Linguaggio C: strutture di controllo

Libro cap. 3 e 4

Moreno Marzolla
Dipartimento di Informatica—Scienza e Ingegneria (DISI)
Università di Bologna
https://www.moreno.marzolla.name/

Copyright © 2008 Stefano Mizzaro
http://users.dimi.uniud.it/~stefano.mizzaro/dida/Prog0708/
Copyright © 2017, 2020 Moreno Marzolla
https://www.moreno.marzolla.name/teaching/FINFA/

This work is licensed under the Creative Commons Attribution-Non Commercial 2.0 (CC BY-NC 2.0) License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/2.0/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Ringraziamenti

- prof. Stefano Mizzaro, Università di Udine
  - http://users.dimi.uniud.it/~stefano.mizzaro/

Strutture di controllo

- Un programma è fatto da istruzioni
- Abbiamo visto
  - Dichiarazioni di variabile (con nome e tipo)
  - Assegnamento
  - Espressioni
  - **`printf()`**
  - …
- Vediamo come metterli insieme

Le strutture di controllo

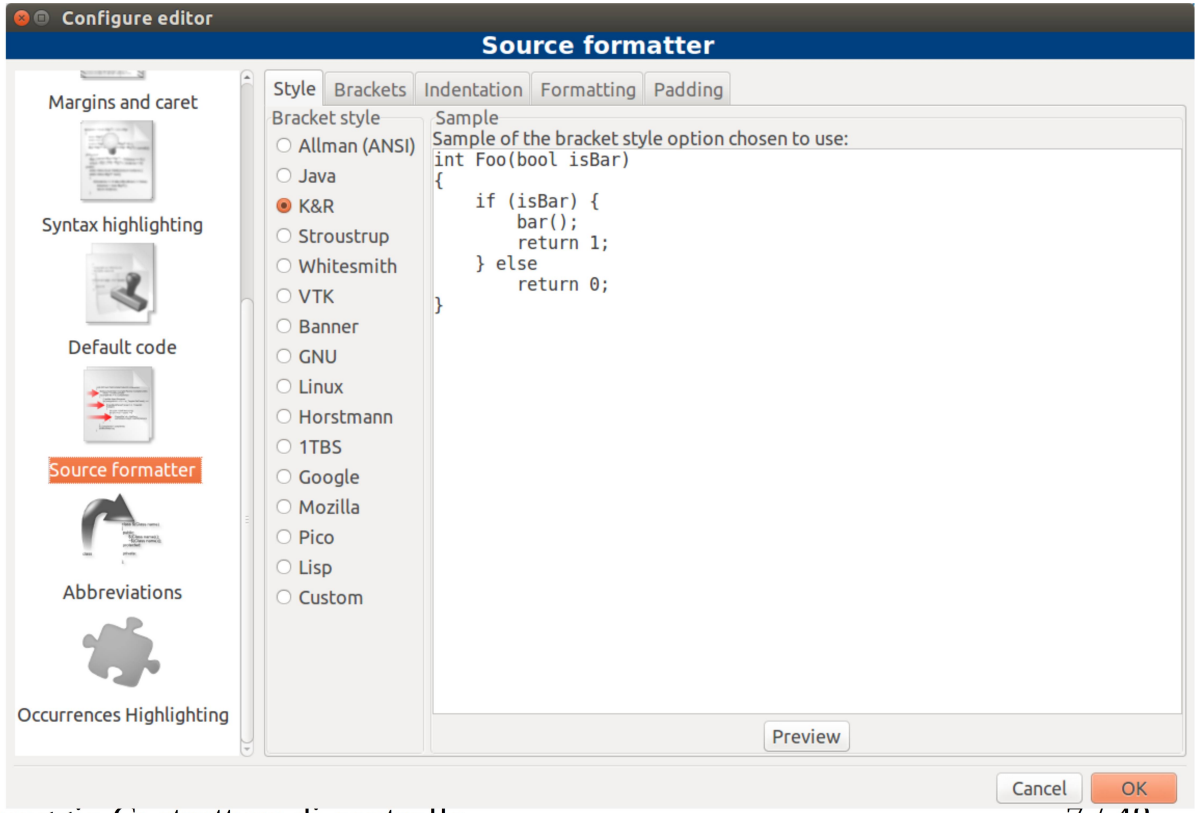
- Sequenza
  - `;` e `{}`
- Selezione
  - `if`, `if/else`, `switch/case`
- Iterazione (ripetizione)
  - `while`, `do/while`, `for`
  - `break`, `continue`

Sequenza

- Ogni istruzione termina con un punto e virgola `;`
- Si possono raggruppare più istruzioni "*per farle diventare come un'unica istruzione*" con le parentesi graffe `{ }`
  - Dopo la `}` di chiusura di un blocco non serve la virgola
- Incolonnare per leggibilità
  - Aggiungere spazi (es. 4) dopo ogni `{`
  - Togliere spazi prima di ogni `}`
  - `{` a fine riga
  - `}` su riga isolata (qualche eccezione…)

CodeBlocks vi aiuta

- *Settings* → *Editor…* → *Source Formatter* → selezionare "*K&R*"
- Per riformattare il proprio programma:
  Edit → *Select All*, e poi *Plugin* → Source Code Formatter (AStyle)

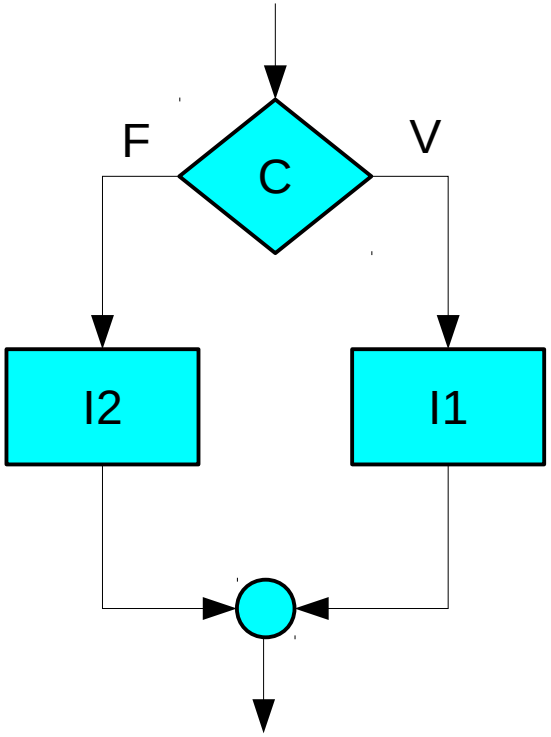


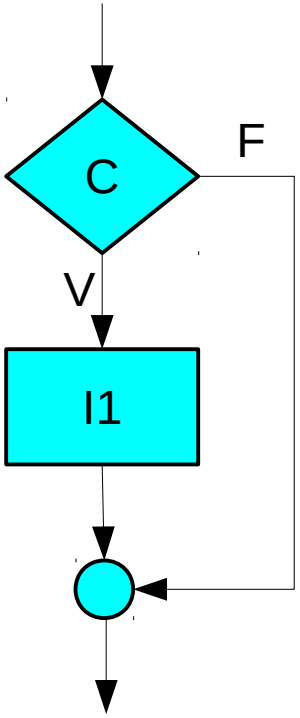
if, if/else

```
if (C) {
   I1;
} else {
   I2;
}
```

