

# Tecniche della Programmazione, lez. 5

Tipi base, loro uso, loro rappresentazione (quasi tutta)

- TIPO
  - Valori
  - Operatori
- Espressioni
  - Valutazione

# Tipo (VALORI + operazioni)

(Per i tipi elementari)

Un tipo denota un  
INSIEME DI VALORI e un  
INSIEME DI OPERAZIONI  
ammissibili su quei valori

**INSIEME DI VALORI:** si chiama anche **DOMINIO** del tipo.

**int**  $[-2^{31}, 2^{31} - 1]$  sottoinsieme degli INTERI;  
rappresentazione binaria (*Complemento a 2*) in 32 bit

**float** sottoinsieme dei numeri reali;  
rappresentazione *Floating Point* in 32 bit

**double** come sopra ma 64 bit

**char**  $[0, 255]$  Caratteri  $[-128, 127]$  !!!  
rappresentazione come codice della tabella ASCII

NB Dimensioni in bit (dipendono dalla configurazione del sistema di programmazione)

# BTW ... come scriviamo i numeri, noi ...

numero = concetto

**numera**le = il modo di scrivere il numero, con le cifre disponibili

cifre disponibili 0 1 2 3 4 5 6 7 8 9

NUMERALE

7

0

10

127

???

1919

0021

0999

99

NUMERO

sette

zero

dieci

centoventisette

millenovecentodiciannove

millenovecentodiciannove

ventuno

novecento99

novantanove

tsk ... non bastano 3 cifre  
ne servono 4, minimo 4

e possiamo scrivere il numerale del  
medesimo numero anche usando piu`  
cifre del minimo: le cifre in piu`  
sono zeri, cosi` non "pesano" sul  
risultato finale

# BTW ... come scriviamo i numeri, noi ...

numero = concetto

numerale = il modo di scrivere il numero, con le cifre disponibili

cifre disponibili 0 1 2 3 4 5 6 7 8 9

Assumiamo che i numerali debbano essere scritti con 4 cifre:

NUMERALE

0127

0007

0000

0010

1919

0021

0999

0061

9999

NUMERO

centoventisette

sette

zero

dieci

millenovecentodiciannove

ventuno

novacentonovantanove

sessantuno

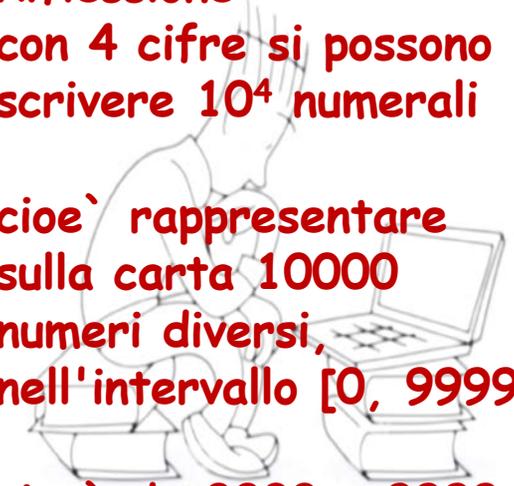
novemilanovecento99

**Riflessione:**

con 4 cifre si possono  
scrivere  $10^4$  numerali

cioè rappresentare  
sulla carta 10000  
numeri diversi,  
nell'intervallo  $[0, 9999]$

cioè da 0000 a 9999



# BTW ... come scriviamo i numeri, noi ... e lui



numero = concetto

**numerales** = il modo di scrivere il numero, con le cifre disponibili

assumiamo invece che le cifre disponibili siano **0 1**

NUMERALE

NUMERO

100

quattro

111

sette

0

zero

1

uno

10

due

???

otto

← tsk ... non bastano 3 cifre, ne servono 4, minimo 4

1000

otto

e possiamo scrivere il numerales del medesimo numero anche usando piu` cifre del minimo ...

0001

uno

????

sedici

← tsk ... non bastano 4 cifre, ne servono 5, minimo 5

# BTW ... come scriviamo i numeri, noi ... e lui



numero = concetto

numerale = il modo di scrivere il numero, con le cifre disponibili  
cifre disponibili 0 1

numero di cifre usate per scrivere i numerali: 8

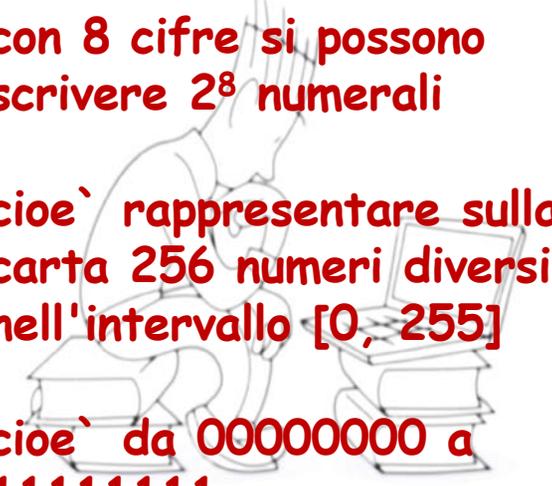
NUMERALE	NUMERO
00000100	quattro
00000111	sette
00000000	zero
00000001	uno
00000010	due
00001000	otto
10000000	128 (sarebbe centoventotto)
10000001	centoventinove
11111110	254 (sarebbe ...)
11111111	255 (...)

## Riflessione:

con 8 cifre si possono scrivere  $2^8$  numerali

cioè rappresentare sulla carta 256 numeri diversi, nell'intervallo  $[0, 255]$

cioè da 00000000 a 11111111



# BTW ... esercizio per casa

numero di cifre usate per scrivere i numerali: 4  
scrivere i numerali binari e i corrispondenti numeri

NUMERALE	NUMERO
0000	...
0001	...
0010	...
0011	...
0100	...
...	...

## quanti possibili numerali con 4 bit?

per ogni cifra del numerale (cioè la singola posizione nel numerale) possiamo usare una tra le due cifre binarie possibili (cifra 0 o cifra 1)

**QUINDI**

$$2*2*2*2$$

2 alla quarta

$$2^4$$

**16**

Wait!

Bit?!

Cioè ogni bit rappresenta una cifra binaria e quindi dire che scriviamo un numero su 4 bit equivale a dire che lo scriviamo con quattro cifre binarie?

Sì.

# Tipo (**VALORI** + operazioni)

(Per i tipi elementari)

Un tipo denota un  
INSIEME DI VALORI e un  
INSIEME DI OPERAZIONI  
ammissibili su quei valori

**INSIEME DI VALORI:** si chiama anche **DOMINIO** del tipo.

**int**  $[-2^{31}, 2^{31} - 1]$  sottoinsieme degli INTERI;  
rappresentazione binaria (*Complemento a 2*) in 32  
bit

**float** sottoinsieme dei numeri reali;  
rappresentazione *Floating Point* in 32 bit

**double** come sopra ma 64 bit

**char**  $[0, 255]$  Caratteri  $[-128, 127]$  !!!  
rappresentazione come codice della tabella ASCII

**NB** Dimensioni in bit (dipendono dal sistema di programmazione)

# Tipo (valori + OPERAZIONI)

TIPO = VALORI + OPERAZIONI ammissibili su quei valori

Per i tipi elementari

## INSIEME DI OPERAZIONI (operatori)

**int** +, -, \*, /, % (producono un valore int)

**float** +, -, \*, / (producono un valore float)

**double** come sopra ma producono un valore double

**char** come sopra per int !!!  
rappresentazione come codice della tabella ASCII

Producono??

# Espressione



(mainly) Combinazione di operatori ed operandi, che, mediante VALUTAZIONE, dà luogo ad un valore

Esempio (usiamo gli interi)

```
int n, m, q, p;  
  
q = (n+m)*p  
printf ("bla bla %d\n", n+m * p)
```

Quali sarebbero le espressioni?



