

- ▶ Tutti i programmi che abbiamo visto fino ad ora prevedono istruzioni che vengono eseguite in **sequenza**. Ma come si fa ad esempio a leggere 2 interi e a stampare il valore massimo?
- ▶ Cioè come si fa ad eseguire un'istruzione solo se si verifica una determinata **condizione**?
- ▶ La risposta sono le **istruzioni condizionali** (esistono praticamente identiche in quasi tutti i linguaggi di programmazione).
- ▶ Per poter definire le istruzioni condizionali è necessario poter esprimere le **condizioni**.
- ▶ Le condizioni si esprimono attraverso le **espressioni booleane**.

Espressioni booleane

- ▶ In C non esiste un tipo Booleano \implies si usa il tipo **int** :

falso \iff 0

vero \iff 1 (in realtà qualsiasi valore diverso da 0)

Esempio: `2 > 3` ha valore 0 (ossia falso)

`5 > 3` ha valore 1 (ossia vero)

- ▶ **Operatori relazionali del C**

- ▶ `<`, `>`, `<=`, `>=` (minore, maggiore, minore o uguale, maggiore o uguale)

— priorità alta

- ▶ `==`, `!=` (uguale, diverso) — priorità bassa

Esempio: `temperatura <= 0` `velocita > velocita_max`

`voto == 30` `anno != 2000`

Operatori logici

- ▶ In ordine di priorità:
 - ▶ ! (negazione) — priorità alta
 - ▶ && (coniunzione)
 - ▶ || (disgiunzione) — priorità bassa

Semantica:

a	b	!a	a && b	a b
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

0 ... falso

1 ... vero (qualsiasi valore $\neq 0$)

Esempio:

$(a \geq 10) \ \&\& \ (a \leq 20)$ vero (1) se $10 \leq a \leq 20$

$(b \leq -5) \ || \ (b \geq 5)$ vero se $|b| \geq 5$

- ▶ Le espressioni booleane vengono valutate **da sinistra a destra**:
 - ▶ con `&&`, appena uno degli operandi è falso, restituisce falso **senza valutare il secondo operando**
 - ▶ con `||`, appena uno degli operandi è vero, restituisce vero **senza valutare il secondo operando**
- ▶ **Priorità** tra operatori di diverso tipo:
 - ▶ not logico — priorità alta
 - ▶ aritmetici
 - ▶ relazionali
 - ▶ booleani (and e or logico) — priorità bassa

Esempio:

`a+2 == 3*b || !trovato && c < a/3`
 è equivalente a
`((a+2) == (3*b)) || ((!trovato) && (c < (a/3)))`

Come va usato il codice dei caratteri

Esempio:

```
char x, y, z;  
x = 'a';  
y = 'd';
```

- ▶ Posso valutare $(x \leq y)$.
- ▶ Cosa mi dice? Se x precede y nell'ordine alfabetico.

Esempio:

```
char ch1 = 'a';  
char ch2 = 'c';  
char ch3 = (char) ((ch1 + ch2)/2);  
printf("%c", ch3);
```

- ▶ Cosa stampa? Stampa il carattere 'b'.

Come va usato il codice dei caratteri

- ▶ Convertiamo una lettera minuscola in maiuscolo:

Esempio:

```
char lower = 'k';  
char upper = (char) (lower - 'a' + 'A');  
printf("%c", upper);
```

- ▶ Convertiamo un carattere numerico (una cifra) nell'intero corrispondente:

Esempio:

```
char ch1 = '9';  
int num = ch1 - '0';
```

- ▶ Questi frammenti di programma sono **completamente portabili** (non dipendono dal codice usato per la rappresentazione dei caratteri).

Istruzione **if-else**

Sintassi:

```
if      (espressione)
        istruzione1
else    istruzione2
```

- ▶ `espressione` è un'**espressione booleana**
- ▶ `istruzione1` rappresenta il **ramo then** (deve essere un'unica istruzione)
- ▶ `istruzione2` rappresenta il **ramo else** (deve essere un'unica istruzione)

Semantica:

1. viene prima valutata `espressione`
2. se `espressione` è vera viene eseguita `istruzione1`
altrimenti (ovvero se `espressione` è falsa) viene eseguita `istruzione2`

```
int temperatura;  
  
printf("Quanti gradi ci sono? "); scanf("%d", &temperatura);  
if (temperatura >= 25)  
    printf("Fa caldo\n");  
else  
    printf("Si sta bene\n");  
  
printf("Arrivederci\n");
```

```
=> Quanti gradi ci sono? 30 ←  
Fa caldo  
Arrivederci  
=>
```

```
=> Quanti gradi ci sono? 18 ←  
Si sta bene  
Arrivederci  
=>
```


Istruzione `if`

- ▶ È un'istruzione **if-else** in cui manca la parte **else**.

