Programmazione I

Il Linguaggio C

Strutture di Controllo

Daniel Riccio

Università di Napoli, Federico II

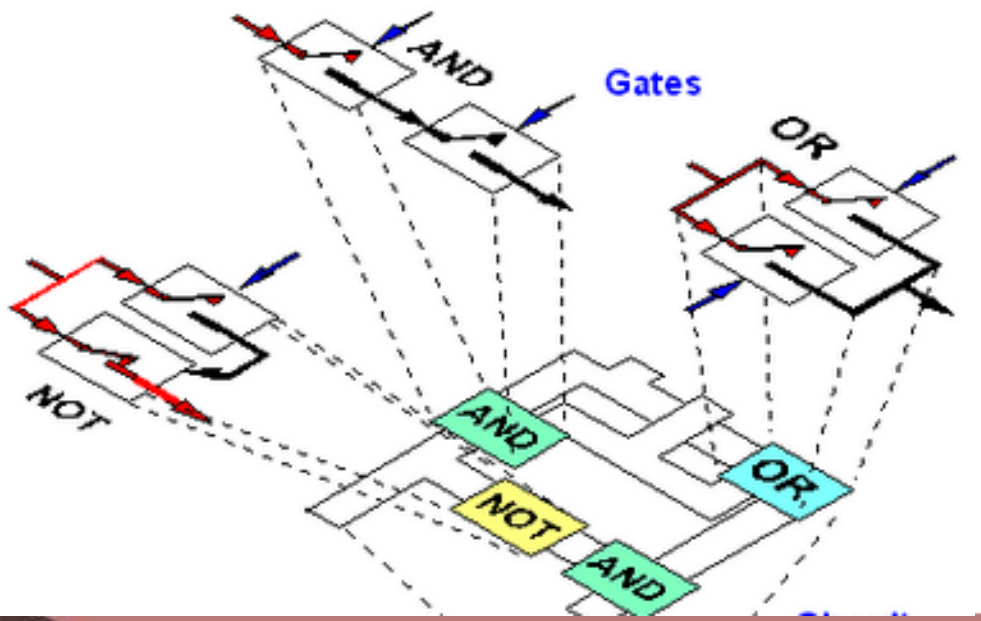
11 Ottobre 2021

Sommario

- Argomenti
  - Algebra di Boole
  - Le strutture condizionali
  - Le strutture iterative
  - Uso dei cicli nella programmazione
  - Esempi di Algoritmi e Programmi

La logica booleana

Nel 1847 George Boole introdusse un nuovo tipo di logica formale, basata esclusivamente su enunciati di cui fosse possibile verificare in modo inequivocabile la verità o la falsità.



George Boole (Lincoln, 2 novembre 1815 – Ballintemple, 8 dicembre 1864) è stato un matematico e logico britannico, ed è considerato il fondatore della logica matematica. La sua opera influenzò anche settori della filosofia e diede vita alla scuola degli algebristi della logica.

Variabili e operatori booleani

Variabili in grado di assumere solo due valori:

- **VERO**
- **FALSO**

Operatori unari (es. Not)

op : **B** → **B**

Operatori binari (es. And)

op : **B**×**B** → **B**

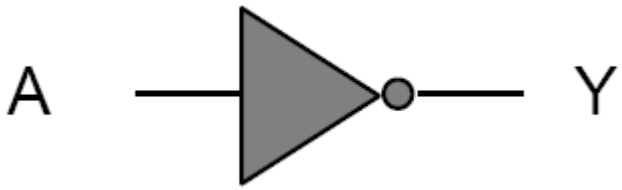
Descritti tramite una tavola della verità (per **N** operandi, la tabella ha 2^N righe che elencano tutte le possibili combinazioni di valori delle variabili indipendenti ed il valore assunto dalla variabile dipendente)

Operatore NOT

L'operatore **NOT** è un operatore logico (operatore booleano) di negazione di una proposizione

Data una proposizione logica **A** la negazione logica determina una seconda proposizione detta "**non A**" che risulta vera quando **A** è falsa e viceversa. L'operatore **NOT** è indicata con il simbolo **¬** oppure con un trattino posto al di sopra della variabile logica