# Espressione

Esempio (usiamo gli interi)

(mainly) Combinazione di operatori ed operandi, che, mediante VALUTAZIONE, dà luogo ad un valore



```
int n=3, m=4, q, p=6;
```

```
q = (n+m)*p
```

```
printf ("bla bla %d\n", n+m * p)
```

Espressione

# Espressione

Esempio (usiamo gli interi)

(mainly) Combinazione di operatori ed operandi, che, mediante VALUTAZIONE, dà luogo ad un valore



```
int n=3, m=4, q, p=6;
```

```
q = (n+m)*p
```

```
printf("bla bla %d\n", n+m * p)
```

Anche questa è un'espressione

Ma ne parliamo tra 17 slide circa ...

Valutiamo  $(n+m)*p$



# VALUTAZIONE di un'espressione ( $(n+m)*p$ )

Esempio (usiamo gli interi)

```
int n=3, m=4, q, p=6;  
  
q = (n+m)*p  
printf ("bla bla %d\n", n+m * p)
```

Espressioni anche queste ...  
(sottoespressioni di una espressione,  
ma sempre espressioni:

la VALUTAZIONE  
dell'espressione "contenente"  
e` fatta attraverso la  
valutazione delle  
sottoespressioni "contenute"  
(o "sottoespressioni")

Espressione elementare ... accesso  
alla variabile

Quanto vale  $(n+m)*p$  ?





# VALUTAZIONE di un'espressione

Esempio (usiamo gli interi)

```
int n=3, m=4, q, p=6;
```

```
q = (n+m)*p  
printf ("bla bla %d\n", n+m * p)
```

Espressioni (sottoespressioni di una espressione, ma sempre espressioni: la VALUTAZIONE dell'espressione contenente e' fatta attraverso la valutazione delle sottoespressioni)

Quanto vale  $(n+m)*p$  ?

Quanto vale  $(n+m)$  ?

Quanto vale  $p$  ?

Espressione elementare ... accesso alla variabile



# VALUTAZIONE di un'espressione

Esempio (usiamo gli interi)

```
int n=3, m=4, q, p=6;
```

```
q = (n+m)*p  
printf ("bla bla %d\n", n+m * p)
```

Espressioni (sottoespressioni di una espressione, ma sempre espressioni: la VALUTAZIONE dell'espressione contenente e' fatta attraverso la valutazione delle sottoespressioni)

Quanto vale  $(n+m)*p$  ?

Quanto vale  $(n+m)$  ?

Quanto vale  $n$  ?

Quanto vale  $m$  ?

Quanto vale  $p$  ?

Espressione elementare ... accesso alla variabile

Espressione: un solo operatore e due sottoespressioni elementari

$(n+m) * p$  e' un'espressione *COMPOSTA* da sottoespressioni



# VALUTAZIONE di un'espressione

Esempio (usiamo gli interi)

```
int n=3, m=4, q, p=6;  
  
q = (n+m)*p  
printf ("bla bla %d\n", n+m * p)
```

Espressioni (sottoespressioni di una espressione, ma sempre espressioni: la VALUTAZIONE dell'espressione contenente e' fatta attraverso la valutazione delle sottoespressioni)

Quanto vale  $(n+m)*p$ ? ???

Quanto vale  $(n+m)$ ? ?

Quanto vale  $n$ ? 3

Quanto vale  $m$ ? 4

Quanto vale  $p$ ? 6

$(n+m) * p$  e' un'espressione *COMPOSTA* da sottoespressioni



# VALUTAZIONE di un'espressione

Esempio (usiamo gli interi)

```
int n=3, m=4, q, p=6;  
  
q = (n+m)*p  
printf ("bla bla %d\n", n+m * p)
```

Espressioni (sottoespressioni di una espressione, ma sempre espressioni: la VALUTAZIONE dell'espressione contenente e' fatta attraverso la valutazione delle sottoespressioni)

Quanto vale  $(n+m)*p$ ? ???

Quanto vale  $(n+m)$ ? 7

Quanto vale  $n$ ? 3

Quanto vale  $m$ ? 4

Quanto vale  $p$ ? 6

$(n+m) * p$  e' un'espressione *COMPOSTA* da sottoespressioni



# VALUTAZIONE di un'espressione

## Esempio (usiamo gli interi)

```
int n=3, m=4, q, p=6;

q = (n+m)*p
printf ("bla bla %d\n", n+m * p)
```

Espressioni (sottoespressioni di una espressione, ma sempre espressioni: la VALUTAZIONE dell'espressione contenente e' fatta attraverso la valutazione delle sottoespressioni)

- Quanto vale  $(n+m)*p$ ?  $7*6$
- Quanto vale  $(n+m)$ ?  $7$
- Quanto vale  $n$ ?  $3$
- Quanto vale  $m$ ?  $4$
- Quanto vale  $p$ ?  $6$

Espressione elementare ... accesso alla variabile

Espressione: un solo operatore e due sottoespressioni elementari

$(n+m) * p$  e' un'espressione **COMPOSTA** da sottoespressioni

# VALUTAZIONE di un'espressione (TIPO del valore)



Il risultato della valutazione e' un valore di un certo tipo; dal tipo degli operandi dipende l'uso di un operatore o di un altro

```
int n, m, q, p;  
double a, b;  
... lettura di n(3), m(4), p(6), a(3.0), b(4.0)  
  
    q = n/m;  
printf ("bla bla %d\n", m/n);  
printf ("bla bla %g\n", a/b);  
printf ("bla bla %g\n", b/a);
```





# VALUTAZIONE di un'espressione (TIPO del valore)

Il risultato della valutazione e` un valore di un certo tipo; dal tipo degli operandi dipende l'uso di un operatore o di un altro

```
int n, m, q, p;  
double a, b;  
... lettura di n(3), m(4), p(6), a(3.0), b(4.0)  
  
q = n/m;  
printf ("bla bla %d\n", m/n);  
printf ("bla bla %g\n", a/b);  
printf ("bla bla %g\n", b/a);
```

0

L'op. / tra interi produce un valore intero;  
quello tra double (e` un altro '/') produce un valore double ...

1/2 e` ?	(divisione tra interi)
3/4 e` ?	(divisione tra interi)
1.0/2.0 e` ?	(divisione tra float/double)
3.0/4.0 e` ?	(divisione tra float/double)



# VALUTAZIONE di un'espressione (TIPO del valore)

Il risultato della valutazione e` un valore di un certo tipo; dal tipo degli operandi dipende l'uso di un operatore o di un altro

```
int n, m, q, p;  
double a, b;  
... lettura di n(3), m(4),  
p(6), a(3.0), b(4.0)  
    q = n/m;  
    printf ("bla bla %d\n", m/n);  
    printf ("bla bla %g\n", a/b);  
    printf ("bla bla %g\n", b/a);
```

Diagram illustrating the evaluation of expressions and their resulting types:

- `q = n/m;` results in `0` (integer).
- `printf ("bla bla %d\n", m/n);` results in a smiley face (integer).
- `printf ("bla bla %g\n", a/b);` results in two smiley faces (double).
- `printf ("bla bla %g\n", b/a);` results in two smiley faces (double).

L'op. / tra interi produce un valore intero; quello tra double (e` un altro /) un valore double ...

<code>1/2 e` 0</code>	<code>(divisione tra interi)</code>	<b>risultato "troncato"</b>
<code>3/4 e` 0</code>	<code>(divisione tra interi)</code>	
<code>1.0/2.0 e` ?</code>	<code>(divisione tra float/double)</code>	
<code>3.0/4.0 e` ?</code>	<code>(divisione tra float/double)</code>	



# VALUTAZIONE di un'espressione (TIPO del valore)

Il risultato della valutazione e` un valore di un certo tipo; dal tipo degli operandi dipende l'uso di un operatore o di un altro

```

int n, m, q, p;
double a, b;
... lettura di n(3), m(4),
p(6), a(3.0), b(4.0)
  q = n/m;
printf ("bla bla %d\n", m/n);
printf ("bla bla %g\n", a/b);
printf ("bla bla %g\n", b/a);

```

L'op. / tra interi produce un valore intero; quello tra double (e` un altro /) un valore double ...