NB: parentesi graffe **opzionali** se I1o I2 sono singole istruzioni

```
if (C) {
   I1;
}
```





if annidati

- **`if`** dentro a **`if`** (dentro a **`if`**…)
- Nel ramo **`if`** o nel ramo **`else`**
  - Annidamenti **`if`** e annidamenti **`else`**
- Fare attenzione
  - L'**`else`** si riferisce all'ultimo **`if`**
- Usare sempre le { } in modo da non sbagliare

Esempio

```
if (x > 0)
  if (x < 10)
    printf("positivo e minore di 10\n");
```

Equivale a...

```
if (x > 0) {
  if (x < 10) {
    printf("positivo e minore di 10\n");
  }
}
```

Meglio...

```
if ((x > 0) && (x < 10)) {
    printf("positivo e minore di 10\n");
}
```

Esempio

```
if (x > 0)
  if (x < 10)
    printf("positivo e minore di 10\n");
  else
    printf("positivo e maggiore o = a 10\n");
```

Equivale a

```
if (x > 0) {
  if (x < 10) {
    printf("positivo e minore di 10\n");
  } else {
    printf("positivo e maggiore o = a 10\n");
  }
}
```

Esercizio

- La pressione sistolica ("massima") *P* di un adulto viene generalmente classificata come segue:
  - $100 \leq P \leq 139$ → *normale*
  - $P < 100$ → ipotensione (di vario grado)
  - $P > 139$ → ipertensione (di vario grado)
- Scrivere un programma che accetta in input un intero *P* e stampa "ipotensione", "normale" oppure "ipertensione" in base al valore di *P*

Attenzione

l'indentazione errata può ingannare

```
if (x > 0)
  if (x < 10)
    printf("positivo e minore di 10\n");
else
  printf("positivo e maggiore o = a 10\n");
```

Equivale a

```
if (x > 0) {
  if (x < 10) {
    printf("positivo e minore di 10\n");
  } else {
    printf("positivo e maggiore o = a 10\n");
  }
}
```

Esercizi

- Per ciascuno dei frammenti di codice seguenti, identificare un valore delle variabili (sono tutte **`int`**) che causa la stampa di "AAA"

```
if ((a < 10) && (a > 15)) {
    printf("AAA\n");
} else {
    printf("BBB\n");
}
```

```
if (a < 10) {
    if ((a > 15 || (a == -9)) {
        printf("BBB\n");
    } else {
        printf("AAA\n");
    }
}
```

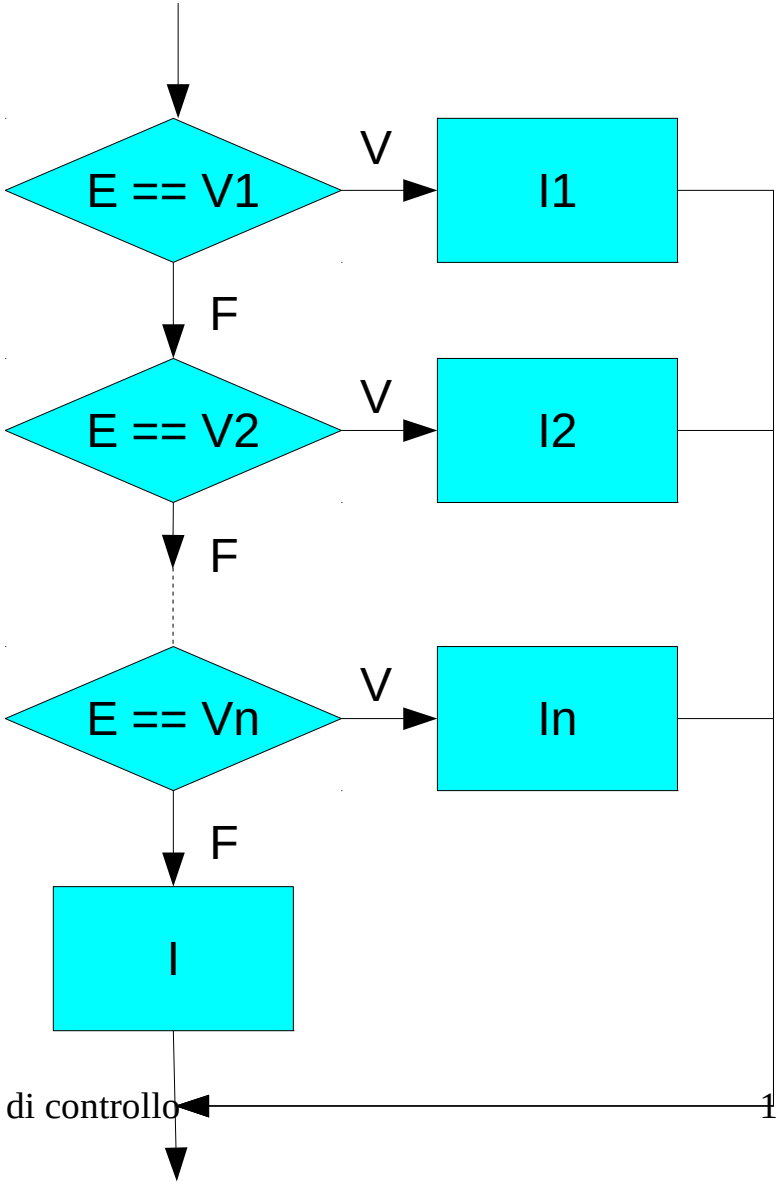
```
if ((a < b) && ((a > 7) || !b)) {
    printf("BBB\n");
} else {
    printf("AAA\n");
}
```