A	$\overline{A}$
falso	vero
vero	falso

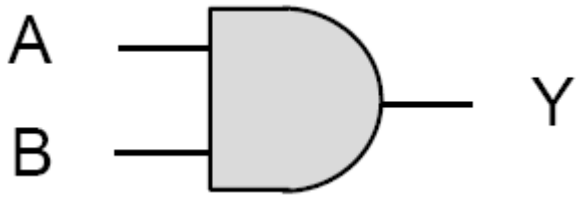


Operatore AND

L'operatore **AND** è un operatore logico di congiunzione logica tra due proposizioni

Date due proposizioni **A** e **B** la congiunzione logica determina una terza proposizione **C** che si manifesta vera soltanto quando entrambe le proposizioni sono vere. L'operatore **AND** è detto anche congiunzione logica o moltiplicazione logica ed è il simbolo ∧ o il simbolo X

A	B	A × B
falso	falso	falso
falso	vero	falso
vero	falso	falso
vero	vero	vero

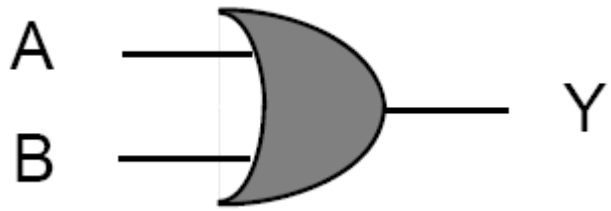


Operatore OR

L'operatore **OR** è un operatore logico di disgiunzione logica tra due proposizioni logiche.

Date due proposizioni **A** e **B**, la disgiunzione logica determina una terza proposizione logica **C** che si manifesta vera quando almeno una delle due proposizioni (A o B) è vera. L'operatore **OR** è rappresentato dal simbolo V o dal simbolo +

A	B	A + B
falso	falso	falso
falso	vero	vero
vero	falso	vero
vero	vero	vero

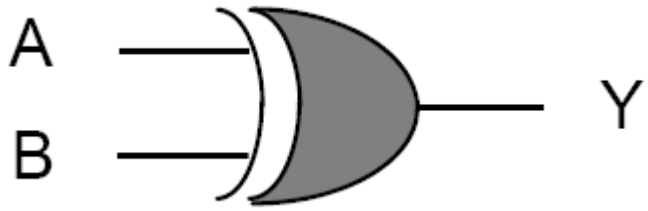


Operatore XOR

L'operatore **XOR** è un operatore logico di disgiunzione esclusiva tra due proposizioni logiche

Date due proposizioni logiche **A** e **B**, la disgiunzione esclusiva tra le due proposizioni è vera soltanto nel caso in cui è vera una delle due proposizioni. L'operatore **XOR** è rappresentato dal simbolo ⊕

A	B	A Å B
falso	falso	falso
falso	vero	vero
vero	falso	vero
vero	vero	falso



Operatori di manipolazione dei bit

Il C possiede una serie di **operatori** che possono agire direttamente sui bit delle variabili e costanti di tipo intero o carattere, dichiarate nel programma.

Si tratta di **4** operatori derivati direttamente dalle operazioni booleane di base e di altri 2 che eseguono l'operazione di shift (destra e sinistra) di un certo numero di bit.

I primi si prestano a "mascherare" o "commutare" i bit, i secondi possono essere utili nelle operazioni di divisione e moltiplicazione per 2.

Tranne uno, sono tutti operatori **binari**, che agiscono cioè su due espressioni.

Operatori di manipolazione dei bit

B \ A	0	1
0	0	0
1	0	1

A AND B

B \ A	0	1
0	0	1
1	1	1

A OR B

A	0	1
	1	0

NOT A

B \ A	0	1
0	0	1
1	1	0

A XOR B

Operatori di manipolazione dei bit

Operazione	Operatore	tipo
AND bit a bit	&	Binario
OR bit a bit	|	Binario
XOR bit a bit	^	Binario
NOT bit a bit (complemento a 1)	~	Unario
Shift a sinistra	<<	Binario
Shift a destra	>>	Binario