1/2 e` 0	(divisione tra interi)	risultato "troncato"
3/4 e` 0	(divisione tra interi)	idem
1.0/2.0 e` 0.5	(divisione tra float/double)	
3.0/4.0 e` 0.75	(divisione tra float/double)	



# VALUTAZIONE di un'espressione (TIPO del valore)

Il risultato della valutazione e' un valore di un certo tipo; dal tipo degli operandi dipende l'uso di un operatore o di un altro

```
int n, m, q, p;  
double a, b;  
... lettura di n(3), m(4),  
p(6), a(3.0), b(4.0)  
    q = n/m;  
printf ("bla bla %d\n", m/n);  
printf ("bla bla %g\n", a/b);  
printf ("bla bla %g\n", b/a);
```

Diagram illustrating the evaluation of expressions and their resulting types:

- `m/n` (integer division) results in `0`.
- `a/b` (double division) results in `0.75`.
- `b/a` (double division) results in a smiley face `😊`.

L'op. / tra interi produce un valore intero; quello tra double (e' un altro /) un valore double ...

<code>1/2 e' 0</code>	(divisione tra interi)	<b>risultato</b> float/double
<code>3/4 e' 0</code>	(divisione tra interi)	
<code>1.0/2.0 e' 0.5</code>	(divisione tra float/double)	
<code>3.0/4.0 e' 0.75</code>	(divisione tra float/double)	



# VALUTAZIONE di un'espressione (TIPO del valore)

Il risultato della valutazione e` un valore di un certo tipo; dal tipo degli operandi dipende l'uso di un operatore o di un altro

```
int n, m, q, p;  
double a, b;  
... lettura di n(3), m(4),  
p(6), a(3.0), b(4.0)  
    q = n/m;  
printf ("bla bla %d\n", m/n);  
printf ("bla bla %g\n", a/b);  
printf ("bla bla %g\n", b/a);
```

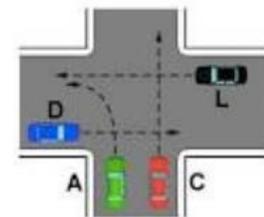
Diagram illustrating the evaluation of expressions and their resulting types:

- `m/n` (integer division) results in `0`.
- `a/b` (double division) results in `0.75`.
- `b/a` (double division) results in `1.33`.

L'op. / tra interi produce un valore intero; quello tra double (e` un altro /) un valore double ...

<code>1/2 e` 0</code>	(divisione tra interi)	<b>risultato</b> float/double
<code>3/4 e` 0</code>	(divisione tra interi)	
<code>1.0/2.0 e` 0.5</code>	(divisione tra float/double)	
<code>3.0/4.0 e` 0.75</code>	(divisione tra float/double)	

# Valutazione di Espressione - PRECEDENZA



Esempio (usiamo gli interi)

```
int n=3, m=4, q, p=6;

q = (n+m)*p
printf ("bla bla %d\n", n+m * p)
```

Quanto vale  $(n+m)*p$  ?  $7*6$

MA

Che viene stampato dalla `printf`??

Cioe' ... Quanto vale  $n+m*p$  ?



# Valutazione di Espressione - PRECEDENZA



```
int n=3, m=4, q, p=6;
```

```
q = (n+m)*p
```

```
printf ("bla bla %d\n", n+m * p)
```

Quanto vale  $(n+m)*p$  ?  $7*6$

Che viene stampato? Cioe` ... Quanto vale  $n+m*p$  ?  $3+24$

L'operatore  $*$  ha precedenza su  $+$ , quindi viene valutata prima  $m*p$  e poi la somma tra il valore ottenuto ed  $n$

# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
... un po' meno	*	/	%
... ancora meno	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo  
... niente '[' ne' '{' ok?

Quanto vale  $(n+m)*p$  ?  $7*6$

Che viene stampato? Cioe' ... Quanto vale  $n+m*p$  ?  $3+24$

# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
... un po' meno	*	/	%
... ancora meno	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo

... niente '[' ne' '{' ok?

Quanto vale  $(n+m)*p$  ?  $7*6$

( $n=3$ ,  $m=4$ ,  $q$ ,  $p=6$ ;) )

Quanto vale  $n+m-p$  ?

Quanto vale  $n*m+p$  ?

Quanto vale  $n*m/p$  ?

Quanto vale  $n*m/(p+1)$  ?

Quanto vale  $n*m/p+1$  ?



# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
... un po' meno	*	/	%
... ancora meno	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo

... niente '[' ne' '{' ok?

Quanto vale  $(n+m)*p$  ?  $7*6$

( $n=3$ ,  $m=4$ ,  $q$ ,  $p=6$ ;) )

Quanto vale  $n+m-p$  ?  $7-6$

Quanto vale  $n*m+p$  ?

Quanto vale  $n*m/p$  ?

Quanto vale  $n*m/(p+1)$  ?

Quanto vale  $n*m/p+1$  ?

# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
... un po' meno	*	/	%
... ancora meno	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo

... niente '[' ne' '{' ok?

Quanto vale  $(n+m)*p$  ?  $7*6$

( $n=3$ ,  $m=4$ ,  $q$ ,  $p=6$ ;) )

Quanto vale  $n+m-p$  ?  $7-6$

Quanto vale  $n*m+p$  ?  $12+6$

Quanto vale  $n*m/p$  ?

Quanto vale  $n*m/(p+1)$  ?

Quanto vale  $n*m/p+1$  ?

# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
... un po' meno	*	/	%
... ancora meno	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo

... niente '[' ne' '{' ok?

Quanto vale  $(n+m)*p$  ?  $7*6$

( $n=3$ ,  $m=4$ ,  $q$ ,  $p=6$ ;) )

Quanto vale  $n+m-p$  ?  $7-6$

Quanto vale  $n*m+p$  ?  $12+6$

Quanto vale  $n*m/p$  ?  $12/6$

Quanto vale  $n*m/(p+1)$  ?

Quanto vale  $n*m/p+1$  ?

# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
... un po' meno	*	/	%
... ancora meno	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo

... niente '[' ne' '{' ok?

Quanto vale  $(n+m)*p$  ?  $7*6$

( $n=3$ ,  $m=4$ ,  $q$ ,  $p=6$ ;) )

Quanto vale  $n+m-p$  ?  $7-6$

Quanto vale  $n*m+p$  ?  $12+6$

Quanto vale  $n*m/p$  ?  $12/6$  cioe` 2

Quanto vale  $n*m/(p+1)$  ?  $12/7$  cioe` 1

Quanto vale  $n*m/p+1$  ?

# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
... un po' meno	*	/	%
... ancora meno	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo

... niente '[' ne' '{' ok?

Quanto vale  $(n+m)*p$  ?  $7*6$

( $n=3$ ,  $m=4$ ,  $q$ ,  $p=6$ ;) )

Quanto vale  $n+m-p$  ?  $7-6$

Quanto vale  $n*m+p$  ?  $12+6$

Quanto vale  $n*m/p$  ?  $2$

Quanto vale  $n*m/(p+1)$  ?  $12/7$  cioe` 1

Quanto vale  $n*m/p+1$  ?  $(12/6)+1$  cioe` 3

# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
	*	/	%
	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo

... niente '[' ne' '{' ok?

Quanto vale  $2 + 3 * 12$ ?

$((9-6)*2)/4$

$(9-6)*2/4$

$(1+3)7$

$(9-6)*(2/4)$

$9/2*3$

$1+1*1+1$

$-1-p$



$(n=3, m=4, q, p=6;)$

# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
	*	/	%
	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo

... niente '[' ne' '{' ok?

Quanto vale  $2 + 3 * 12$ ? **2+36**

$((9-6)*2)/4$  *remember, sono interi*

$(9-6)*2/4$  *come cambia la valutazione?*

$(1+3)7$

$(9-6)*(2/4)$

$9/2*3$

$1+1*1+1$

$-1-p$

**(n=3, m=4, q, p=6;)**

# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
	*	/	%
	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo

... niente '[' ne' '{' ok?

