{  
    B ;  
}
```



Istruzione for



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



Esempio



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA



num1-10v2.c

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

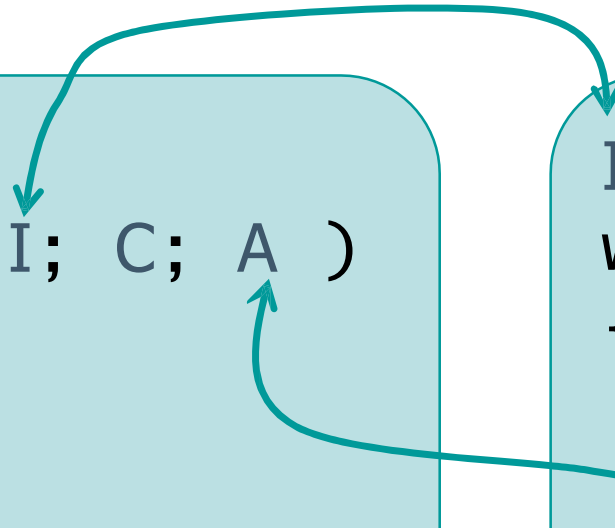
Equivalenza for/while



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

```
for ( I; C; A )  
{  
    B ;  
}
```

```
I ;  
while ( C )  
{  
    B ;  
    A ;  
}
```



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

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

Utilizzo prevalente (1/2)



- Le istruzioni di inizializzazione **I** e di aggiornamento **A** possono essere qualsiasi
- Solitamente **I** viene utilizzata per inizializzare il contatore di controllo del ciclo, e quindi è del tipo
 - $i = 0$
- Solitamente **A** viene utilizzata per incrementare (o decrementare) il contatore, e quindi è del tipo
 - $i = i + 1$

```
for ( I; C; A )  
{  
    B ;  
}
```

Utilizzo prevalente (2/2)



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- L'istruzione `for` può sostituire un qualsiasi ciclo `while`
- Solitamente viene utilizzata, per maggior chiarezza, nei cicli con numero di iterazioni noto a priori

Cicli for con iterazioni note



```
int i ;  
  
for ( i=0; i<N; i=i+1 )  
{  
    .....  
}
```

```
int i ;  
  
for ( i=1; i<=N; i=i+1 )  
{  
    .....  
}
```

```
int i ;  
  
for ( i=N; i>0; i=i-1 )  
{  
    .....  
}
```

```
int i ;  
  
for ( i=N-1; i>=0; i=i-1 )  
{  
    .....  
}
```