Operatori di manipolazione dei bit

```
unsigned char z, x = 3, y = 13;
```

x	0	0	0	0	0	0	1	1
y	0	0	0	0	1	1	0	1

```
z = x & y
```

AND

x	0	0	0	0	0	0	1	1
y	0	0	0	0	1	1	0	1
z	0	0	0	0	0	0	0	1

```
z = ~ x
```

NOT

x	0	0	0	0	0	0	1	1
z	1	1	1	1	1	1	0	0

Operatori di manipolazione dei bit

```
z = x | y
```

OR

0	0	0	0	0	0	1	1
0	0	0	0	1	1	0	1
0	**0**	**0**	**0**	**1**	**1**	**1**	**1**

```
z = x ^ y
```

XOR

0	0	0	0	0	0	1	1
0	0	0	0	1	1	0	1
0	**0**	**0**	**0**	**1**	**1**	**1**	**0**

Operatori di manipolazione dei bit

Equivalenti alla divisione/moltiplicazione per le potenze di 2.

Sintassi:

<operando>	**>>**	*<num. posizioni>*
<operando>	**<<**	*<num. posizioni>*

Significato: fai scorrere *<operando>* a destra/sinistra di un numero di bit pari a *<num. posizioni>*

Sia *<operando>* che *<num. posizioni>* devono essere valori interi.

Operatori di manipolazione dei bit

- I due operatori si comportano diversamente a seconda del tipo numerico

- Dati **`unsigned`**: equivale allo **shift logico**;
  - **<<** inserisce degli '0' nelle posizioni meno significative
  - **>>** inserisce degli '0' nelle posizioni più significative

- Dati **signed**: equivale allo **shift aritmetico**;
  - **<<** aggiunge degli '0' nelle posizioni meno significative, e mantiene inalterato il bit più significativo (segno)
  - **>>** inserisce un valore uguale al bit più significativo (bit di segno) mantenendo pertanto inalterato il segno;

Operatori di manipolazione dei bit

- Esempio:

```
unsigned char x = 15;   /*   x = 00001111   */
x = x << 2              /*   x = 00111100   (15 x 2^2 = 60)   */
x = x >> 2              /*   x = 00000011   (15 / 2^2 = 3)    */

char x = -15            /*   x = 11110001     */
x = x <<  2             /*   x = 11000100   (-15 x 2^2 = - 60)   */
x = x >>  2             /*   x = 11111100   (-15 / 2^2 = - 4)    */
```

- Anche per questi operatori è consentita la scrittura abbreviata:

x **<<=** 2 equivale a x = x **<<** 2

Codifica dei caratteri alfabetici

- Oltre ai numeri, molte applicazioni informatiche elaborano caratteri (simboli)
- Gli elaboratori elettronici trattano numeri
- Si codificano i caratteri e i simboli per mezzo di numeri
- Per poter scambiare dati (testi) in modo corretto, occorre definire uno standard di codifica

A	→	01000001
3	→	00110011
$	→	00100100

Codifica dei caratteri alfabetici

- Quando si scambiano dati, deve essere noto il tipo di codifica utilizzato
- La codifica deve prevedere le lettere dell'alfabeto, le cifre numeriche, i simboli, la punteggiatura, i caratteri speciali per certe lingue (æ, ã, ë, è,…)
- Lo standard di codifica più diffuso è il **codice ASCII**, per **American Standard Code for Information Interchange**

Codifica ASCII

- Definisce una tabella di corrispondenza fra ciascun carattere e un codice a **7 bit** (128 caratteri)
- I caratteri, in genere, sono rappresentati con **1 byte** (8 bit); i caratteri con il bit più significativo a 1 (quelli con codice dal 128 al 255) rappresentano un'estensione della codifica
- La tabella comprende sia **caratteri di controllo** (codici da 0 a 31) che **caratteri stampabili**
- I caratteri alfabetici/numerici hanno codici ordinati secondo l'ordine alfabetico/numerico

0 48	A 65	a 97
1 49	B 66	b 98
…….	…….	…….
8 56	Y 89	y 121
9 57	Z 90	z 122
cifre	maiuscole	minuscole

