

# Programmazione 1 - Modulo C

## Java: Operatori logici lazy vs bitwise

**Marco Beccuti**

*Università degli Studi di Torino*

*Dipartimento di Informatica*

Ottobre 2020



# Operatori logici lazy

- Gli operatori **&&** e **||** sono operatori lazy (pigri) o short-cut:  
*valutano il secondo argomento solo se é necessario*

- Per esempio:

- ▶ **A && B**  $\Rightarrow$  B viene valutato solo se A vale true;
- ▶ **A || B**  $\Rightarrow$  B viene valutato solo se A vale false;

P	Q	P AND Q
1	1	1
1	0	0
0	1	0
0	0	0

P	Q	P OR Q
1	1	1
1	0	1
0	1	1
0	0	0

# Operatori logici lazy

- **$(a \neq 0) \ \&\& \ (b/a > 100)$**   $\Rightarrow$  non dà errore quando ad **a** è assegnato 0;
- **$(b/a > 100)$**   $\Rightarrow$  viene valutato solo quando **a** è diverso da zero.

# Operatori logici bitwise

- Gli operatori **&** e **|** sono operatori bitwise:

*valutano sempre i due argomenti*

- Per esempio:

- ▶ **A & B**  $\Rightarrow$  A e B sono sempre valutati;
- ▶ **A | B**  $\Rightarrow$  A e B sono sempre valutati;

# Operatori logici bitwise

- $(a \neq 0) \ \& \ (b/a > 100)$  da errore quando ad **a** é assegnato 0;
- $(b/a > 100)$  viene valutato sempre!!!.

# Operatori logici bitwise

- L'operatore AND bitwise `&` se applicato a due variabili intere effettua AND dei bit:

```
int a=60 // = 0011 1100
```

```
int b=13 // = 0000 1101
```

```
c=a&b // = 0000 1100 ossia c=12
```

a	0	0	1	1	1	1	0	(60)
b	0	0	0	0	1	1	0	(13)
<hr/>								
a&b	0	0	0	0	1	1	0	(12)

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

Assumiamo per semplicità che gli interi siano codificati con 8bit

# Operatori logici bitwise

- L'operatore OR bitwise | se applicato a due variabili intere effettua OR dei bit:

```
int a=60 // = 0011 1100
```

```
int b=13 // = 0000 1101
```

```
c=a|b // = 0011 1101 ossia c=61
```

a	0011	1100	(60)
b	0000	1101	(13)
<hr/>			
a b	0011	1101	(61)

a	b	a b
0	0	0
1	0	1
0	1	1
1	1	1

Assumiamo per semplicità che gli interi siano codificati con 8bit

# Operatori logici bitwise

- L'operatore NOT bitwise  $\sim$  esegue la negazione dei bit:

```
int a=60 // = 0011 1100
```

```
c=~a // = 1100 0011 ossia c=-61
```

a	0 0 1 1	1 1 0 0	(60)
<hr/>			
$\sim a$	1 1 0 0	0 0 1 1	(61)

a	$\sim a$
0	1
1	0

Assumiamo per semplicità che gli interi siano codificati con 8bit

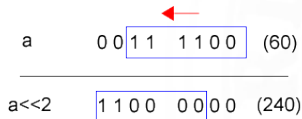


# Operatori logici bitwise

- L'operatore shift `<<` scorre n bit a sinistra.  
Gli spazi vuoti a destra sono riempiti con degli zeri, mentre quelli che escono a sinistra sono eliminati.

```
int a=60 // = 0011 1100
```

```
c=a<<2 // = 1111 0000 ossia c=240
```



a<<n

n=2


Assumiamo per semplicità che gli interi siano codificati con 8bit

# Operatori logici bitwise

- L'operatore shift `>>` scorre `n` bit a destra.  
Gli spazi vuoti a sinistra sono riempiti con degli zeri mentre quelli che escono a destra sono eliminati.

```
int a=60 // = 0011 1100
```

```
c=a>>2 // = 0000 1111 ossia c=15
```

a            (60)

---

a>>2      00 00 11 11      (15)

a>>n

n=2

Assumiamo per semplicità che gli interi siano codificati con 8bit