Selezione *n*-aria: switch/case

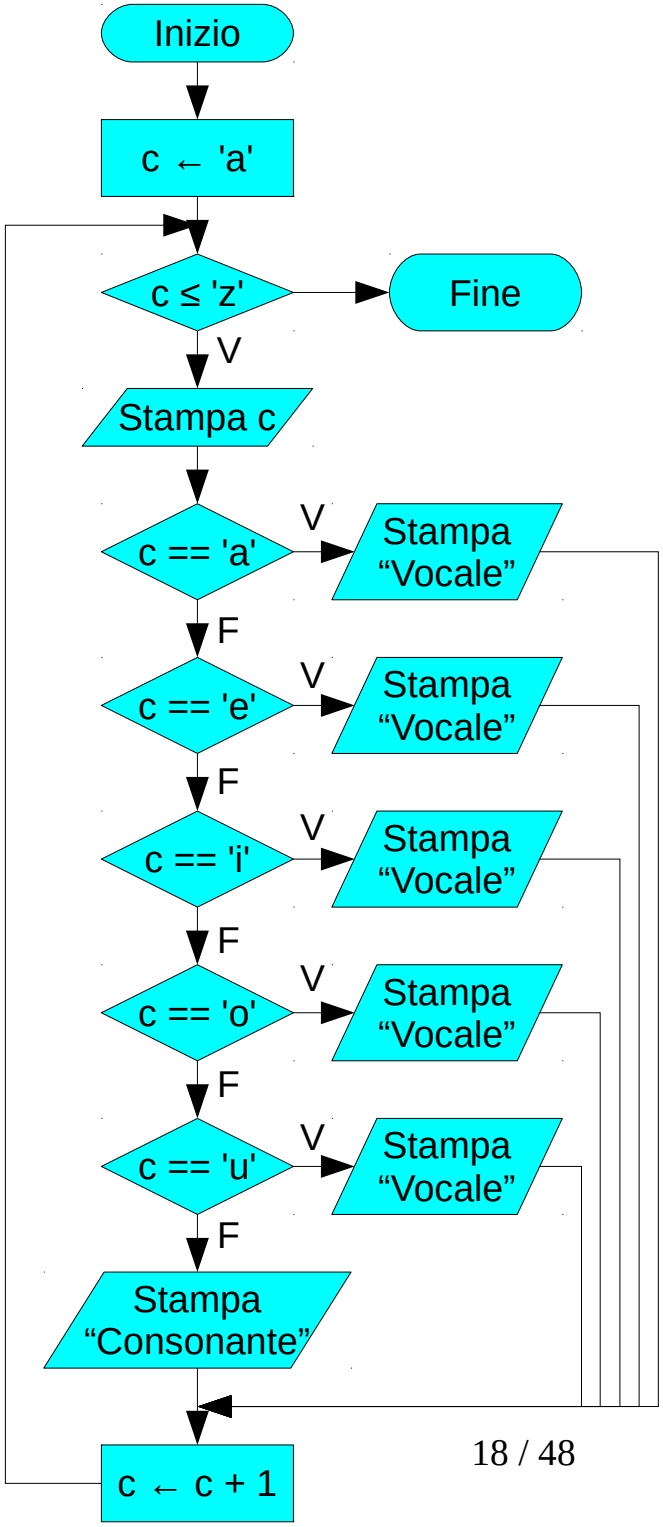
- Selezione fra più alternative
- Forma generale:

```
switch (E) {
  case V1: I1; break;
  case V2: I2; break;
  ...
  case Vn: In; break;
  default: I;  break;
}
```

- Il caso “default:” può essere omesso



```c
/* alfabeto.c : per ogni lettera minuscola dell'alfabeto, stampa se
e' una vocale o una consonante */

#include <stdio.h>

int main( void )
{
    char c = 'a';
    while (c <= 'z') {
        printf("%c : ", c);
        switch (c) {
        case 'a':
            printf("Vocale\n"); break;
        case 'e':
            printf("Vocale\n"); break;
        case 'i':
            printf("Vocale\n"); break;
        case 'o':
            printf("Vocale\n"); break;
        case 'u':
            printf("Vocale\n"); break;
        default:
            printf("Consonante\n"); break;
        }
        c = c + 1;
    }
    return 0;
}
```



Stesso programma usando solo if-else

```
#include <stdio.h>
int main( void )
{
    char c = 'a';
    while (c <= 'z') {
        printf("%c : ", c);
        if (c == 'a') {
            printf("Vocale\n");
        } else {
            if (c == 'e') {
                printf("Vocale\n");
            } else {
                if (c == 'i') {
                    printf("Vocale\n");
                } else {
                    if (c == 'o') {
                        printf("Vocale\n");
                    } else {
                        if (c == 'u') {
                            printf("Vocale\n");
                        } else {
                            printf("Consonante\n");
                        }
                    }
                }
            }
        }
        c = c + 1;
    }
    return 0;
}
```

Meglio...

```
#include <stdio.h>

int main( void )
{
    char c = 'a';
    while (c <= 'z') {
        printf("%c : ", c);
        if ((c == 'a') || (c == 'e') || (c == 'i') ||
             (c == 'o') || (c == 'u')) {
            printf("Vocale\n");
        } else {
            printf("Consonante\n");
        }
        c = c + 1;
    }
    return 0;
}
```

Costrutto *switch*

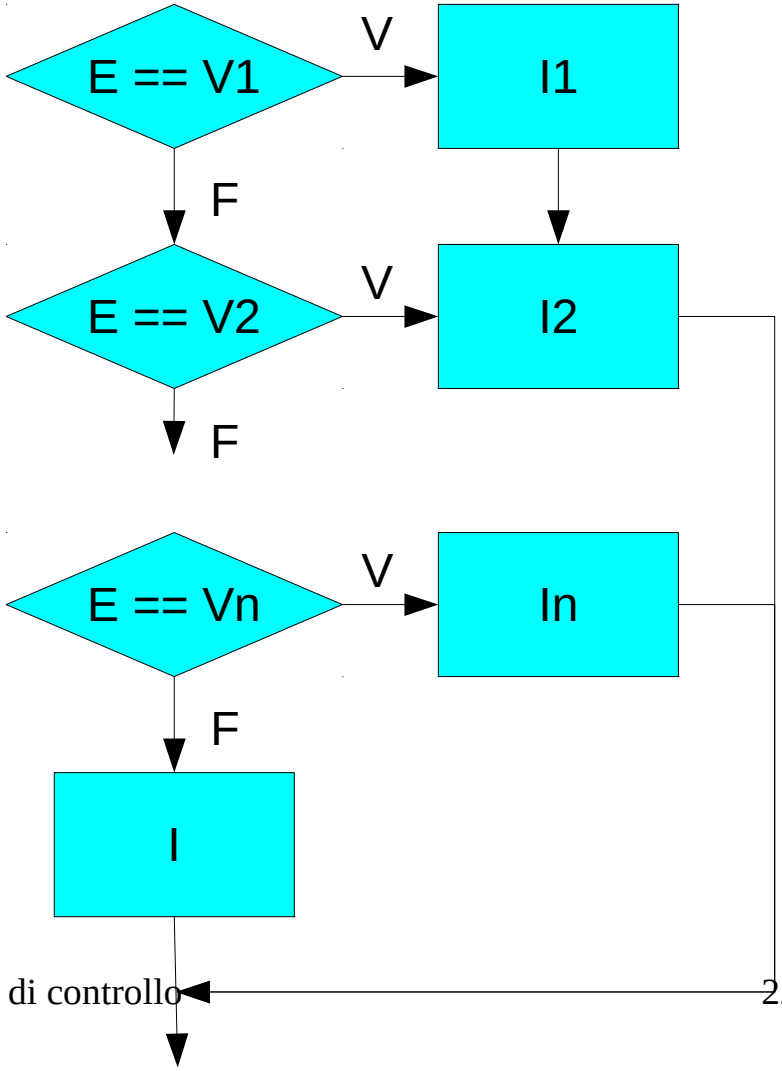
- Le condizioni sono solo del tipo **`E==Vi`**
  - mentre nell'**`if`** possono essere arbitrarie, es. **`E > 0`**
- Le **`Vi`** devono essere espressioni il cui valore deve essere noto a tempo di compilazione
  - cioè un letterale, una costante…
- ...e di tipo **`char`** o **`int`**

Attenzione

- Cosa succede se viene omesso un break?