Caratteri di controllo ASCII

- I caratteri di controllo (codice da 0 a 31) hanno funzioni speciali
- Si ottengono o con tasti specifici o con una sequenza **Ctrl+carattere**

Ctrl	Dec	Hex	Code	Nota
^@	0	0	NULL	carattere nullo
^A	1	1	SOH	partenza blocco
……	…	…	……	……
^G	7	7	BEL	beep
^H	8	8	BS	backspace
^I	9	9	HT	tabulazione orizzontale
^J	10	A	LF	line feed (cambio linea)
^K	11	B	VT	tabulazione verticale
^L	12	C	FF	form feed (alim. carta)
^M	13	D	CR	carriage return (a capo)
……	…	…	……	……
^Z	26	1A	EOF	fine file
^[	27	1 B	ESC	escape
……	…	…	……	……
^_	31	1F	US	separatore di unità

Caratteri ACII stampabili

Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr
32	20	SPACE	48	30	0	64	40	@	80	50	P	96	60	`	112	70	p
33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
34	22	”	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
36	24	$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
39	27	’	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
40	28	(	56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
41	29	)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
42	2A	*	58	3A	:	74	4A	J	90	5A	Z	106	6A	j	122	7A	z
43	2B	+	59	3B	;	75	4B	K	91	5B	[	107	6B	k	123	7B	{
44	2C	,	60	3C	<	76	4C	L	92	5C	\	108	6C	l	124	7C	|
45	2D	-	61	3D	=	77	4D	M	93	5D	]	109	6D	m	125	7D	}
46	2E	.	62	3E	>	78	4E	N	94	5E	^	110	6E	n	126	7E	~
47	2F	/	63	3F	?	79	4F	O	95	5F	_	111	6F	o	127	7F	DEL

Nota: il valore numerico di una cifra può essere calcolato come differenza del suo codice ASCII rispetto al codice ASCII della cifra 0 (es. ‘5’–‘0’ = 53–48 = 5)

Tabella ASCII estesa

- I codici oltre il 127 non sono compresi nello standard originario

128	Ç	144	É	160	á	176	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	161	í	177	▒	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	162	ó	178	▓	195	├	211	╙	227	π	243	≤
131	â	147	ô	163	ú	179	│	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	164	ñ	180	┤	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	165	Ñ	181	╡	198	╞	214	╓	230	µ	246	÷
134	å	150	û	166	ª	182	╢	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	167	º	183	╖	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	168	¿	184	╕	201	╔	217	┘	233	Θ	249	·
137	ë	153	Ö	169	_	185	╣	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	170	¬	186	║	203	╦	219	▉	235	δ	251	√
139	ï	156	£	171	½	187	╗	204	╠	220	▄	236	∞	252	_
140	î	157	¥	172	¼	188	╝	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	173	¡	189	╜	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	174	«	190	╛	207	╧	223	▀	239	∩	255	
143	Å	192	└	175	»	191	┐	208	╨	224	α	240	≡		

Operatori di manipolazione dei bit

Con questi operatori si possono eseguire operazioni complesse sui singoli bit delle variabili dei programmi.

Esempio:

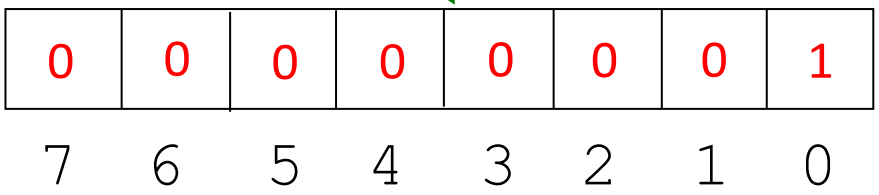
```
unsigned char car = 'm',   val = 1;
.....................
car = car  &     ~     (val <<  5 );
```

AND NOT left shift

Provoca la trasformazione del carattere 'm' (il cui codice ASCII è 109) nel valore intero 77 corrispondente, in codice ASCII, al carattere 'M'.

