

 POLITECNICO DI MILANO

Dipartimento di  
Elettronica e Informazione

# Seconda esercitazione

Riccardo Cattaneo

Dipartimento di Elettronica, Informazione e Biomedica  
Politecnico di Milano



POLITECNICO  
DI MILANO



# 😊 Materiale e feedback 😊

---

Materiale e feedback form su  
[www.riccardocattaneo.eu/teaching](http://www.riccardocattaneo.eu/teaching)

Il feedback form è importante per migliorare le esercitazioni e le lezioni durante il corso; ricordatevi di compilarlo!



# Una nuova istruzione di controllo: `switch`

---

- Prima di cominciare con l'esercitazione, vediamo una nuova istruzione di controllo
- Supponiamo di dover **scegliere** cosa fare **tra una o più opzioni**
  - tutte le opzioni sono note a ***compile time*** (ossia sono tutte note quando scriviamo il codice)
- istruzione `switch`



# switch

---

```
int a;  
// ... assegnamento di a ...  
switch (a) {  
    case 0:  
        { blocco di codice 0 }  
    case 1:  
        { blocco di codice 1 }  
    ...  
    default:  
        {eseguito solo se non viene eseguito alcun altro}  
}
```

variabile su  
cui switch  
predica

# switch

```
int a;  
// ... assegnamento di a ...  
switch (a) {  
    case 0:  
        { blocco di codice 0 }  
    case 1:  
        { blocco di codice 1 }  
    ...  
    default:  
        {eseguito solo se non viene eseguito alcun altro}  
}
```

# switch

variabile su  
cui switch  
predica

istruzione switch e  
segnalazione di a  
come variabile su cui  
predichiamo

```
int a;  
// ... assegnamento  
switch (a) {  
    case 0:  
        { blocco di codice 0 }  
    case 1:  
        { blocco di codice 1 }  
    ...  
    default:  
        {eseguito solo se non viene eseguito alcun altro}  
}
```

# switch

variabile su  
cui switch  
predica

istruzione switch e  
segnalazione di a  
come variabile su  
predichiamo

case indica che  
cominciano le  
istruzioni (*blocco*)  
associate ad una  
delle opzioni ( $a==0$ )

```
int a;  
// ... assegnamento  
switch (a) {  
    case 0: { blocco di codice 0 }  
    case 1: { blocco di codice 1 }  
    ...  
    default:  
        {eseguito solo se non viene eseguito alcun altro}  
}
```

# switch

variabile su  
cui switch  
predica

istruzione switch e  
segnalazione di a  
come variabile su  
predichiamo

case indica che  
cominciano le  
istruzioni (*blocco*)  
associate ad una  
delle opzioni ( $a==0$ )

default indica  
che cominciano le  
istruzioni (*blocco*)  
associate all'opzione,  
appunto, di default

```
int a;  
// ... assegnamento  
switch (a) {  
    case 0: { blocco di codice 0 }  
    case 1: { blocco di codice 1 }  
    ...  
    default: { eseguito solo se non viene eseguito }  
}
```



# switch: simulazione esecuzione

---

```
int a;      ↓ dichiarazione variabile
// ... assegnamento di a ...
switch (a) {
    case 0:
        { blocco di codice 0 }
    case 1:
        { blocco di codice 1 }
    ...
    default:
        {eseguito solo se non viene eseguito alcun altro}
}
```

# switch: simulazione esecuzione


```
int a;  
// ... assegnamento di a ...  
switch (a) {  
    case 0:  
        { blocco di codice 0 }  
    case 1:  
        { blocco di codice 1 }  
    ...  
    default:  
        {eseguito solo se non viene eseguito alcun altro}  
}
```



qui assegniamo (tipo con scanf, oppure "=") un valore ad a PRIMA DI USARE "a" ipotizziamo che a valga 0 dopo l'assegnamento.