Quanto vale  $2 + 3 * 12$ ? **2+36**

$((9-6)*2)/4$  **1**

$(9-6)*2/4$  *come cambia la valutazione?*

$(1+3)7$

$(9-6)*(2/4)$

$9/2*3$

$1+1*1+1$

$-1-p$

**(n=3, m=4, q, p=6;)**

# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
	*	/	%
	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo

... niente '[' ne' '{' ok?

Quanto vale  $2 + 3 * 12$ ? **2+36**

$((9-6)*2)/4$  **1**

$(9-6)*2/4$  non cambia ... **1**

$(1+3)7$  ma che hai scritto???

$(9-6)*(2/4)$

$9/2*3$

$1+1*1+1$

$-1-p$

**(n=3, m=4, q, p=6;)**

# Valutazione di Espressione - PRECEDENZA

## Regole di priorit  (o precedenza)

<b>Prec. max</b>	-	++	--
	*	/	%
	+	-	
<b>Prec. min</b>	=	+=	-= *= /=

Si valutano le sottoespressioni denotate dagli operatori con maggiore precedenza, poi quelle ottenute con gli operatori a minore precedenza, in ordine.

Per espressioni composte con pi  operatori della medesima precedenza, si procede da sinistra verso destra.

Le parentesi '(' e ')' permettono di organizzare la valutazione come preferiamo

... niente '[' ne' '{' ok?

Quanto vale  $2 + 3 * 12$ ? **2+36**

$((9-6)*2)/4$  **1**

$(9-6)*2/4$  **1**

$(1+3)7$  niente ... scritta male ... ☹

$(9-6)*(2/4)$  **3\*0**

$9/2*3$  **4\*3**

$1+1*1+1$  **3**

(n=3, m=4, q, p=6;)

$-1-p$  **-7**

(NB c'  il - che significa "negativo" e c'  il - che significa "meno":

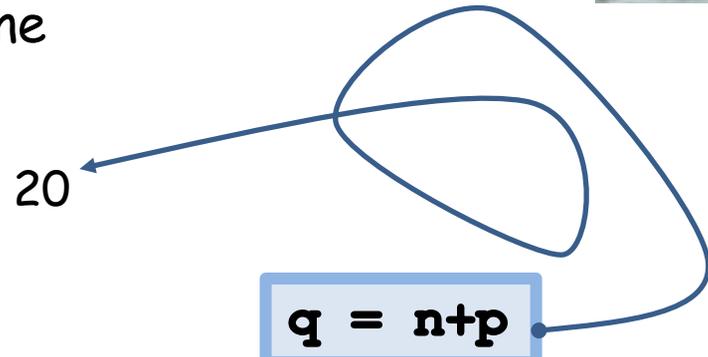
due distinti operatori; il primo unario, in secondo binario)



# Un'espressione particolare: operatore di assegnazione

$n = 10$  e' un'espressione, e puo' essere usata come tale, oltre che per costruire un'istruzione di assegnazione.

```
n = 10;  
p = n;  
q = n+p;
```



**Istruzione di assegnazione:**  
A destra c'e' un'espressione (right value); l'espressione viene valutata e il valore viene assegnato alla variabile indicata nel left value.  
Dopo queste tre istruzioni n contiene 10, p idem, e q 20

**Espressione di assegnazione:**  
La valutazione di questa espressione restituisce un valore: quello usato nell'assegnazione alla variabile.  
Con n e p uguali a 10, la valutazione dell'espressione di assegnazione qui sopra

- produce l'effetto di assegnare 20 a q,
- e restituisce il valore 20.

```
printf("bla %d\n", q=n+p);
```



```
n = m = p+q
```



# Un'espressione particolare: operatore di assegnazione



`n = 10` e' un'espressione, e puo' essere usata come tale, oltre che per costruire un'istruzione di assegnazione.

`q = n+p`

```
n = 10;  
p = n;  
q = n+p;
```

## Espressione di assegnazione:

La valutazione di questa espressione restituisce un valore: quello usato nell'assegnazione alla variabile. Con `n` e `p` uguali a 10,

la valutazione dell'espressione di assegnazione qui sopra

- produce l'effetto di assegnare 20 a `q`, e
- restituisce il valore 20.

viene stampato il risultato di `q = n+p` cioè viene stampato il valore di `(n+p)`

```
printf("bla %d\n", q=n+p);
```

Stampa 20.

```
n = m = p+q
```

viene eseguita l'assegnazione "`n = risultato di (m=p+q)`"

`p+q` e' 30;

- `m=p+q` assegna 30 ad `m` e
- restituisce 30.
- Il risultato della valutazione di `m=p+q` (30) viene assegnato a `n`
- e l'espressione complessiva (`n=m=p+q`) restituisce 30

# Operatore % (modulo)

Magari fa comodo sapere che questo operatore su interi restituisce il resto della divisione tra gli interi, come segue

$$n \% m = n - (n / m) * m$$

**Esempi**

$$7 \% 2 = 1$$

$$7 - (7 / 2) * 2$$

$$7 \% -2 = 1$$

$$7 - (7 / -2) * -2$$

$$-7 \% 2 = -1$$

$$-7 - (-7 / 2) * 2$$

$$-107 \% 100 = -7$$

$$-107 - (-107 / 100) * 100$$

provare almeno queste quattro espressioni in un programma  
e poi provarne altre Quattro che sembrano interessanti

# Operatori unari

Cioè operatori che gestiscono un solo operando per ottenere un'espressione

## - MENO (NEGATIVO)

L'espressione `-i` valuta al valore  $(-1) * i$

Es. `a=100.33;`

```
printf ("bla ... %g\n", -a) stampa -100.33
```

**++** operatore di PREincremento o POSTincremento (di 1)

**--** operatore di PRE/POST decremento (di 1)

# Operatori unari

Cioè operatori che gestiscono un solo operando per ottenere un'espressione (i è una variabile che usiamo come esempio)

- MENO (NEGATIVO)

++ operatore di PREincremento o POSTincremento (di 1)

La valutazione dell'espressione **++i** produce

- l'incremento di i, che diventa (i+1)
- la restituzione del valore di i **DOPO** l'incremento

-- operatore di PRE/POST decremento (di 1)





# Operatori unari

Cioè operatori che gestiscono un solo operando per ottenere un'espressione (i è una variabile che usiamo come esempio)

## - MENO (NEGATIVO)

L'espressione `-i` valuta al valore  $(-1)*i$

Es. `a=100.33;`  
`printf ("bla ... %g\n", -a)` stampa `-100.33`

## ++ operatore di PREincremento o POSTincremento (di 1)

La valutazione dell'espressione `++i` produce

- l'incremento di `i`, che diventa `(i+1)`
- la restituzione del valore di `i` **DOPO** l'incremento

La valutazione dell'espressione `i++` produce

- l'incremento di `i`, che diventa `(i+1)`
- la restituzione del valore di `i` **PRIMA** dell'incremento

## -- operatore di PRE/POST decremento (di 1)

`--i` predecremento: decrementa `i` e valuta al valore di `i` dopo il decremento

`i--` postdecremento: valuta al valore attuale di `i` e decrementa `i`

# ++ -- esercizio per casa

sol in complementi  
suggerimento in fondo

Scrivere un programma che produca questo output ... circa

```

+-----+
| ESEMPI DI PRE- E POST-INCREMENTO |
+-----+

Caro/a utente, dammi un intero, che chiamiamo, per comodita', n:
45

QUI n vale 45 ...
il valore dell'espressione di POST-INCRemento di n e` 45 ... e ora n vale 46

-----

QUI n vale 46 ...
il valore dell'espressione di PRE-INCRemento di n e` 47 ... e ora n vale 47

-----

QUI n vale 47 ...
il valore dell'espressione di POST-DECRemento di n e` 47 ... e ora n vale 46

-----

QUI n vale 46 ...
infine, il valore dell'espressione di PRE-DECRemento di n e' 45 ... e ora n vale 45

Fine programma: cari saluti.
```

# Assegnazione composta

Assegnazioni speciali, per evitare di scrivere troppo ...