Operatori di manipolazione dei bit

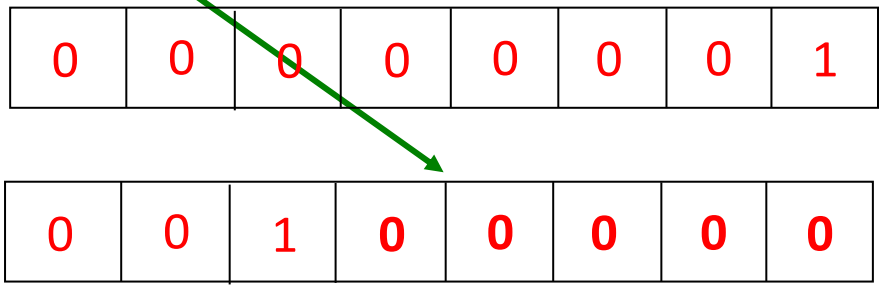
```
car = car  &  ~  (  val  <<  5 )
```



Codice binario del numero 1 su 8 bit (variabile **val**)

Operatori di manipolazione dei bit

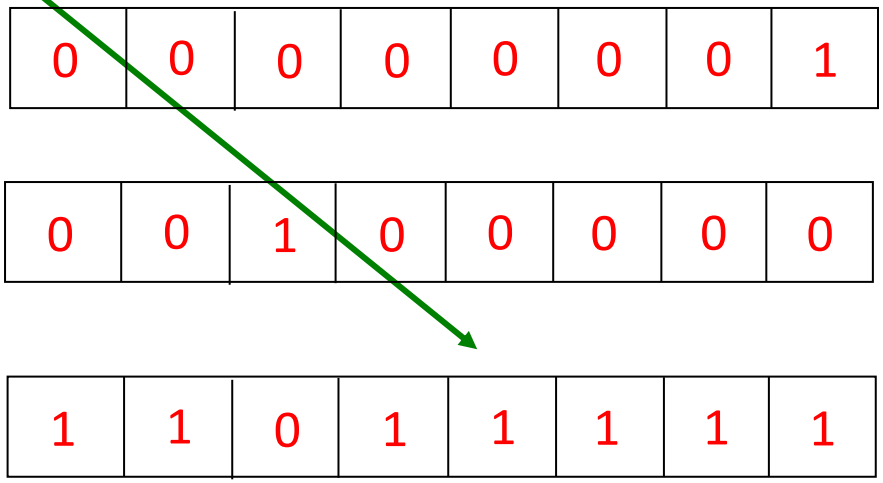
car = car & ~ (**val << 5**)



Operazione di shift a sinistra di 5 posizioni sulla variabile **val** (risultato = 32).