# switch: simulazione esecuzione

---

```
int a;  
// ... assegnamento di a ...  
switch (a) {  
    case 0:  comincia lo switch, che legge il valore di a (ossia 0)  
        { blocco di codice 0 }  
    case 1:  
        { blocco di codice 1 }  
    ...  
    default:  
        {eseguito solo se non viene eseguito alcun altro}  
}
```

# switch: simulazione esecuzione

---

```
int a;  
// ... assegnamento di a ...  
switch (a) {  
    case 0:  a == 0 abbiamo un match, quindi ...  
        { blocco di codice 0 }  
    case 1:  
        { blocco di codice 1 }  
    ...  
    default:  
        {eseguito solo se non viene eseguito alcun altro}  
}
```

# switch: simulazione esecuzione

---

```
int a;  
// ... assegnamento di a ...  
switch (a) {  
    case 0:  
        { blocco di codice 0 }  
    case 1:  
        { blocco di codice 1 }  
    ...  
    default:  
        {eseguito solo se non viene eseguito alcun altro}  
}
```

↓ siccome  $a==0$ , eseguo il blocco di codice "blocco 0"

# switch: simulazione esecuzione

---

```
int a;  
// ... assegnamento di a ...  
switch (a) {  
    case 0:  
        { blocco di codice 0 }  
    case 1:  
        { blocco di codice 1 }  
    ...  
    default:  
        {eseguito solo se non viene eseguito alcun altro}  
}
```

DOMANDA: ora cosa succede?



# switch: simulazione esecuzione

```
int a;  
// ... assegnamento di a ...  
switch (a) {  
    case 0:  
        { blocco di codice 0 }  
    case 1:  
        { blocco di codice 1 }  
    ...  
    default:  
        {eseguito solo se non viene eseguito alcun altro}  
}
```



RISPOSTA: eseguo \*tutto\* quello che c'è sotto, ivi compreso default

NOTA che non testo nuovamente le condizioni "case"

# switch: simulazione esecuzione

---

```
int a;  
// ... assegnamento di a ...  
switch (a) {
```

```
    case 0:
```

```
        { blocco di codice 0 }
```

```
        break;
```

```
    case 1:
```

```
        { blocco di codice 1 }
```

```
        break;
```

```
    ...
```

```
    default:
```

```
        {eseguito solo se non viene eseguito alcun altro}
```

```
}
```



Se invece introduciamo un'istruzione  
break lo switch...



# switch: simulazione esecuzione

```
int a;  
// ... assegnamento di a ...  
switch (a) {  
    case 0:  
        { blocco di codice 0 }  
        break;  
    case 1:  
        { blocco di codice 1 }  
        break;  
    ...  
    default:  
        {eseguito solo se non viene eseguito alcun altro}  
}
```

Se invece introduciamo un'istruzione  
break lo switch...

... esce! e riprende ad  
eseguire il resto del codice  
ALL'ESTERNO dello switch

# *Separation of concerns*

---

## Nozione di *separation of concerns*

Concetto declinabile in: affronta i problemi *dividendoli* in sottoproblemi, risolvi separatamente e poi *combina* le soluzioni parziali per ottenere la soluzione del problema originale



# ***Modularità***

---

## Nozione di ***modularità***

La modularità permette al progettista (non solo informatico) di realizzare il concetto astratto di *separation of concerns*. Un modulo é un componente che ***risolve*** uno dei sottoproblemi, ***isolatamente*** da tutti gli altri.

(pensate ai “macroblocchi” di una macchina o di una turbina a gas per la produzione di energia elettrica: sono tutti istanze concrete del concetto più astratto di “modulo”)



# Es.1

---

- Si chieda all'utente di inserire un numero N con valori da 1 a 3.

