Istruzioni condizionali

- finora abbiamo visto come modificare il valore di variabili
- vediamo ora come **verificare il valore di variabili** e **compiere azioni diverse** a seconda del risultato della verifica

Istruzione `if-else`

Sintassi:

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

dove

- *`espressione`* è un'**espressione condizionale** il cui valore può essere vero o falso
- *`istruzione1`* rappresenta il **ramo then** (deve essere un'unica istruzione)
- *`istruzione1`* rappresenta il **ramo else** (deve essere un'unica istruzione)

Semantica: (ovvero qual'è il significato)

1. viene prima valutata *`espressione`*
2. se *`espressione`* è vera viene eseguita *`istruzione1`*
   altrimenti (ovvero se *`espressione`* è falsa) viene eseguita *`istruzione2`*
3. l'esecuzione procede con l'istruzione successiva all'istruzione `if-else`

Operatori relazionali del C

Tipicamente *`espressione`* coinvolge il **confronto** di due valori tramite gli **operatori relazionali**:

- `<`, `>`, `<=`, `>=` (minore, maggiore, minore o uguale, maggiore o uguale) — priorità alta
- `==`, `!=` (uguale, diverso) — priorità bassa

Esempio:

```
temperatura <= 0
velocita > velocita_max
voto == 30
anno != 2000
```

In C non esiste un tipo Booleano (unici valori: vero, falso) $\Longrightarrow$ si usa il tipo `int`:

falso $\Longleftrightarrow$ 0
vero $\Longleftrightarrow$ 1 (in realtà qualsiasi valore diverso da `0`)

Esempio:
`2 == 3` ha valore 0 (ossia falso)
`5 > 3` ha valore 1 (ossia vero)
`3 != 3` ha valore 0 (ossia falso)

Attenzione: non confondere "`=`" con "`==`"

Esempio di istruzione `if-else`:

```
int temperatura;

scanf("%d", &temperatura);
if (temperatura >= 25)
  printf("Fa caldo\n");
else
  printf("Si sta bene\n");

printf("Ciao\n");
```

```
# => 32^
Fa caldo
Ciao
#
```

Esercizio: Leggere due interi e stampare il maggiore dei due.

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* e si procede con l'istruzione successiva
3. altrimenti si procede direttamente con l'istruzione successiva

Esempio di istruzione `if`

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


Blocco di istruzioni

La sintassi di `if-else` ci permette soltanto di avere un'unica istruzione nel ramo then (o nel ramo else).

Se nel ramo then (o nel ramo else) vogliamo eseguire più istruzioni dobbiamo usare un **blocco di istruzioni**.

Sintassi:

```
{
   dichiarazioni-di-variabili
   istruzione-1
   ...
   istruzione-n
}
```

Le variabili dichiarate all'interno di un blocco vengono dette **locali** al blocco.

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

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

if (mese == 12) {
  mesesucc = 1;
  annosucc = succ + 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`.

If annidati con condizioni mutuamente escludentisi

Esempio: Leggere una temperatura (intero) e stampare un messaggio secondo la seguente tabella:

temperatura t	messaggio
$30 < t$	molto caldo
$20 < t \leq 30$	caldo
$10 < t \leq 20$	gradevole
$t \leq 10$	freddo

File: `ifelse/temperat.c`

Osservazioni:

- si tratta di un'unica istruzione `if-else`
- non serve che la seconda condizione sia composta `(t <= 30)` e `(t > 20)`
- `else` si riferisce all'`if` più vicino

Ambiguità dell'else

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

`printf("???")` può essere la parte else

- del primo `if` $\Longrightarrow$ `printf("a negativo");`
- del secondo `if` $\Longrightarrow$ `printf("b negativo");`

Ambiguità si risolve considerando che un `else` fa sempre riferimento all'`if` piu' vicino:

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

Perché un `else` si riferisca ad un `if` precedente, devo inserire l'ultimo `if` 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

Implementazione: file `ifelse/triang.c`

Esercizio: Si risolva 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
  then è scaleno
  else if le coppie uguali sono 1
    then è isoscele
    else è equilatero

Soluzione: file `ifelse/triang2.c`

Esercizio: Leggere i coefficienti a, b, c e calcolare gli zeri dell'equazione quadratica

$$a \cdot x^2 + b \cdot x + c = 0$$

A seconda del segno del discriminante $b^2 - 4 \cdot a \cdot c$ stampare le due soluzioni reali distinte, la soluzione reale doppia, o le due soluzioni complesse coniugate.

Soluzione: file `ifelse/equaquad.c`

Operatori logici (o booleani)

Permettono di combinare più condizioni ottenendo condizioni complesse.

In ordine di priorità:

- `!` (**not** logico) — priorità alta
- `&&` (**and** logico)
- `||` (**or** logico) — 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 >= 10) && (a <= 20)` risulta vero (pari a 1) se `a` è compresa tra 10 e 20

`(b <= -5) || (b >= 5)` risulta vero se il valore assoluto di `b` è ≥ 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
((a+2) == (3*b)) || ((!trovato) && (c < (a/3)))
```

è equivalente a

Esercizio: Determinare il tipo di un triangolo usando condizioni composte.

Soluzione: file `ifelse/triang3.c`

Istruzione `switch`

Può essere usate 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 prima valutata *espressione*
2. viene cercato il primo *i* per cui il valore di *espressione* è pari a *valore-i*
3. se si è trovato tale *i*, allora vengono eseguite *istruzioni-i*
   altrimenti vengono eseguite *istruzioni-default*
4. l'esecuzione procede con l'istruzione successiva all'istruzione `switch`

Se abbiamo più valori per cui eseguire le stesse istruzioni, si possono raggruppare omettendo `break`:

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

Esempio: calcolo dei giorni del mese
Implementazione: file `ifelse/giormese.c`

Esercizio: Calcolo della data del giorno successivo, tenendo conto anche degli anni bisestili.
Soluzione: file `ifelse/datasucc.c`

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 intere** (ovvero interi noti a tempo di compilazione). In particolare, non possono essere espressioni che fanno riferimento a variabili.

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*/
}
```

$\Longrightarrow$ L'utilità dell'istruzione `switch` è **limitata**.

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 è trovato tale *i*, allora vengono eseguite 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*
4. l'esecuzione procede con l'istruzione successiva all'istruzione `switch`

Esempio: più `case` di uno `switch` eseguiti in sequenza

```
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("\nImmetti un valore <= 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. $\Longrightarrow$

È buona norma mettere `break` come ultima istruzione di ogni `case`.