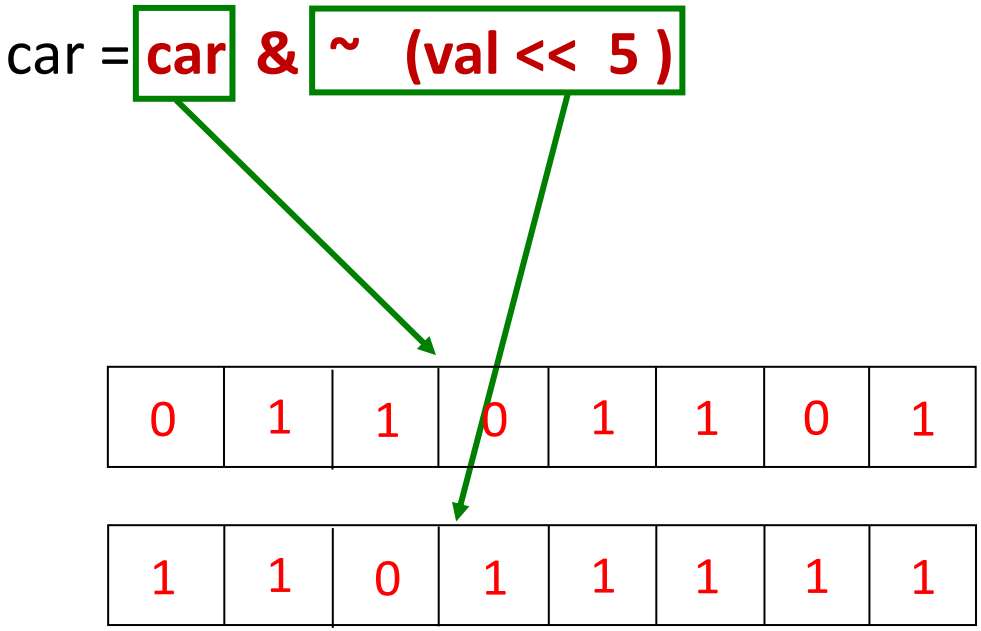
Operatori di manipolazione dei bit

car = car & **~ (val << 5)**



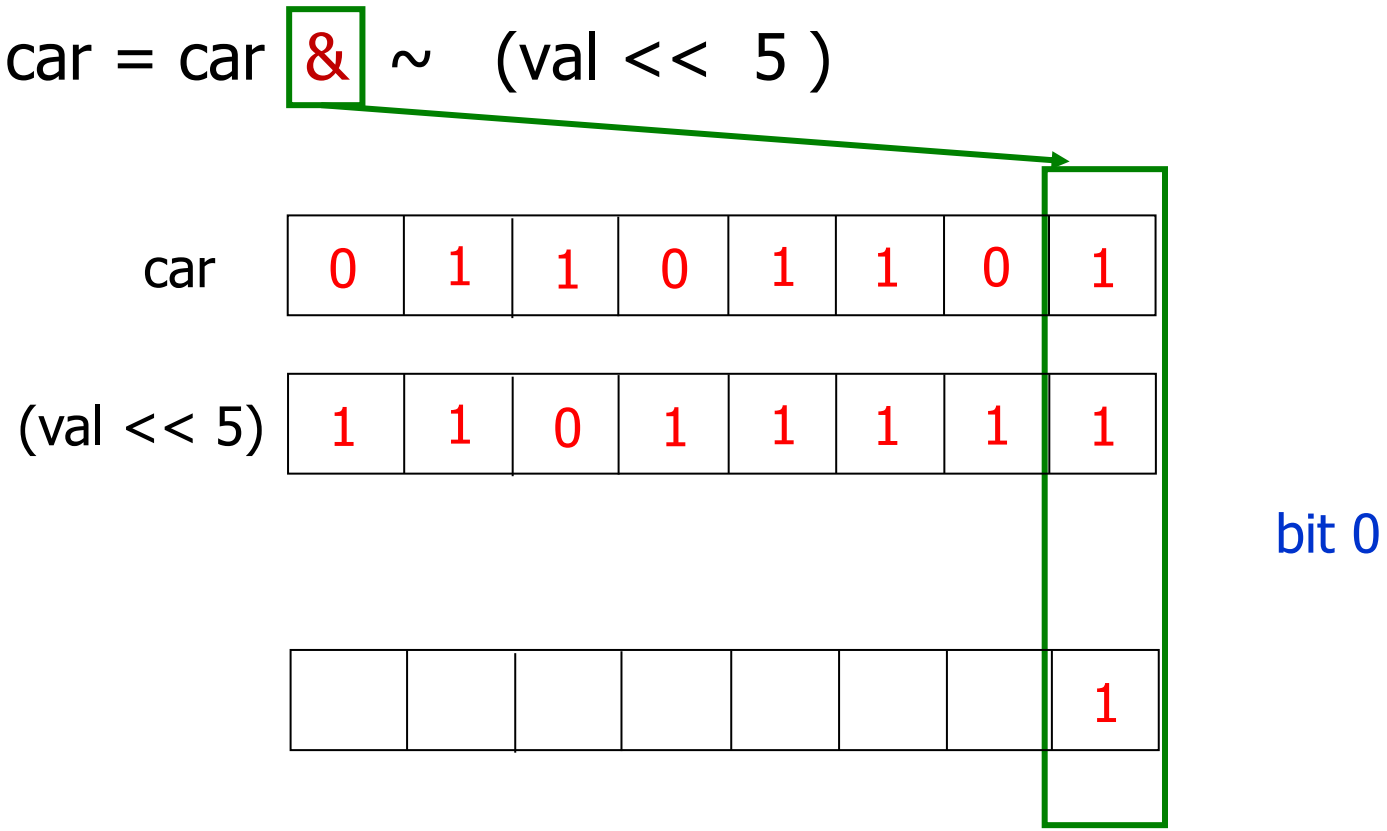
Operazione **NOT** sul risultato del precedente shift (risultato = 223 in binario puro).