- se N vale 1

- stampa “CASO 1”

- se N vale 2

- stampa “CASO 2”

- se N vale 3

- stampa “CASO 3”

[ avete mai chiamato un call center ;)? ]

[ ecco svelato come fanno la voce-guida ;) ]



## Es.2

---

- Si chieda all'utente di inserire un numero N con valori da 1 a 3.
  - se N vale 1
    - Si chieda all'utente di inserire un numero N con valori da 1 a 3.
      - caso “1” stampa “CASO 1.1”
      - caso “2” stampa “CASO 1.2”
      - caso “3” stampa “CASO 1.1”
  - se N vale 2
    - stampa “CASO 2”
  - se N vale 3
    - stampa “CASO 3”

# *flag*

---

## Nozione di *flag*

Concetto declinabile in: variabile che contiene il risultato di una condizione logica per quanto complessa che descrive uno stato del sistema. Comoda per ***riassumere uno stato*** descritto da variabili logiche in una variabile sola.

## Es.3

---

Chiedere all'utente di inserire un numero N. Se N é primo, stampalo, altrimenti richiedi all'utente di inserire un numero primo.

- definizione di numero primo: N primo iff N é divisibile SOLO per N ed 1
  - A divisibile B  $\iff A \bmod B == 0$ 
    - $A \bmod B \iff$  resto della divisone intera tra A e B
    - nel linguaggio C,  $A \bmod B$  si scrive  $A\%B$ 
      - `int resto = A%B;`

# Es.4

- Calcolare la radice quadrata di un numero  $N$  maggiore di 1 con un'accuratezza  $A$  *PROGRAMMABILE*.
  - es:  $N=10$ ;  $A=0,05$ 
    - $1^2=1$ ;  $2^2=4$ ;  $3^2=9$ ;  $4^2=16$ 
      - $9 \leq 10 \leq 16$
      - calcolo del massimo errore:  $|16-10|=6 > 0,05$
    - $3,1^2=9,61$ ;  $3,2^2=10,24$ 
      - $9,61 \leq 10 \leq 10,24$
      - calcolo del massimo errore:  $|10-9,61|=0,39 > 0,05$
    - $3,11^2=9,6721$ ;  $3,12^2=9,7344$ ;  $3,13^2=9,7969$ ;  $3,14^2=9,8596$ ;  $3,15^2=9,9225$ ;  $3,16^2=9,9856$ ;  $3,17^2=10,0489$ 
      - $9,9856 \leq 10 \leq 10,0489$
      - calcolo del massimo errore:  $|10-10,0489|=0,0489 \leq 0,05$
  - Soluzione: 3,165 (media tra 3,16 e 3,17)



# Es.5

---

- Si chieda all'utente di inserire un numero N con valori da 1 a 3.
  - se N vale 1
    - Si chieda all'utente di inserire un numero N con valori da 1 a 3.
      - caso “1” stampa “CASO 1.1”
      - caso “2” stampa “CASO 1.2”
      - caso “3” stampa “CASO 1.1”
  - se N vale 2
    - [ Es.3 ]
  - se N vale 3
    - stampa “CASO 3”

# Commento: differenza if e switch

---

```
int a, b;  
// inizializziamo a e b  
if (a == b)  
{  
    ...  
}
```

Come realizzo questa cosa con lo switch?



# Commento: differenza if e switch

Non si può! (almeno, direttamente!!! con una flag, magari... ☺)

- Lo **switch** può confrontare una variabile solo con valori *costanti* (ossia, come specificato nell'introduzione allo switch, noti a **compile time**). Lo switch a compile time non può sapere che valori assumerà una variabile durante l'esecuzione
  - (perché, ad esempio, è inizializzata con una scanf)
- L'**if**, invece, fa un confronto a **run time** quindi non è importante che il confronto avvenga tra una variabile ed una costante o fra più variabili; l'if può realizzare entrambi i confronti.

# 😊 Materiale e feedback 😊

---

Materiale e feedback form su  
[www.riccardocattaneo.eu/teaching](http://www.riccardocattaneo.eu/teaching)

Il feedback form è importante per migliorare le esercitazioni e le lezioni durante il corso; ricordatevi di compilarlo!