Manca il break

```
switch (E) {
  case V1: I1;
  case V2: I2; break;
  ...
  case Vn: In; break;
  default: I;  break;
}
```

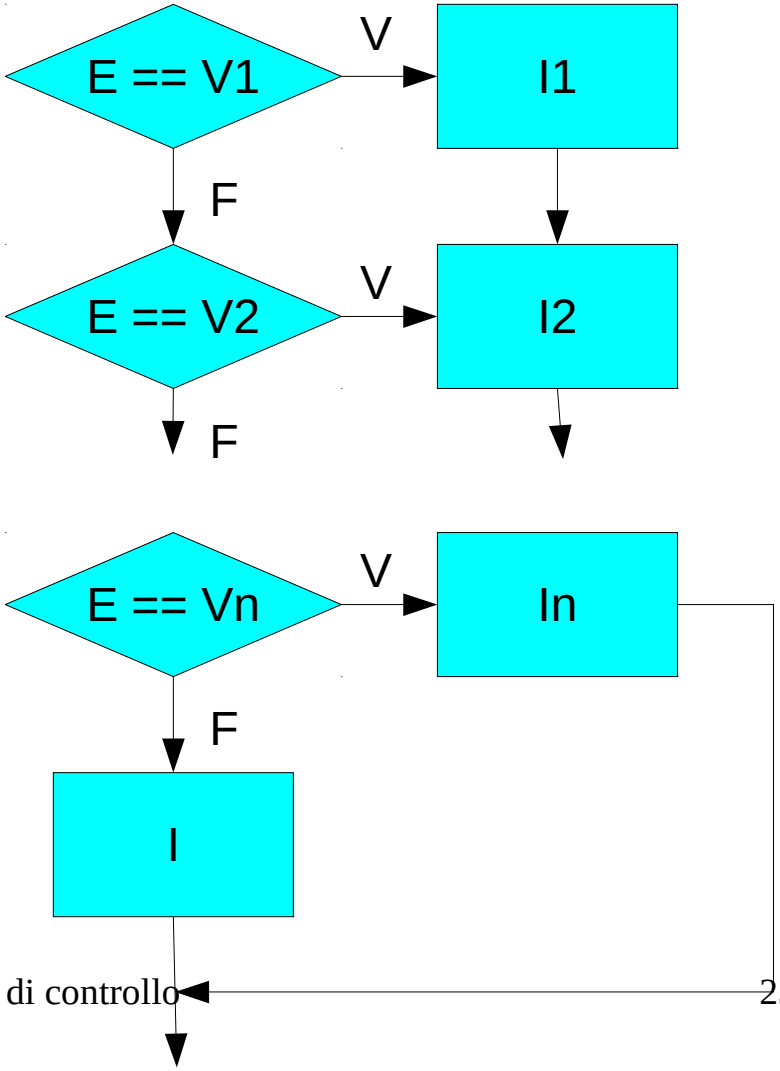


Attenzione

- Cosa succede se viene omesso un break?

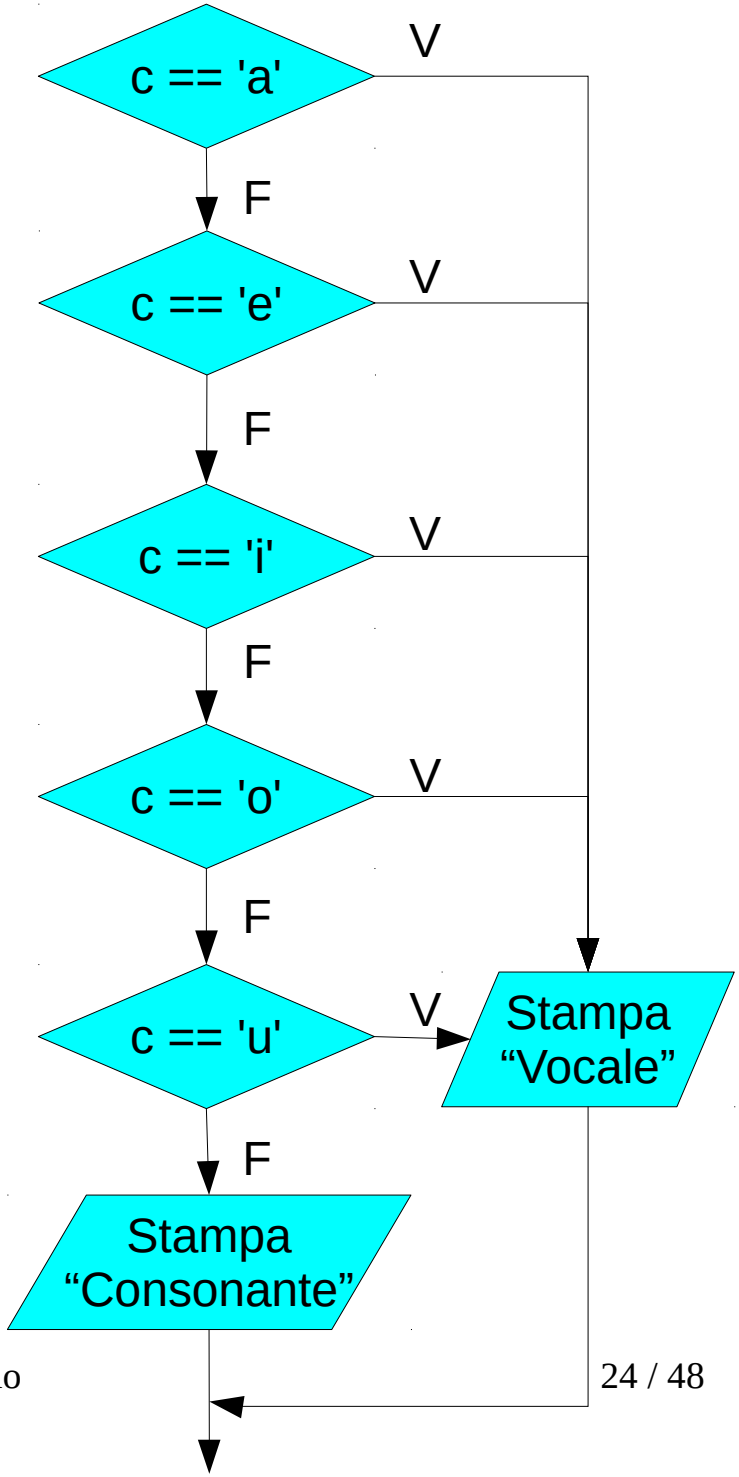
Mancano i break

```
switch (E) {
  case V1: I1;
  case V2: I2;
  ...
  case Vn: In; break;
  default: I;  break;
}
```



