



UNIVERSITÀ DI PARMA

# Controllo di Flusso

*Begin at the beginning and go on till you  
come to the end: then stop.*

Lewis Carroll, Alice's Adventures in  
Wonderland.

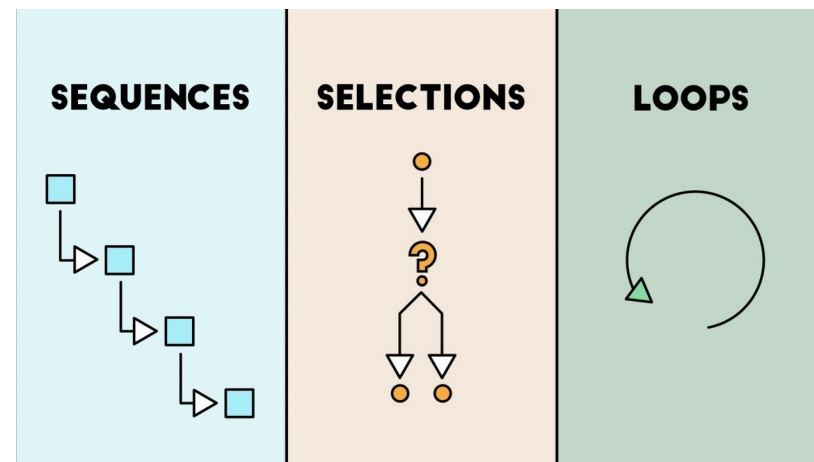
- Teorema Jacopini Böhm
- Sequenza
- Selezione
  - `if()`
  - `switch()-case`
- Iterazioni
  - `while()`
  - `do-while()`
  - `For(;;)`
- Indentazione

SUMMARY

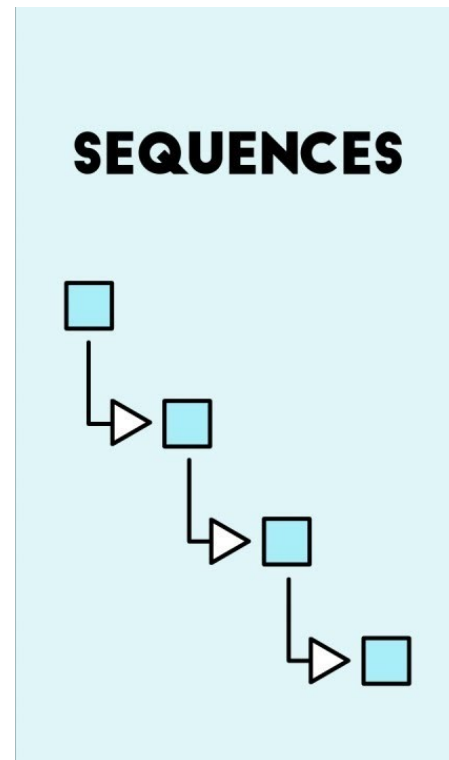


- Indica in che ordine vengono eseguite le direttive del C
  - Rememeber: Algoritmo = Passi + Ordine Esecuzione
- Fino ad ora ci siamo limitati alla “sequenza”
  - Non basta!
- Per alcune istruzioni vorrei poter impostare:
  - Ordine di esecuzione
  - Se eseguire o meno
  - Quante volte eventualmente ripetere

- Per costruire un programma sono necessari solo tre tipologie di esecuzione
  - Sequenza
  - Selezione (valutazione condizioni)
  - Iterazione (ciclo)
- Il C fornisce soluzioni multiple a queste strutture di controllo



- Quanto visto fino ad ora negli esempi
  - Le istruzioni sono eseguite in sequenza
- Prestare attenzione all'ordine
  - Chi scrive il codice decide l'ordine
  - Il sistema segue pedissequamente



- Definiamo il “blocco di istruzioni” (*Compound Statement*)
  - In pratica una sequenza di istruzioni poste tra “{” e “}”
  - Esempio → corpo della main()  

```
{  
    <dichiarazioni variabili>  
    <istruzione1>  
  
    ...  
    <istruzionen>  
}
```