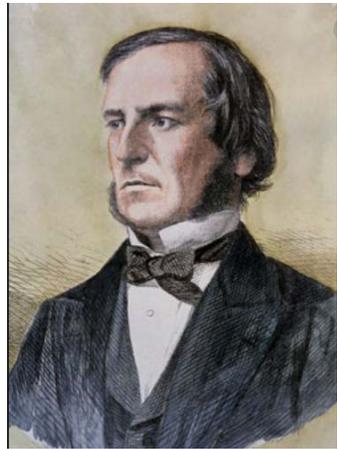
$a += m$       equivale a       $a = a+m$

$a -= m$       equivale a       $a = a-m$

$a *= m$       equivale a       $a = a*m$

$a /= m$       equivale a       $a = a/m$

# Algebra di Boole



E' parte della Logica Matematica,  
ed e' anche alla base della progettazione dei circuiti logici ...  
Si puo' immaginare come un tipo, con i suoi valori e i suoi operandi:  
**valori logici** (valori di verita') e **operatori logici**

Valori Logici = {TRUE, FALSE}

una espressione logica **Esp** ha valore TRUE oppure valore FALSE  
**Esempio Esp="siamo piu' di 50"**

Operatori Logici (o Booleani)

**NOT** negazione  
se l'espressione logica **Esp** ha valore TRUE,  
allora l'espressione

**NOT(Esp) ha valore FALSE**  
**(NOT(Esp) == FALSE)**

se l'espressione logica **Esp** ha valore FALSE,  
allora l'espressione

**NOT(Esp) ha valore TRUE**  
**(NOT(Esp) == TRUE)**

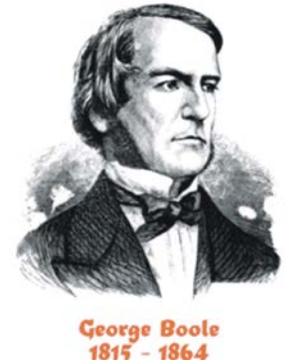
# Algebra di Boole

Esempio Esp="siamo piu` di 50" AND "Siamo meno di 100"

Esempio Esp="siamo piu` di 50" AND "Siamo tutti con i capelli ricci"

Esempio Esp="siamo meno di 20" AND "Siamo tutti con i capelli ricci"

Esempio Esp="siamo meno di 20" AND "Siamo piu` di 10"



## Operatori Logici (o Booleani)

**AND** congiunzione  
date due espressioni logiche, **Esp1** , **Esp2** ,  
**Esp1 AND Esp2** e` **TRUE**  
se e solo se  
sia **Esp1** che **Esp2** sono **TRUE**  
**ALTRIMENTI** la congiunzione ha valore **FALSE**

# Algebra di Boole

Esempio Esp="siamo piu` di 50" AND "Siamo meno di 100"

Esempio Esp="siamo piu` di 50" AND "Siamo tutti con i capelli ricci"

Esempio Esp="siamo meno di 20" AND "Siamo tutti con i capelli ricci"

Esempio Esp="siamo meno di 20" AND "Siamo piu` di 10"



George Boole  
1815 - 1864

## Operatori Logici (o Booleani)

**AND** congiunzione  
date due espressioni logiche, **Esp1** , **Esp2**,  
**Esp1 AND Esp2** e` **TRUE**  
se e solo se  
sia **Esp1** che **Esp2** sono **TRUE**  
**ALTRIMENTI** la congiunzione ha valore **FALSE**

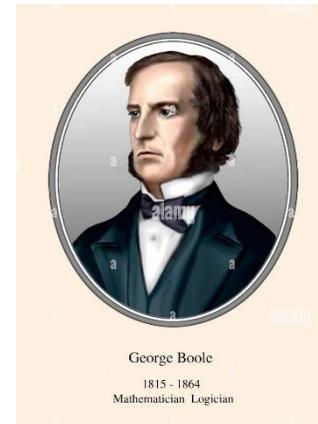
# Algebra di Boole

Esempio Esp="siamo piu` di 50" OR "Siamo meno di 100"

Esempio Esp="siamo piu` di 50" OR "Siamo tutti con i capelli ricci"

Esempio Esp="siamo meno di 20" OR "Siamo tutti con i capelli ricci"

Esempio Esp="siamo meno di 20" OR "Siamo piu` di 10"



## Operatori Logici (o Booleani)

**OR**            disgiunzione  
                  date due espressioni logiche, **Esp1** , **Esp2** ,  
**Esp1 OR Esp2** e` **TRUE**  
                  se e solo se  
                  almeno una tra **Esp1** e **Esp2** ha valore **TRUE**  
  
**ALTRIMENTI** la congiunzione ha valore **FALSE**

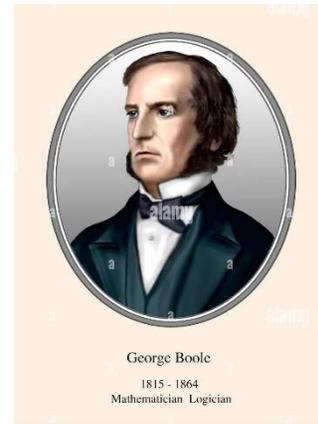
# Algebra di Boole

Esempio Esp="siamo piu` di 50" OR "Siamo meno di 100"

Esempio Esp="siamo piu` di 50" OR "Siamo tutti con i capelli ricci"

Esempio Esp="siamo meno di 20" OR "Siamo tutti con i capelli ricci"

Esempio Esp="siamo meno di 20" OR "Siamo piu` di 10"



## Operatori Logici (o Booleani)

**OR**        disgiunzione  
            date due espressioni logiche, **Esp1** , **Esp2** ,  
**Esp1 OR Esp2** e` **TRUE**  
            se e solo se  
            almeno una tra **Esp1** e **Esp2** ha valore **TRUE**  
  
**ALTRIMENTI** la congiunzione ha valore **FALSE**

# Algebra di Boole

E' parte della Logica Matematica,  
ed è anche alla base della progettazione dei circuiti logici ...  
Si può immaginare come un tipo, con i suoi valori e i suoi operandi:  
**valori logici** (valori di verità) e **operatori logici**

No general method for the solution of questions in the theory of probabilities can be established which does not explicitly recognise, not only the special numerical bases of the science, but also those universal laws of thought which are the basis of all reasoning, and which, whatever they may be as to their essence, are at least mathematical as to their form.  
G.Boole

Valori Logici = {TRUE, FALSE}

una espressione logica **Esp** ha valore TRUE oppure valore FALSE

**Esempio** **Esp="siamo più di cinque"**

Operatori Logici (o Booleani)

**NOT** negazione

se l'espressione logica **Esp** ha valore TRUE, **NOT(Esp) = FALSE**

se l'espressione logica **Esp** e' FALSE, **NOT(Esp) è TRUE**

**AND** congiunzione

date due espressioni logiche, **Esp1**, **Esp2**,

**Esp1 AND Esp2** è TRUE se e solo se sia **Esp1** che **Esp2** sono TRUE

(altrimenti ... FALSE)

**OR** disgiunzione

date due espressioni logiche, **Esp1**, **Esp2**,

**Esp1 OR Esp2** e' TRUE se almeno una tra **Esp1** e **Esp2** è TRUE

(altrimenti ... FALSE)

54/89

# Operatore AND

Qui usiamo una **TABELLA DI VERITA'** per definirlo

Esp1	Esp2	(Esp1 AND Esp2)
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

TRUE AND TRUE == TRUE

TRUE AND FALSE == FALSE

FALSE AND TRUE == FALSE

sia Esp1 = "siamo più di cinque"  
sia Esp2 = "siamo meno di dieci"

- se siamo 8  
allora Esp1=TRUE Esp2=TRUE e Esp1 AND Esp2 == 😊
- se siamo 3  
allora Esp1=FALSE Esp2=TRUE e Esp1 AND Esp2 == 😊
- se siamo 11  
allora Esp1=TRUE Esp2=FALSE e Esp1 AND Esp2 == 😊