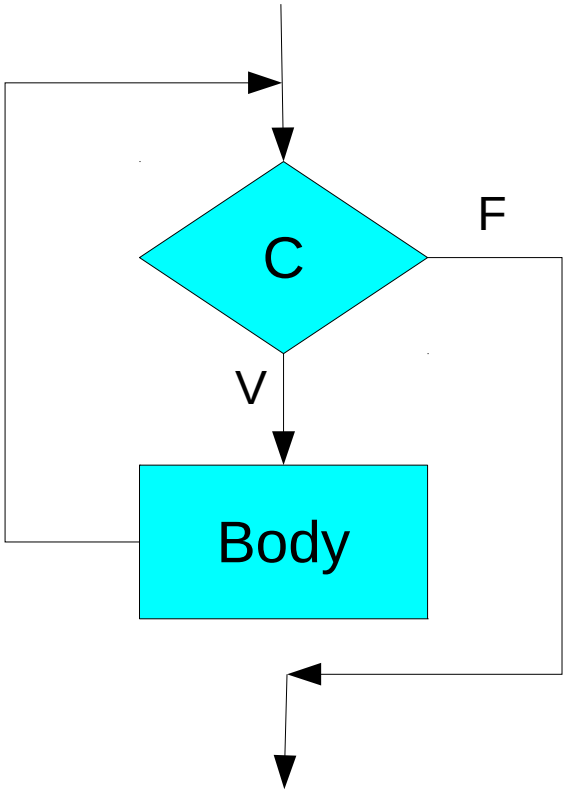
Esempio (più sintetico)

```
switch (c) {
  case 'a':
  case 'e':
  case 'i':
  case 'o':
  case 'u':
    printf("Vocale\n");
    break;
  default:
    printf("Consonante\n");
    break;
}
```

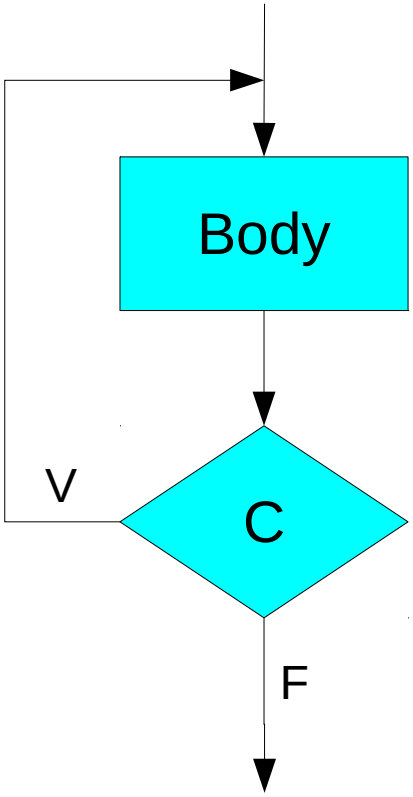


Iterazione
`while` e `do/while`

`while (C) {Body}`



`do {Body} while (C);`



https://www.reddit.com/r/ProgrammerHumor/comments/a5mghb/the_importance_of_knowing_how_to_correctly_use/

Esempio

- Cosa stampano i programmi seguenti?

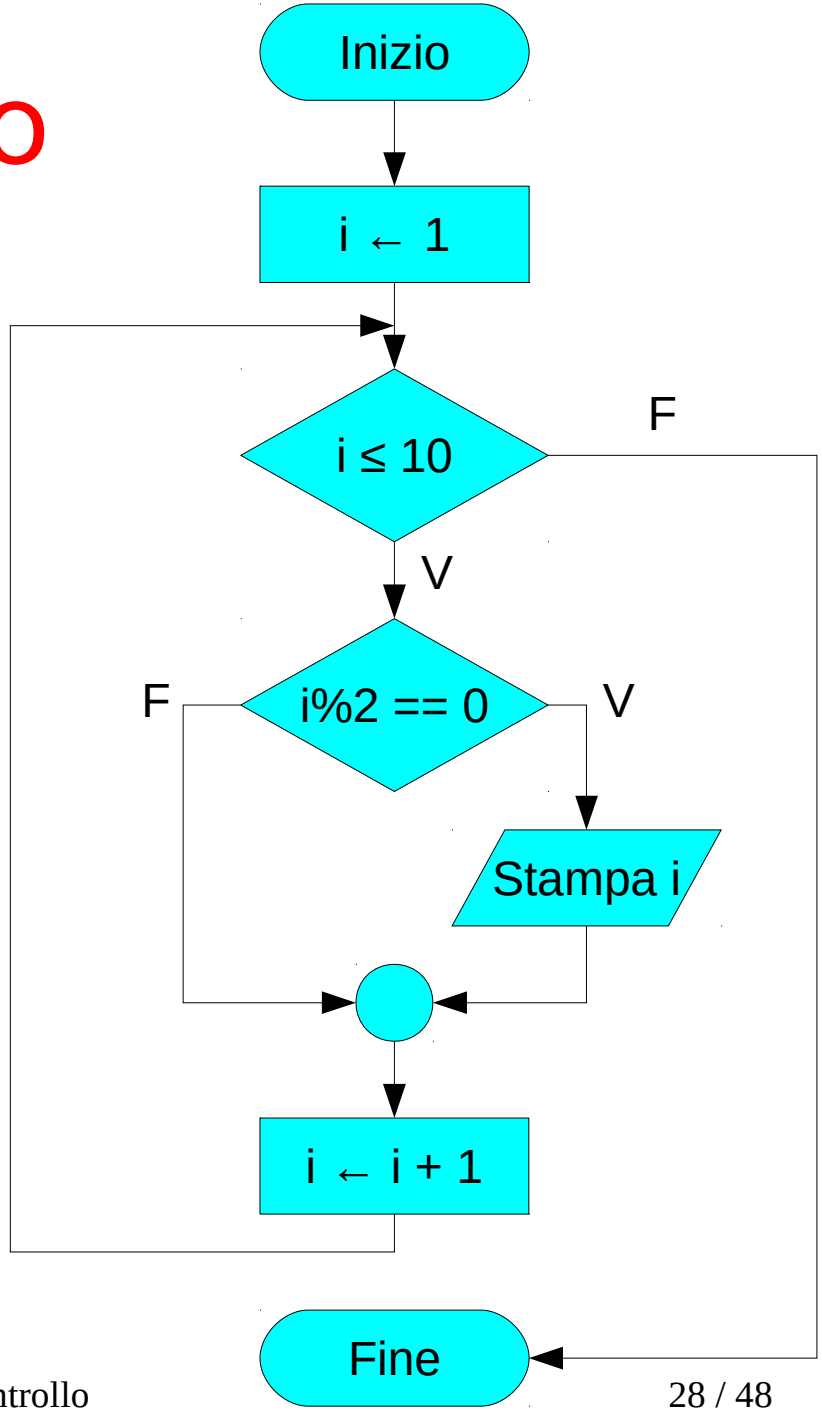
```
/* while-do.c : Esempio di ciclo while */
#include <stdio.h>

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