- Detta anche “selezione”
- Valuto una o più condizioni
- In base all’esito della valutazione decido cosa fare



## SELECTIONS



```
if(espressione)  
    istruzione|blocco di istruzioni
```

- Se l'espressione è valutata come “vera” ( $\neq \emptyset$ )
  - L'istruzione o il blocco di istruzioni sottostanti vengono eseguiti
- Se l'istruzione è valutata come “falsa” ( $= \emptyset$ )
  - L'istruzione o il blocco di istruzioni sottostanti vengono saltate



```
if(espressione)
    istruzione|blocco di istruzioni
else
    istruzione|blocco di istruzioni
```

- Indico un'alternativa se condizione non si verifica

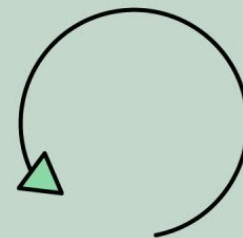
```
switch (espressione){  
    case <val1>: istruzione|blocco  
    case <val2>: istruzione|blocco  
    ...  
    default:      istruzione|blocco  
}
```

- Spesso occorre confrontare un'espressione con differenti valori
- Praticamente un if( == ) multiplo

- Nello switch()-case
  - L'espressione deve essere "intera"
  - No virgola mobile, no stringhe
- I vari "case" devono corrispondere a costanti
  - Tutte differenti
  - L'ordine è influente, "default" non deve obbligatoriamente essere l'ultimo
  - Non ci vogliono {} salvo che non definisca variabili
- In assenza del "break" l'esecuzione passa al "case" successivo
- Il "default" non è obbligatorio

- Ripeto istruzione o blocco di istruzioni
- In C tre tipologie
  - `while()`
  - `do-while()`
  - `for(;;)`

## LOOPS



```
while(espressione)  
    istruzione|blocco di istruzioni
```

- Fino a che l'espressione è vera l'istruzione/blocco viene eseguito
  - Se l'espressione è inizialmente falsa, nessuna esecuzione
- Problemi:
  - Rischio ciclo infinito

do

istruzione|blocco di istruzioni

while(espressione) ;

- Fino a che l'espressione è vera l'istruzione/blocco viene eseguito
  - Se l'espressione è inizialmente falsa, almeno una esecuzione
- Problemi:
  - Anche qui rischio ciclo infinito

- while() loop checks the condition first and then executes the statement
  - If the condition is false the loop is never executed
  - Entrance controlled loop
- do-while() loop executes the statement, then the condition is checked
  - Statement is executed at least once
  - Exit controlled loop
  - Brackets are mandatory!

- When using a loop, often:
  - We need to initialise something
  - We need to check a condition
  - We need to update something related to the condition



```
for(<inizializzazione>; <condizione>; <aggiornamento>)  
    istruzione|blocco di istruzioni
```

- Inizializzazione
  - espressione valutata inizialmente, tipicamente iniz. variabili
- Condizione
  - espressione valutata per stabilire se eseguire il corpo del ciclo
- Aggiornamento
  - espressione valutata dopo aver eseguito il corpo del ciclo

- Di fatto molto sovrapponibile al ciclo while()
  - Ma piú comodo
- Nessuna delle 3 espressioni è obbligatoria
- Ciclo infinito se tutte assenti
  - for(;;) equivalente a un while(1)
- In C99 è permessa la definizione variabili in for()

# while() vs for(;;)

- Perfettamente intercambiabili
- Tipicamente for() quando il numero di iterazioni è noto



```
<expr1>  
while(<expr2>)  
{  
    statement;  
    <expr3>;  
}
```