Sintassi:

```
if (espressione)
    istruzione
```

Semantica:

1. viene prima valutata **espressione**
2. se **espressione** è vera viene eseguita **istruzione** altrimenti non si fa alcunché

Esempio:

```
int temperatura;
scanf("%d", &temperatura);
if (temperatura >= 25)
    printf("Fa caldo\n");
printf("Arrivederci\n");
```

==> 18 ←
Arrivederci

==> 30 ←
Fa caldo
Arrivederci

Blocco

- ▶ La sintassi di **if-else** consente di avere un'unica istruzione nel ramo **then** (o nel ramo **else**).
- ▶ Se in un ramo vogliamo eseguire più istruzioni dobbiamo usare un **blocco**.

Sintassi:

```
{  
    istruzione-1  
    ...  
    istruzione-n  
}
```

- ▶ Come già sappiamo e come rivedremo più avanti, un blocco può contenere anche **dichiarazioni**.

Esempio: Dati mese ed anno, calcolare mese ed anno del mese successivo.

```
int mese, anno, mesesucc, annosucc;
```

```
if (mese == 12)
{
    mesesucc = 1;
    annosucc = anno + 1;
}
else
{
    mesesucc = mese + 1;
    annosucc = anno;
}
```

If annidati (in cascata)

- ▶ Si hanno quando l'istruzione del ramo then o else è un'istruzione **if** o **if-else**.

Esempio: Data una temperatura, stampare un messaggio secondo la seguente tabella:

temperatura t	messaggio
$30 < t$	Molto caldo
$20 < t \leq 30$	Caldo
$10 < t \leq 20$	Gradevole
$0 < t \leq 10$	Freddo
$t \leq 0$	Molto freddo

```

if (temperatura > 30)
    printf("Molto caldo\n");
else if (temperatura > 20)
    printf("Caldo\n");
else if (temperatura > 10)
    printf("Gradevole\n");
else if (temperatura > 0)
    printf("Freddo\n");
else
    printf("Molto freddo\n");

```

Osservazioni:

- ▶ si tratta di un'unica istruzione **if-else**

```
if (temperatura > 30)
    printf("Molto caldo\n");
else ...
```
- ▶ non serve che la seconda condizione sia composta

```
if (temperatura > 30) printf("Molto caldo\n");
else /* il valore di temperatura e' <= 30 */
    if (temperatura > 20)
```

Non c'è bisogno di una congiunzione del tipo

```
(t <= 30) && (t > 20)
```

(analogamente per gli altri casi).

- ▶ **Attenzione:** il seguente codice

```
if (temperatura > 30) printf("Molto caldo\n");
if (temperatura > 20) printf("Caldo\n");
```

 ha ben altro significato (quale?)

Ambiguità dell'else

```
if (a >= 0) if (b >= 0) printf("b positivo");
else printf("???");
```

- ▶ `printf("???")` può essere la parte **else**
 - ▶ del primo **if** \Rightarrow `printf("a negativo");`
 - ▶ del secondo **if** \Rightarrow `printf("b negativo");`
- ▶ L'ambiguità sintattica si risolve considerando che un **else** fa sempre riferimento all'**if** più vicino, dunque

```
if (a > 0)
  if (b > 0)
    printf("b positivo");
  else
    printf("b negativo");
```

- ▶ Perché un **else** si riferisca ad un **if** precedente, bisogna inserire quest'ultimo in un blocco

```
if (a > 0)
  { if (b > 0) printf("b positivo"); }
else
  printf("a negativo");
```

Esercizio

Leggere un reale e stampare un messaggio secondo la seguente tabella:

gradi alcolici g	messaggio
$40 < g$	superalcolico
$20 < g \leq 40$	alcolico
$15 < g \leq 20$	vino liquoroso
$12 < g \leq 15$	vino forte
$10.5 < g \leq 12$	vino normale
$g \leq 10.5$	vino leggero

Esempio: Dati tre valori che rappresentano le lunghezze dei lati di un triangolo, stabilire se si tratti di un triangolo equilatero, isoscele o scaleno.

Algoritmo: determina tipo di triangolo
leggi i tre lati
confronta i lati a coppie, fin quando non
hai raccolto una quantità di informazioni
sufficiente a prendere la decisione
stampa il risultato


```
main()  {  
float primo, secondo, terzo;  
  
printf("Lunghezze lati triangolo ? ");  
scanf("%f%f%f", &primo, &secondo, &terzo);  
  
if (primo == secondo) {  
    if (secondo == terzo)  
        printf("Equilatero\n");  
    else  
        printf("Isoscele\n");  
}  
else {  
    if (secondo == terzo)  
        printf("Isoscele\n");  
    else if (primo == terzo)  
        printf("Isoscele\n");  
    else  
        printf("Scaleno\n");  
}
```

Esercizio

Risolvere il problema del triangolo utilizzando il seguente algoritmo:

```
Algoritmo: determina tipo di triangolo con conteggio  
leggi i tre lati  
confronta i lati a coppie contando  
  quante coppie sono uguali  
if le coppie uguali sono 0  
  è scaleno  
else if le coppie uguali sono 1  
  è isoscele  
  else è equilatero
```

Istruzione **switch**

- Può essere usata per realizzare una **selezione a più vie**.