```
/* do-while.c : Esempio di ciclo do-while */
#include <stdio.h>

int main( void )
{
    int i = 0;
    do {
        printf("%d\n", i);
        i = i + 1;
    } while (i < 0);
    return 0;
}
```

Esempio

- Stampare i numeri pari compresi tra 1 a 10 (estremi inclusi)

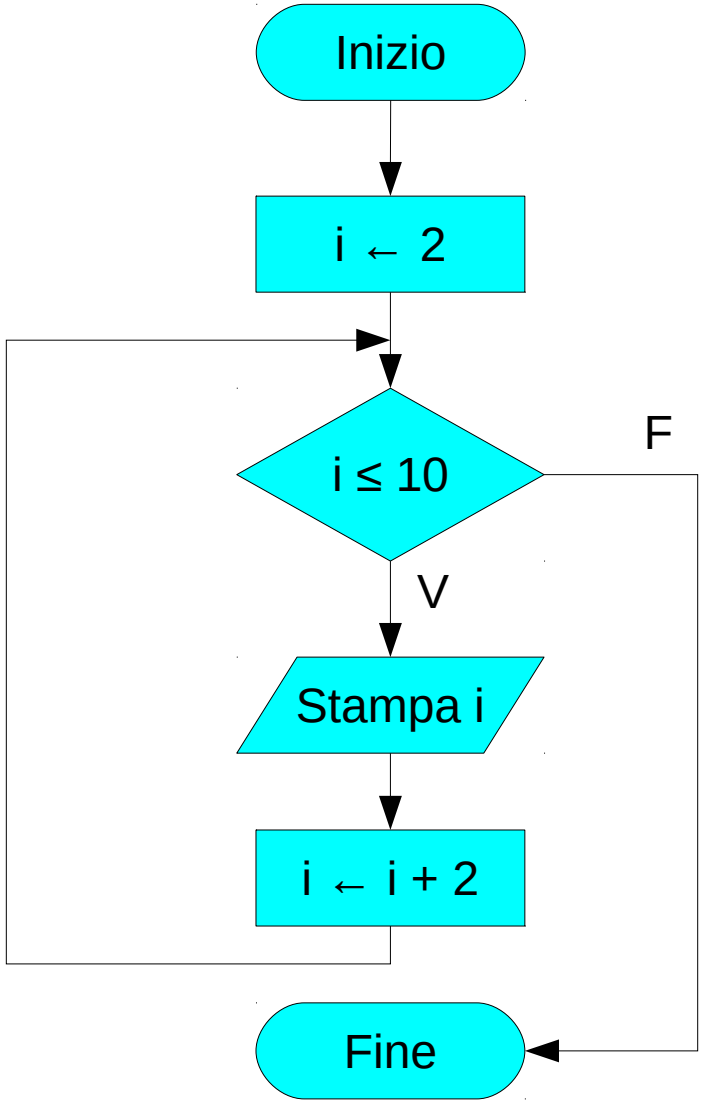
```
#include <stdio.h>

int main( void )
{
    int i = 1;
    while (i <= 10) {
        if (i % 2 == 0) {
            printf("%d\n", i);
        }
        i = i + 1;
    }
    return 0;
}
```



Esempio

- Stampare i numeri pari compresi tra 1 a 10 (estremi inclusi)
  - Ancora meglio: più sintetico, più efficiente
  - Richiede però di ripensare l'algoritmo per sfruttare proprietà dei numeri pari

```
/* num-pari.c - Stampa i numeri pari tra 1 e 10 */
#include <stdio.h>

int main( void )
{
    int i = 2;
    while (i <= 10) {
        printf("%d\n", i);
        i = i + 2; /* oppure i += 2 */
    }
    return 0;
}
```



Errore comune

- Cosa stampa il programma seguente?
  - ATTENZIONE...

```
#include <stdio.h>

int main( void )
{
    int i = 2;

    while (i <= 10)
      printf("%d\n", i);
      i = i + 2;
    printf("Fine stampa\n");
    return 0;
}
```

Ciclo infinito

- Come possiamo realizzare un ciclo infinito?
  - Cioè un ciclo che non termina mai

```
/* ciclo-infinito.c */
#include <stdio.h>

int main( void )
{
    while ( 1 ) {
        printf("Hello, world!\n");
    }
    return 0;
}
```

- Per terminare occorre premere Ctrl + C

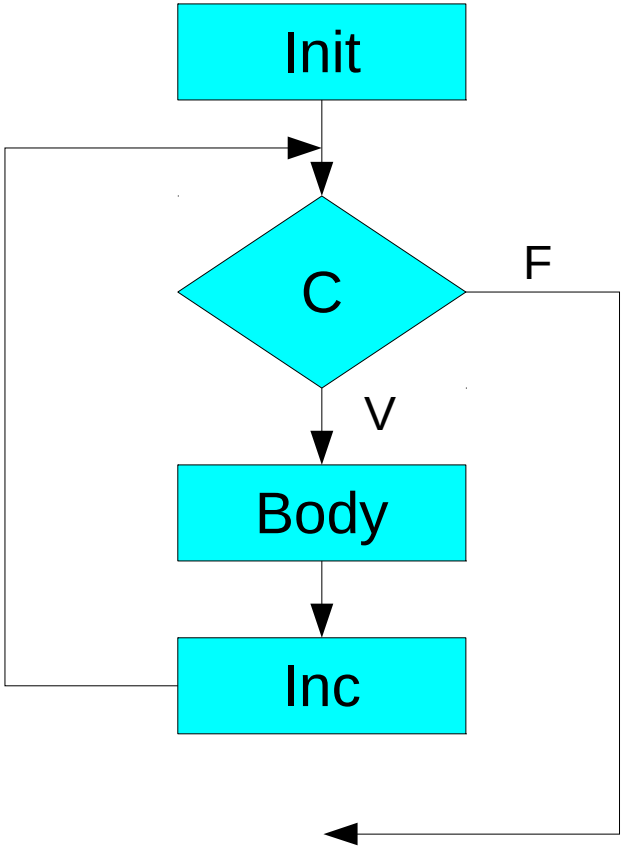
Uso di do-while

- Sebbene i cicli do-while siano raramente utilizzati, esistono dei casi in cui risultano utili. Esempio:
  - Leggere un valore intero compreso tra 1 e 10 (estr. inclusi)
  - Se l'utente digita un valore non valido, si richiede l'input