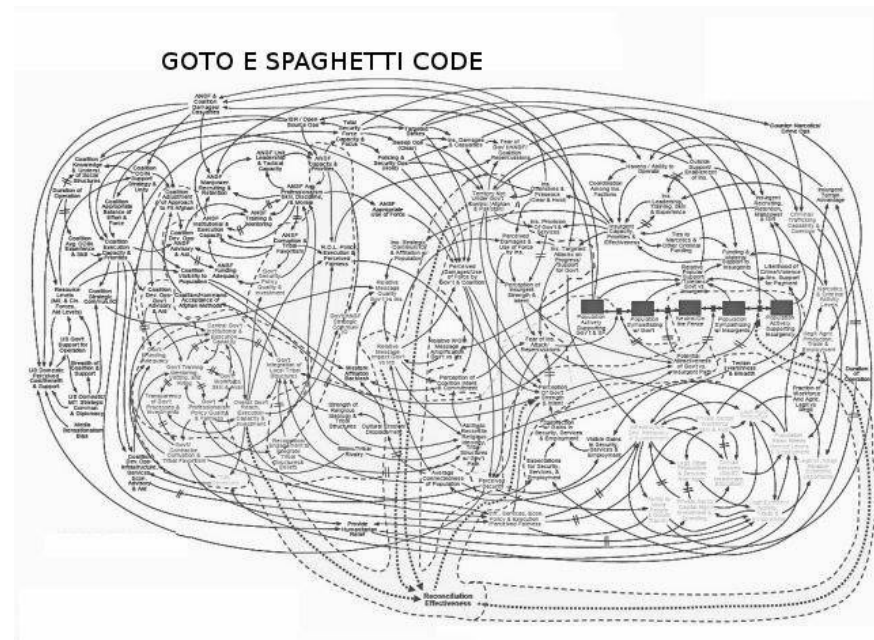


```
for(<expr1>;<expr2>;<expr3>)  
{  
    statement;  
}
```

- break:
  - Permette uscita da corpo ciclo o da case
  - Cicli nidificati → esco dal solo corpo del ciclo in cui viene usata
- continue:
  - Usata con cicli permette la ripetizione immediata del ciclo
  - Nei for(;;) → salto ad esecuzione aggiornamento
  - Nei while()/do-while() → salto a valutazione condizione

- Non voluti → il male
- Ma ci sono condizioni in cui servono
  - Semplificare il codice
- Con break esco
- Possibilità:
  - `for(;;)` // preferibile
  - `while(1)`

- goto + label
- Rendono i programmi complessi e difficilmente manutenibili o modificabili
- *Spaghetti Code*
- In parte vero anche per break e continue



- Indenting C programs can greatly help the readability
  - Clarity
- Who has to read your code?
  - Yes, that's me!
- As a general rule, a programmer has to dig out old code
  - And then to understand it!

```
#include <stdio.h>
int main()
{
    int v[10], n, i, j, temp;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", v+i);
    for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++)
            if (v[i] > v[j])
            {
                tmp = v[i];
                v[i] = v[j];
                v[j] = tmp;
            }
    for (i = 0; i < n; i++)
        printf("%d ", v[i]);
    printf("\n");
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    int v[10], n, i, j, temp;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", v+i);
    for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++)
            if (v[i] > v[j])
            {
                tmp = v[i];
                v[i] = v[j];
                v[j] = tmp;
            }
    for (i = 0; i < n; i++)
        printf("%d ", v[i]);
    printf("\n");
    return 0;
}
```



- We need to indent
  - Functions body (i.e. `main()`)
  - Loops statement
  - Selection statement
- How many spaces?
  - 2-4 are the most used options
  - >8 reduce readability
  - Avoid tabs

- Different styles
  - In examples we usually adopt the “Allman” or “BSD” style
  - Up to you, but being consistent is suggested

```
while (something)
{
    something();
    something_else();
}

final_thing();
```

- Cave Canem!
- There is someone that does not care about the indentation!
  - Yes, that's the compiler
  - You have always to use { }

```
int sum, sum2;  
for(int i=0; i<17; ++i)  
    sum += i;  
    sum2 += i*i;
```



UNIVERSITÀ DI PARMA

# Controllo di Flusso



**KEEP  
CALM  
IT'S  
QUESTION  
TIME**

*Begin at the beginning and go on till you  
come to the end: then stop.*

Lewis Carroll, Alice's Adventures in  
Wonderland.