# Operatore AND

## AND: TABELLA DI VERITA'

Esp1	Esp1	(Esp1 AND Esp2)
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

TRUE AND TRUE == TRUE

TRUE AND FALSE == FALSE

FALSE AND FALSE == FALSE

sia Esp1 = "siamo piu' di cinque" e sia Esp2 = "siamo meno di dieci"

- se siamo 8  
allora Esp1==TRUE Esp2==TRUE e Esp1 AND Esp2 == TRUE
- se siamo 3  
allora Esp1==FALSE Esp2==TRUE e Esp1 AND Esp2 == FALSE
- se siamo 11  
allora Esp1==TRUE Esp2==FALSE e Esp1 AND Esp2 == ~~TRUE~~ **FALSE**

# Operatore AND

AND: TABELLA DI VERITA'

Esp1	Esp1	(Esp1 AND Esp2)
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

TRUE AND TRUE == TRUE

TRUE AND FALSE == FALSE

FALSE AND TRUE == FALSE

TRUE AND NOT(TRUE) e`

TRUE AND NOT(FALSE) e`

NOT (TRUE AND FALSE) e`

NOT (TRUE AND TRUE) e`



# Operatore AND

## AND: TABELLA DI VERITA'

Esp1	Esp2	(Esp1 AND Esp2)
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

TRUE AND TRUE == TRUE

TRUE AND FALSE == FALSE

FALSE AND TRUE == FALSE

TRUE AND NOT(TRUE) e' FALSE

TRUE AND NOT(FALSE) e' TRUE AND TRUE == TRUE

NOT (TRUE AND FALSE) e' TRUE

NOT (TRUE AND TRUE) e' FALSE

# Operatore NOT

Qui usiamo una **TABELLA DI VERITA'** per definirlo

<b>Esp</b>	<b>NOT ( Esp )</b>
<b>TRUE</b>	<b>FALSE</b>
<b>FALSE</b>	<b>TRUE</b>

**NOT(FALSE) = TRUE**  
**NOT(TRUE) = FALSE**

sia **Esp = "siamo più di dieci"**  
e siamo 8, quindi **Esp è FALSE**  
allora

**NOT(Esp) == TRUE**

inoltre

**NOT (NOT(Esp)) == FALSE**

# Operatore OR

Qui usiamo una **TABELLA DI VERITA'** per definirlo

Esp1	Esp1	(Esp1 OR Esp2)
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

TRUE OR TRUE = TRUE

TRUE OR FALSE = TRUE

FALSE OR FALSE = FALSE

sia Esp1 = "siamo piu' di cinque" e sia Esp2 = "siamo affamat\* di conoscenza"

- se siamo 8 e non abbiamo fame  
allora

Esp1 OR Esp2 ==

- se siamo 3 e siamo affamati  
allora

Esp1 OR Esp2 ==

- se siamo 3 e non siamo affamati  
allora

Esp1 OR Esp2 ==

# Operatore OR

OR: TABELLA DI VERITA'

Esp1	Esp1	(Esp1 OR Esp2)
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

TRUE OR TRUE = TRUE

FALSE OR FALSE == TRUE

FALSE OR FALSE == FALSE

sia Esp1 = "siamo piu` di cinque" e sia Esp2 = "siamo affamat\* di conoscenza"

- se siamo 8 e non abbiamo fame  
allora

Exp1 OR Exp2 == TRUE

- se siamo 3 e siamo affamati  
allora

Exp1 OR Exp2 == TRUE

- se siamo 3 e non siamo affamati  
allora

Exp1 OR Exp2 == FALSE

# Operatore OR

OR: TABELLA DI VERITA'

Esp1	Esp1	(Esp1 OR Esp2)
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

TRUE OR TRUE = TRUE

TRUE OR FALSE = TRUE

FALSE OR FALSE = FALSE

TRUE OR NOT(TRUE) e' (TRUE OR FALSE) == TRUE

TRUE OR NOT(FALSE) e' (TRUE OR TRUE) == TRUE

NOT (TRUE OR FALSE) e' NOT(TRUE) == FALSE

NOT (FALSE OR FALSE) e' NOT(FALSE) == TRUE

# Operatore OR

Per definirlo usiamo una TABELLA DI VERITA'

Esp1	Esp1	(Esp1 OR Esp2)
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

TRUE OR TRUE = TRUE

TRUE OR FALSE = TRUE

FALSE OR FALSE = FALSE

TRUE OR NOT(TRUE)

e' (TRUE OR FALSE) == TRUE

TRUE OR NOT(FALSE)

e' (TRUE OR TRUE) == TRUE

NOT (TRUE OR FALSE)

e' NOT(TRUE) == FALSE

NOT (FALSE OR FALSE)

e' NOT(FALSE) == TRUE

**NOT (Esp1 AND Esp2) == NOT(Esp1) OR NOT (Esp2) ... ricordare mcd ...**

# Algebra di Boole and the C

espressioni logiche.

costituenti fondamentali

- `!` `||` e `&&`, realizzano NOT, OR ed AND.
- Il valore FALSE viene rappresentato dal valore 0:
- Il valore TRUE viene rappresentato dal valore 1 (o, più in generale, da qualsiasi valore diverso da zero):

espressioni logiche

Esp1 OR Esp2

Esp1 AND Esp2

NOT(Esp1)

(Esp1 OR Esp2) AND (NOT(Esp3) OR Esp1)



le medesime, in C

# Algebra di Boole and the C

In C le **decisioni** prese nelle istruzioni strutturate **condizionale** ed **iterativa** (o **di ripetizione**) sono scritte come **espressioni logiche**.

Un'espressione logica viene costruita come vediamo tra poco; ma i suoi costituenti fondamentali li possiamo dire fin da ora:

- Gli operatori logici del linguaggio sono **!** e **&&**, che realizzano, rispettivamente, NOT, OR ed AND.
- Il valore FALSE viene rappresentato dal valore 0:  
se un'espressione logica viene valutata a 0, è falsa.
- Il valore TRUE viene rappresentato dal valore 1 (o, più in generale, da qualsiasi valore diverso da zero):  
se un'espressione logica viene valutata uguale ad un valore diverso da zero, è vera.

## espressioni logiche

Esp1 OR Esp2

Esp1 AND Esp2

NOT(Esp1)

(Esp1 OR Esp2) AND (NOT(Esp3) OR Esp1)

## le medesime, in C

exp1 ☺ exp2

exp1 ☺ exp2

☺

☺

# Algebra di Boole and the C

In C le **decisioni** prese nelle istruzioni strutturate **condizionale** ed **iterativa** (o *di ripetizione*) sono scritte come **espressioni logiche**.

Un'espressione logica viene costruita come vediamo tra poco; ma i suoi costituenti fondamentali li possiamo dire fin da ora:

- Gli operatori logici del linguaggio sono **!**, **||** e **&&**, che realizzano, rispettivamente, NOT, OR ed AND.
- Il valore FALSE viene rappresentato dal valore 0:  
se un'espressione logica viene valutata a 0, e' falsa.
- Il valore TRUE viene rappresentato dal valore 1 (o, piu' in generale, da qualsiasi valore diverso da zero):  
se un'espressione logica viene valutata uguale ad un valore diverso da zero, è vera.