```
/* chiedi-input.c */
#include <stdio.h>

int main( void )
{
    int val;
    do {
        printf("Inserire un intero compreso tra 1 e 10 ");
        printf("(estremi inclusi)\n");
        scanf("%d", &val);
    } while ((val < 1) || (val > 10));
    printf("Hai inserito %d\n", val);
    return 0;
}
```

Il ciclo `for`

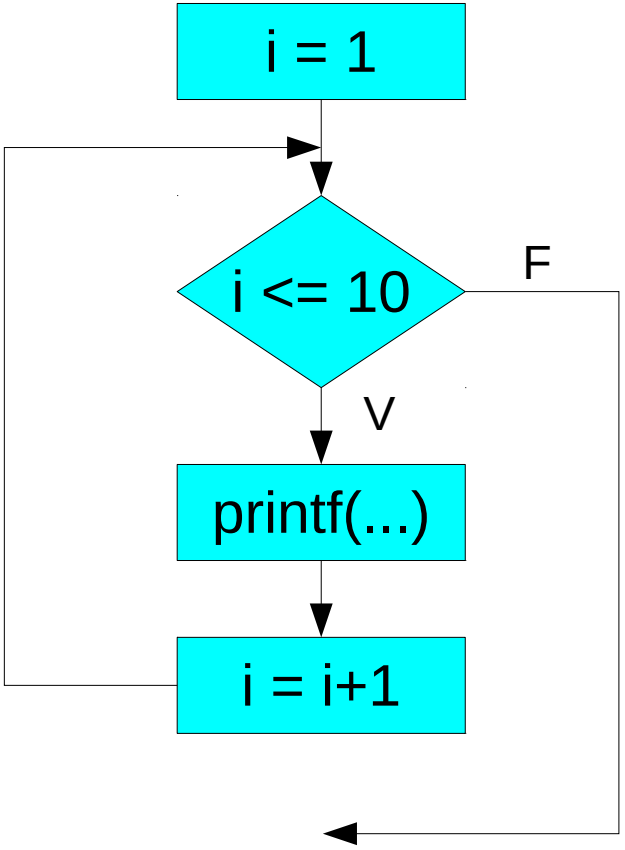
- **`for (Init; C; Inc) {Body}`**
  1. L'istruzione **`Init`** inizializza il valore di una variabile di controllo
  2. Se la condizione **`C`** è vera (diversa da zero), il corpo del ciclo (**`Body`**) viene eseguito
  3. Al termine dell'esecuzione di **`Body`** viene eseguita l'istruzione di incremento **`Inc`** e si torna al punto 2
  - Se **`Body`** è composto da una sola istruzione si possono omettere le parentesi graffe



Iterazione: `for`

- **`for`** `(`*`Init; C; Inc`*`)` `{`*`Body`*`}`

```
/* for.c : Esempio di ciclo for */
#include <stdio.h>

int main( void )
{
    int i;
    for (i=1; i <= 10; i=i+1) {
        printf("%d\n", i);
    }
    return 0;
}
```



Incrementi e decrementi…

- L'incremento può essere arbitrario…

```
for (i = 2; i <= 10; i = i + 2) {
  printf("%d\n", i);
}
```

- Può anche essere un decremento

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

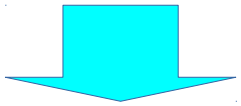
- Spesso si usano gli operatori **++** o **--**

```
for (i = 0; i < 10; i++) {
  printf("%d\n", i);
}
```

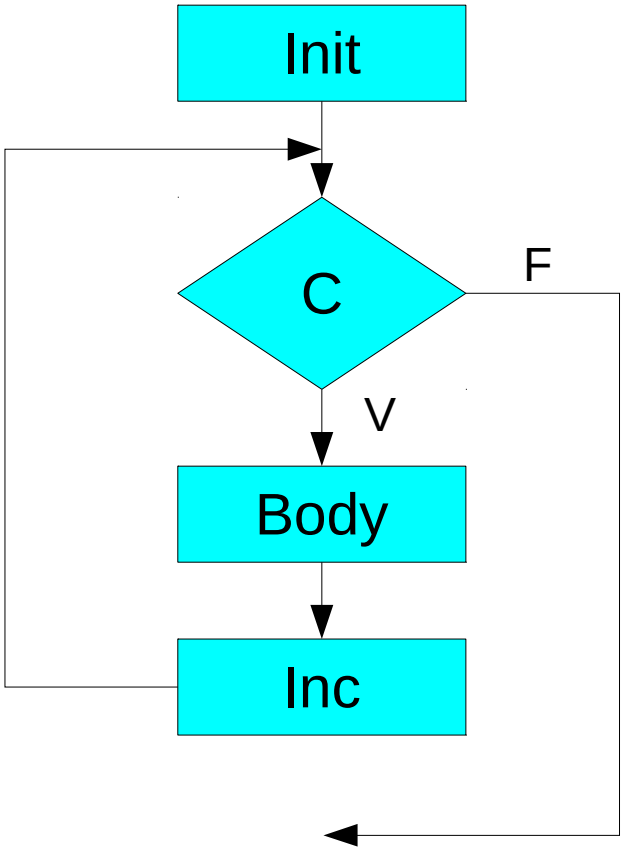
Cicli *for* e *while*

- Si può sempre simulare un **`for`** con un **`while`**

```
for ( Init ; C ; Inc ) {Body}
```



```
Init;
while ( C ) {
   Body;
   Inc;
}
```



Vantaggio del *for* rispetto al *while*
(nei casi in cui si possano usare entrambi)

- Gestione unitaria della variabile di controllo
  - Inizializzazione
  - Incremento
  - Test
- Con **`while`** o **`do/while`** devo invece cercare in giro per il codice

Come NON usare un ciclo “for”

...anche se tecnicamente corretto

```
i = 2;
for (; i <= 10; i = i + 2) {
  printf("%d\n", i);
}
```

Invece di scrivere così...

```
for (i = 2; i <= 10; ) {
  printf("%d\n", i);
  i = i + 2;
}
```

```
for (i = 2; i <= 10; i = i + 1) {
  printf("%d\n", i);
  i = i + 1; /* AARGHH!! */
}
```

Meglio scrivere così...

```
for (i = 2; i <= 10; i = i + 2) {
  printf("%d\n", i);
}
```

Ciclo infinito

- **`for (Init ; C ; Inc) {Body}`**
  - Uno o più tra *Init*, *C*, *Inc* e *Body* possono essere vuoti
  - Se *C* manca, è come se fosse sempre *True*

```
#include <stdio.h>

int main( void )
{
    for ( ; ; ) {
        printf("Hello, world!\n");
    }
    return 0;
}
```

Cicli annidati

- Cosa stampa questo frammento di codice?

```
int i, j;
for (i = 1; i <= 5; i++) {
  for (j = 1; j <= 10; j++) {
    printf("*");
  }
  printf("\n");
}
```

- Cosa stampa questo frammento di codice?

```
int i, j;
for (i = 1; i <= 5; i++) {
  for (j = 1; j <= i; j++) {
    printf("*");
  }
  printf("\n");
}
```

Attenzione: è una lettera 'i'

Esercizio

- Il triangolo di Floyd elenca i numeri interi positivi 1, 2, … su *n* righe
  - La prima riga contiene 1
  - La seconda riga contiene 2, 3
  - La terza riga contiene 4, 5, 6
  - ...
- Es (*n* = 5)

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

- Scrivere un programma che legge in input un valore intero *n*>0 e stampa il triangolo di Floyd con *n* righe
- Modificare il programma per stampare i primi *n* numeri interi, disponendoli secondo un triangolo di Floyd

Errori tipici

- … quello che sembra nel corpo del **`for`** non lo è…

```
int i;
for (i = 0; i <= 10; i = i + 1);
  printf("%d\n", i);
```

SBAGLIATO

- ... la condizione è invertita

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

SBAGLIATO

Altri errori tipici (1/2)

- Ciclo infinito per mancata modifica variabile

SBAGLIATO

```
int i = 0;
while (i <= 10) {
  printf("%d\n", i);
}
```

- Ciclo infinito per errata condizione

SBAGLIATO