Sintassi:

```
switch (espressione) {  
    case valore-1: istruzioni-1  
        break;  
    ...  
    case valore-n: istruzioni-n  
        break;  
    default : istruzioni-default  
}
```

Semantica:

1. viene valutata **espressione**
2. viene cercato il primo **i** per cui il valore di **espressione** è uguale a **valore-i**
3. se si trova tale **i**, allora vengono eseguite **istruzioni-i**
altrimenti vengono eseguite **istruzioni-default**

Esempio:

```
int giorno;
...
switch (giorno) {
    case 1: printf("Lunedì'\n");
            break;
    case 2: printf("Martedì'\n");
            break;
    case 3: printf("Mercoledì'\n");
            break;
    case 4: printf("Giovedì'\n");
            break;
    case 5: printf("Venerdì'\n");
            break;
    default : printf("Week end\n");
}
}
```

- ▶ Se abbiamo più valori a cui corrispondono le stesse istruzioni, possiamo raggrupparli come segue:

```
case valore-1: case valore-2:
                istruzioni
                break;
```

Esempio:

```
int giorno;
```

```
...
```

```
switch (giorno) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5: printf("Giorno lavorativo\n");
            break;

    case 6:
    case 7: printf("Week end\n");
            break;

    default : printf("Giorno non valido\n");
}
}
```

Osservazioni sull'istruzione **switch**

- ▶ L'**espressione** usata per la selezione può essere una qualsiasi espressione C che restituisce un valore **intero**.
- ▶ I valori specificati nei vari **case** devono invece essere **costanti** (o meglio valori noti a tempo di compilazione). In particolare, **non** possono essere espressioni in cui compaiono **variabili**.

Esempio: Il seguente frammento di codice è sbagliato:

```
int a;
switch (a) {
    case a<0: printf("negativo\n");
              /* ERRORE: a<0 non e' una costante*/
    case 0:   printf("nullo\n");
    case a>0: printf("positivo\n");
              /* ERRORE: a>0 non e' una costante*/
}
```

- ▶ In realtà il C non richiede che nei **case** di un'istruzione **switch** l'ultima istruzione sia **break**.

Quindi, in generale la **sintassi** di un'istruzione **switch** è:

```
switch (espressione) {  
    case valore-1:  istruzioni-1  
    ...  
    case valore-n:  istruzioni-n  
    default :      istruzioni-default  
}
```

Semantica:

1. viene prima valutata **espressione**
2. viene cercato il primo **i** per cui il valore di **espressione** è pari a **valore-i**
3. se si trova tale **i**, allora si eseguono in sequenza **istruzioni-i**, **istruzioni-(i+1)**, ..., fino a quando non si incontra **break** o è terminata l'istruzione **switch**, altrimenti vengono eseguite **istruzioni-default**

Esempio: più **case** di uno **switch** eseguiti in sequenza (corretto)

```
int lati;
printf("Immetti il massimo numero di lati del poligono (al piu' 6): ");
scanf("%d", &lati);
printf("Poligoni con al piu' %d lati: ", lati);

switch (lati) {
    case 6: printf("esagono, ");
    case 5: printf("pentagono, ");
    case 4: printf("rettangolo, ");
    case 3: printf("triangolo\n");
            break;
    case 2: case 1: printf("nessuno\n");
            break;
    default : printf("\nErrore: valore immesso > 6.\n");
}

```

- ▶ N.B. Quando si omettono i **break**, diventa rilevante l'**ordine** in cui vengono scritti i vari **case** . Questo può essere facile causa di errori.
È buona norma mettere **break** come ultima istruzione di ogni **case**

Esempio: più **case** di uno **switch** eseguiti in sequenza (scorretto)

```
int b;
printf("Immetti un numero tra 1 e 6: ");
scanf("%i", &b);

switch (b) {
    case 1: case 2: case 3: case 5: printf("Numero primo\n");
    case 4: case 6:                printf("Numero non primo\n");
    default :                       printf("Valore non valido!\n");
}
```

=> 3 ←

Numero primo
Numero non primo
Valore non valido!

=>

=> 4 ←

Numero non primo
Valore non valido!

=>