## espressioni logiche

Esp1 OR Esp2

Esp1 AND Esp2

NOT(Esp1)

(Esp1 OR Esp2) AND (NOT(Esp3) OR Esp1)

## le medesime, in C

exp1 || exp2

exp1 && exp2

!exp1

(exp1 || exp2) &&

(!exp3 || exp1)

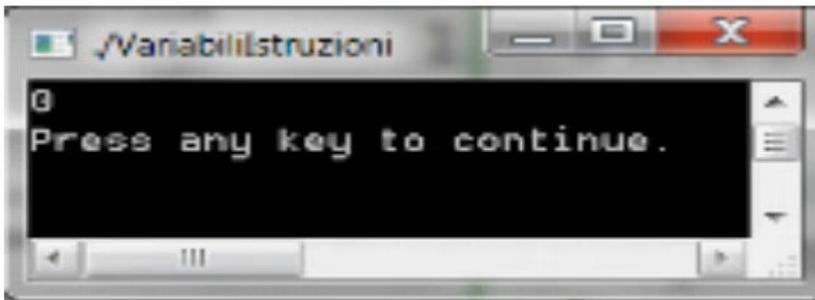
# Algebra di Boole and the C

In C le **decisioni** prese nelle istruzioni strutturate **condizionale** ed **iterativa** (o *di ripetizione*) sono scritte come **espressioni logiche**.

Un'espressione logica viene costruita come vediamo tra poco; ma i suoi costituenti fondamentali li possiamo dire fin da ora:

- Gli operatori logici del linguaggio sono **!**, **||** e **&&**, che realizzano, rispettivamente, NOT, OR ed AND.
- Il valore FALSE viene rappresentato dal valore 0:  
se un'espressione logica viene valutata a 0, e' falsa.
- Il valore TRUE viene rappresentato dal valore 1 (o, piu' in generale, da qualsiasi valore diverso da zero):  
se un'espressione logica viene valutata uguale ad un valore diverso da zero, e' vera.

```
int main() {  
    printf("%d", 1 && 0);  
}
```



```
int main() {  
    printf("%d", (1 && 0) || (1 || 0));  
}
```



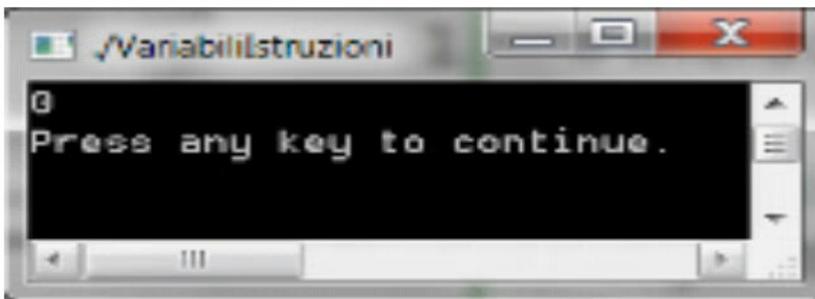
# Algebra di Boole and the C

In C le **decisioni** prese nelle istruzioni strutturate **condizionale** ed **iterativa** (o *di ripetizione*) sono scritte come **espressioni logiche**.

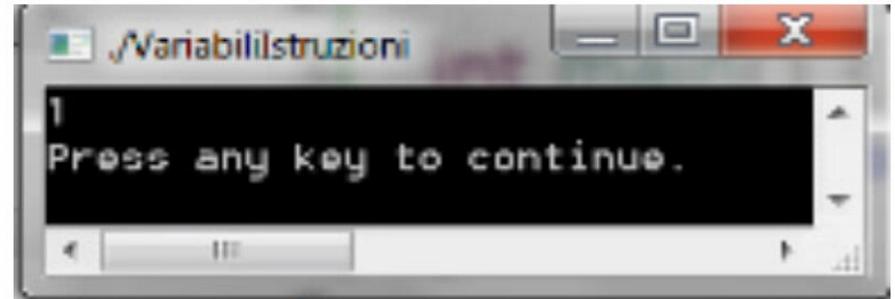
Un'espressione logica viene costruita come vediamo tra poco; ma i suoi costituenti fondamentali li possiamo dire fin da ora:

- Gli operatori logici del linguaggio sono **!**, **||** e **&&**, che realizzano, rispettivamente, NOT, OR ed AND.
- Il valore FALSE viene rappresentato dal valore 0:  
se un'espressione logica viene valutata a 0, e' falsa.
- Il valore TRUE viene rappresentato dal valore 1 (o, piu' in generale, da qualsiasi valore diverso da zero):  
se un'espressione logica viene valutata uguale ad un valore diverso da zero, e' vera.

```
int main() {  
    printf("%d", 1 && 0);  
}
```



```
int main() {  
    printf("%d", (1 && 0) || (1 || 0));  
}
```



# Operatori Relazionali

Si tratta di operatori binari, che verificano l'esistenza di una RELAZIONE tra due espressioni e restituiscono il **valore di verità** della relazione (cioè se la relazione ESISTE (valore di verità **1**) oppure no (valore **0**))

==	uguale
!=	diverso
<	minore
>	maggiore
<=	minore-uguale
>=	maggiore-uguale

$3 == 9$

$3 == (7/2)$

$5 != (18/4)$

$5.5 > 5$

$2 >= 4 / 2$

$1/2 >= 0$

$-1/2 >= 0$



# Operatori Relazionali

Si tratta di operatori binari, che verificano l'esistenza di una RELAZIONE tra due espressioni e restituiscono il **valore di verità** della relazione

(se la relazione ESISTE il valore di verità è **1**;                      se non esiste ... **0**)

==	uguale
!=	diverso
<	minore
>	maggiore
<=	minore-uguale
>=	maggiore-uguale

3 == 9	0
3 == (7/2)	1
5 != (18/4)	1
5.5 > 5	1
2 >= 4 / 2	1
1/2 >= 0	1
-1/2 >= 0	1



# Espressioni Logiche

Sono espressioni ottenute combinando operatori LOGICI (e relazionali), con variabili, 1, 0 (e anche funzioni, che restituiscano valori assimilabili a valori di verita' (0 / diverso\_da\_zero).

Vengono valutate restituendo un valore di verità (1=VERO/0=FALSO)

`(x>100) || (x < 100)`      1 se x e` diverso da 100

`(x>100) || (x < -100)`      1 se x e` molto piccolo o molto grande

`(x>100) && (x < -100)`      1 se x e` molto piccolo e molto grande ... QUINDI ... 0

`(x>100) && (x < -100) && 1`      come sopra, con un AND TRUE in fondo, ma dato che la prima parte e` FALSE ... tutto valuta a 0

`((x>100) && (x < -100)) || 1`      come sopra, **MA** con un OR TRUE: in fondo. La prima sottoespressione e` FALSE ma la seconda e` TRUE, quindi l'OR valuta a 1

# Espressioni Logiche, altri esempi

Sono espressioni ottenute combinando operatori LOGICI (e relazionali), con variabili, 1, 0

(e anche funzioni, che restituiscano valori assimilabili a valori di verita' - 0, diverso\_da\_zero).

Vengono valutate restituendo un valore di verita' (1=VERO/0=FALSO)

```
(x>100) || (x < 100)      1 se x e` diverso da 100
(x>100) || (x < -100)    1 se x e` molto piccolo o molto grande
(x>100) && (x < -100)    1 se x e` molto piccolo e molto grande ... QUINDI 0
(x>100) && (x < -100) && 1 ... 0
((x>100) && (x < -100)) || 1 ... 1
```

```
(ris%n != 0) || (ris%m != 0)
```

ris non divide n OR ris non divide m

```
((a<b) && (b<c)) || ((c<b) && (b<a))
```

vero se a,b,c ordinate in qualche direzione

# Espressioni Logiche - ancora MCD

Sono espressioni ottenute combinando operatori LOGICI (e relazionali), con variabili, 1, 0

(e anche funzioni, che restituiscano valori assimilabili a valori di verita' - 0, diverso\_da\_zero).

Vengono valutate restituendo un valore di verita' (1=VERO/0=FALSO)

```
(x>100) || (x < 100)      1 se x e' diverso da 100
(x>100) || (x < -100)    1 se x e' molto piccolo o molto grande
(x>100) && (x < -100)    1 se x e' molto piccolo e molto grande ... cioe' vale 0
(x>100) && (x < -100) && 1    ... 0
((x>100) && (x < -100)) || 1    ... 1
```

```
(ris%n != 0) || (ris%m != 0)   ris non divide n OR ris non divide m
```

come riscrivere la condizione di ripetizione dell'esercizio MCD, a partire dall'espressione che segue?

```
(ris%n == 0) && (ris%m == 0)
```

# Espressioni Logiche - ancora MCD

Sono espressioni ottenute combinando operatori LOGICI (e relazionali), con variabili, 1, 0

(e anche funzioni, che restituiscano valori assimilabili a valori di verita' - 0, diverso\_da\_zero).