```
int i = 1;
while (i != 10) {
  printf("%d\n", i);
  i = i + 2;
}
```

Altri errori tipici (2/2)

“Errore di uno”

- Stampare gli interi da 1 a 10 (estremi inclusi)

SBAGLIATO

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

SBAGLIATO

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

OK

```
int i = 0;
do {
  i++;
  printf("%d\n", i);
} while (i < 10);
```

OK

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

OK

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

OK

```
int i = 0;
while (i <= 9) {
  i++;
  printf("%d\n", i);
}
```

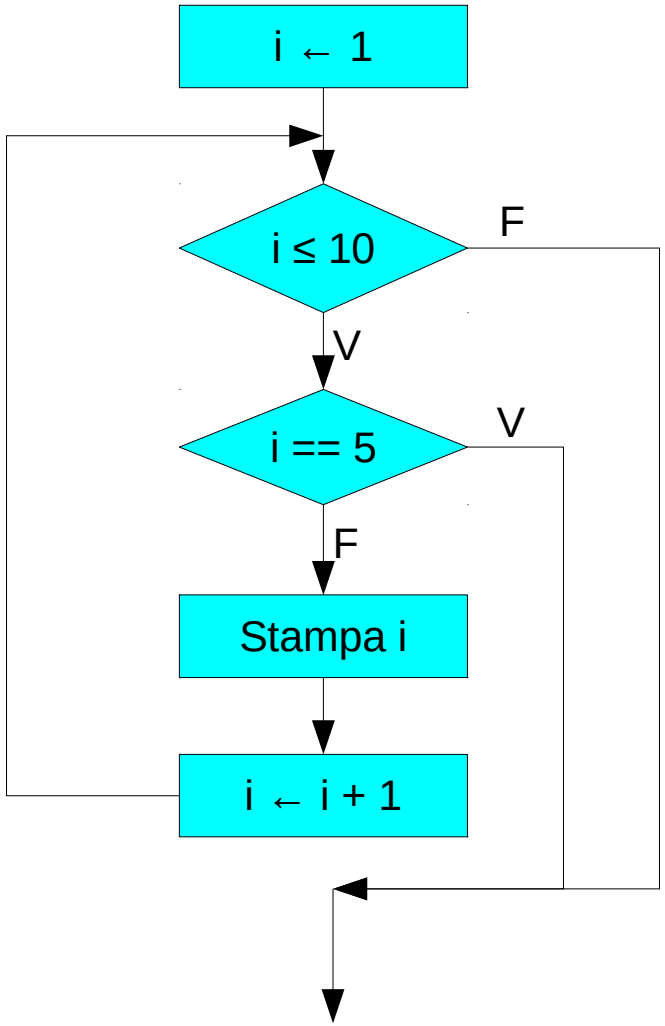
`break` e `continue`

- Interrompono il ciclo che li racchiude
- **`break`**: esce dal ciclo che contiene tale istruzione
  - dopo la "}"
- **`continue`**: passa all'iterazione successiva nel ciclo che contiene tale istruzione
  - prima della "}"

Esempio: **break**

```
int i;
for (i = 1; i <= 10; i++) {
  if (i == 5) {
    break;
  }
  printf("%d\n", i);
}
printf("Fuori dal ciclo\n");
```

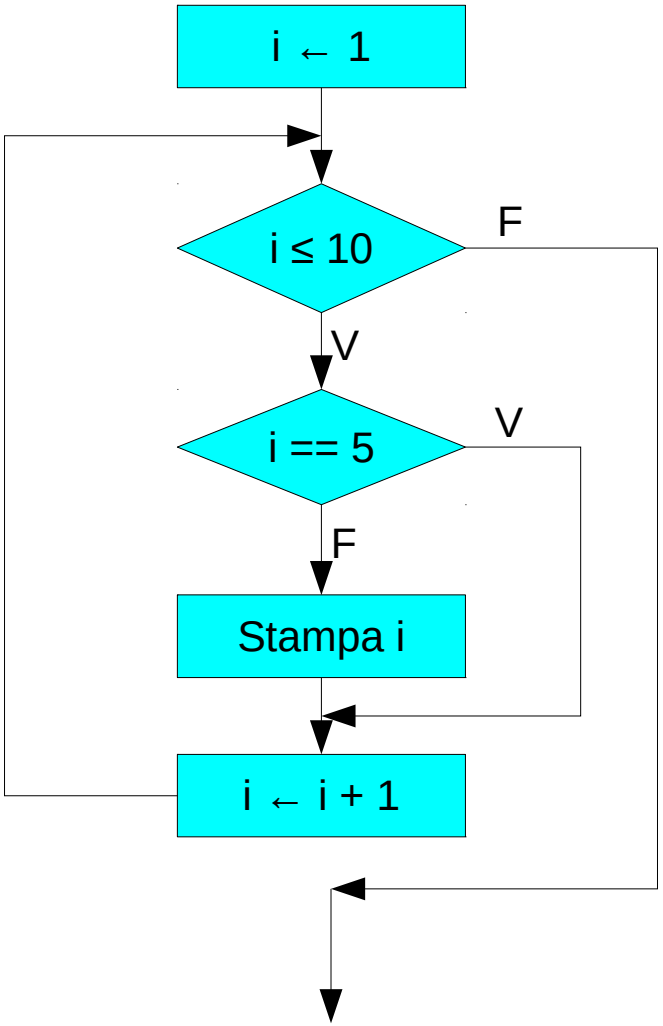
```
1
2
3
4
Fuori dal ciclo
```



Esempio: `continue`

```
int i;
for (i = 1; i <= 10; i++) {
  if (i == 5) {
    continue;
  }
  printf("%d\n", i);
}
printf("Fuori dal ciclo\n");
```

```
1
2
3
4
6
7
8
9
10
Fuori dal ciclo
```



Esercizio (per casa/lab)

- Scrivere un programma che stampa a video una scacchiera quadrata 8 × 8 come quella di fianco, alternando i caratteri '#' e '.' sulle righe
- La prima riga deve contenere i caratteri '#'
- Il programma deve essere facilmente modificabile per stampare una scacchiera $n \times n$
  - n non deve necessariamente essere pari

```
#######
.......
#######
.......
#######
.......
#######
.......
```

Esercizio (per casa/lab)

- Scrivere un programma che stampa a video una scacchiera quadrata 8×8 come quella di fianco, alternando i caratteri '#' e '.' sulle colonne
- La prima colonna a sinistra deve contenere i caratteri '#'
- Il programma deve essere facilmente modificabile per stampare una scacchiera $n \times n$
  - n non deve necessariamente essere pari

```
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
#.#.#.#.
```

Esercizio (per casa/lab)

- Scrivere un programma che stampa a video una scacchiera quadrata 8 × 8 come quella di fianco, alternando i caratteri '#' e '.' su righe e colonne
- Il primo carattere in alto a sinistra deve essere '#'
- Il programma deve essere facilmente modificabile per stampare una scacchiera $n \times n$
  - n non deve necessariamente essere pari

```
#.#.#.#.
.#.#.#.#
#.#.#.#.
.#.#.#.#
#.#.#.#.
.#.#.#.#
#.#.#.#.
.#.#.#.#
```