Vengono valutate restituendo un valore di verita' (1=VERO/0=FALSO)

```
(x>100) || (x < 100)      1 se x e' diverso da 100
(x>100) || (x < -100)     1 se x e' molto piccolo o molto grande
(x>100) && (x < -100)     1 se x e' molto piccolo e molto grande ... cioe' vale 0
(x>100) && (x < -100) && 1 ... 0
((x>100) && (x < -100)) || 1 ... 1
```

```
(ris%n != 0) || (ris%m != 0)   ris non divide n OR ris non divide m
```

**l'espressione equivalente e'**

```
!( (ris%n == 0) && (ris%m == 0) )
```

**condizione di ripetizione VERA se e solo se ris NON divide contemporaneamente n ed m (cioe` NON puo` essere l'MCD)**



# Precedenze nelle espressioni logiche

In ordine crescente

min || OR  
&& AND  
.  
.  
Relazionali uguaglianza ( == != )  
.  
Relazionali confronto ( > < >= <= )  
max ! NOT

per il resto, da sinistra verso destra

a <= 20 || 1 && !cond || x != 5

a <= 20 || 1 && (!cond) || x != 5 !cond e' la prima ad essere valutata

(a <= 20) || 1 && (!cond) || x != 5 !poi il <=

(a <= 20) || 1 && (!cond) || (x != 5)

(a <= 20) || (1 && (!cond)) || (x != 5)

((a <= 20) || (1 && (!cond))) || (x != 5)

(( (a <= 20) || (1 && (!cond))) || (x != 5))

# Esercizio: Quanto vale?

`1 || 1 && 0` ?

`((n || m) && (1 && m)) && m) && 0` ?

`(a && b) || a` 1 se e solo se ...

`(!a && b) || a` 1 se ...

per le ultime due è bene fare la tabella di verità ...



# Quanto valeva l'ultima?

`1 || 1 && 0` `1`

`((n || m) && (1 && m)) && m` `0`

`(a && b) || a` `1` sse `a` diverso da `0`

`(!a && b) || a` ☺

<code>a</code>	<code>b</code>	<code>a &amp;&amp; b</code>	<code>(a &amp;&amp; b)    a</code>
<code>0</code>	<code>0</code>	<code>0</code>	<code>0</code>
<code>1</code>	<code>1</code>	<code>1</code>	<code>1</code>
<code>1</code>	<code>0</code>	<code>0</code>	<code>1</code>
<code>0</code>	<code>1</code>	<code>0</code>	<code>0</code>

# Quanto vale?

`1 || 1 && 0` `1`

`((n || m) && (1 && m)) && m` `0`

`(a && b) || a` `1` se a diverso da 0

`(!a && b) || a` `1` se uno tra a e b e ` != 0

a	b	!a && b	(!a && b)    a
			😊



# Quanto vale?

`1 || 1 && 0` `1`

`((n || m) && (1 && m)) && m` `0`

`(a && b) || a` `1` sse `a` diverso da `0`

`(!a && b) || a` `1` se uno tra `a` e `b` e `!= 0`

<code>a</code>	<code>b</code>	<code>!a &amp;&amp; b</code>	<code>(!a &amp;&amp; b)    a</code>
<code>0</code>	<code>0</code>	<code>0</code>	
<code>1</code>	<code>1</code>	<code>0</code>	
<code>1</code>	<code>0</code>	<code>0</code>	
<code>0</code>	<code>1</code>	<code>1</code>	

# Quanto vale?

`1 || 1 && 0` `1`

`((n || m) && (1 && m)) && m` `0`

`(a && b) || a` `1` se `a` diverso da `0`

`(!a && b) || a` `1` se uno tra `a` e `b` e `!= 0`

<code>a</code>	<code>b</code>	<code>!a &amp;&amp; b</code>	<code>(!a &amp;&amp; b)    a</code>
<code>0</code>	<code>0</code>	<code>0</code>	<code>0</code>
<code>1</code>	<code>1</code>	<code>0</code>	<code>1</code>
<code>1</code>	<code>0</code>	<code>0</code>	<code>1</code>
<code>0</code>	<code>1</code>	<code>1</code>	<code>1</code>

# Altre prove ... cioè` altri esercizi

```
int n, m, p;
```

```
n=m=1;
```

```
p = n && !m;
```

```
m = (m && p) && !m;
```

```
n = !p;
```

```
printf ("Caro/a utente, il valore di n %c %d\n", 138, n);
```

```
printf ("Caro/a utente, il valore di m %c %d\n", 138, m);
```

```
printf ("Caro/a utente, il valore di p %c %d\n", 138, p);
```

cosa stampa?

quanto vale

```
(x<=10) || (x>9)
```

```
int n, m
```

Scrivere un'istruzione che assegni a n il valore 1 sse m e' positivo

tutte le soluzioni nella slide successiva

```
int n, m, r
```

Assegnare ad r il valore 1 sse sia n che m sono maggiori di 19 ma m e' piu' grande di n

# Altre prove ...

```
int n, m, p;
```

```
n=m=1;
```

```
p = n && !m;
```

```
m = (m && p) && !m;
```

```
n = !p;
```

```
printf ("Caro/a utente, il valore di n %c %d\n", 138, n);
```

```
printf ("Caro/a utente, il valore di m %c %d\n", 138, m);
```

```
printf ("Caro/a utente, il valore di p %c %d\n", 138, p);
```

```
Caro/a utente, il valore di n è 1
```

```
Caro/a utente, il valore di m è 0
```

```
Caro/a utente, il valore di p è 0
```

```
Fine programma: cari saluti.
```

quanto vale

```
(x<=10) || (x>9)
```

1

```
int n, m
```

Scrivere un'istruzione che assegni a n il valore 1 sse m e' positivo

```
n = (m>0)
```

```
int n, m, r
```

Assegnare ad r il valore 1 sse sia n che m sono maggiori di 19 ma m e' piu' grande di n

```
r = (n>19) && (m>19) && (m>n)
```

# BTW ... esercizio per casa - soluzione

numero di cifre usate per scrivere i numerali: 4  
scrivere I numerali binari e I corrispondenti numeri

NUMERALE	NUMERO
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	sette
1000	8
1001	nove
1010	10
1011	11
1100	dodici
1101	13
1110	14
1111	15

**e il 16?**

# BTW ... esercizio per casa - soluzione

numero di cifre usate per scrivere i numerali: 4  
scrivere I numerali binari e I corrispondenti numeri

NUMERALE	NUMERO
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	sette
1000	8
1001	nove
1010	10
1011	11
1100	dodici
1101	13
1110	14
1111	15

E POI

scrivere i primi 36 numerali, indicando i relativi numeri. Scrivere numerali di 8 bit.

**e dov'e' il 16? Non c'e' ... servono 5 bit ... 10000**

# ++ -- suggerimento per l'esercizio

ESEMPI DI PRE- E POST-INCREMENTO

```
Caro/a utente, dammi un intero, che chiamiamo, per comodita', n:  
45
```

```
QUI n vale 45 ...  
il valore dell'espressione di POST-INCRemento di n e` 45 ... e ora n vale 46
```

```
printf ... n++
```

```
QUI n vale 46 ...  
il valore dell'espressione di PRE-INCRemento di n e` 47 ... e ora n vale 47
```

```
printf ... ++n
```

```
QUI n vale 47 ...  
il valore dell'espressione di POST-DECRemento di n e` 47 ... e ora n vale 46
```

```
printf ... n--
```

```
QUI n vale 46 ...  
infine, il valore dell'espressione di PRE-DECRemento di n e' 45 ... e ora n vale 45
```

```
printf ... --n
```

```
Fine programma: cari saluti.
```