Operatori di manipolazione dei bit



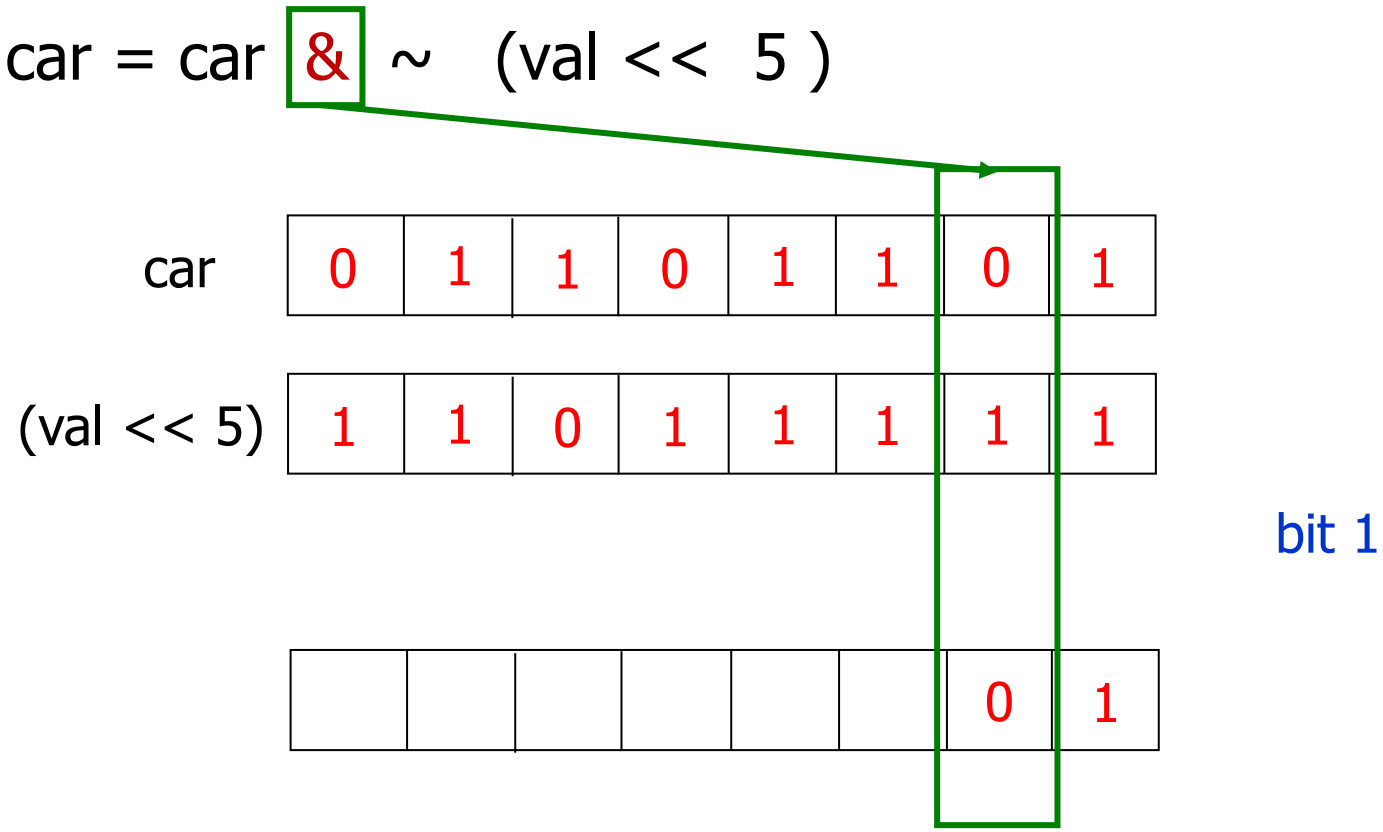
In car vi è il codice di ‘m’, cioè 109.

Operatori di manipolazione dei bit



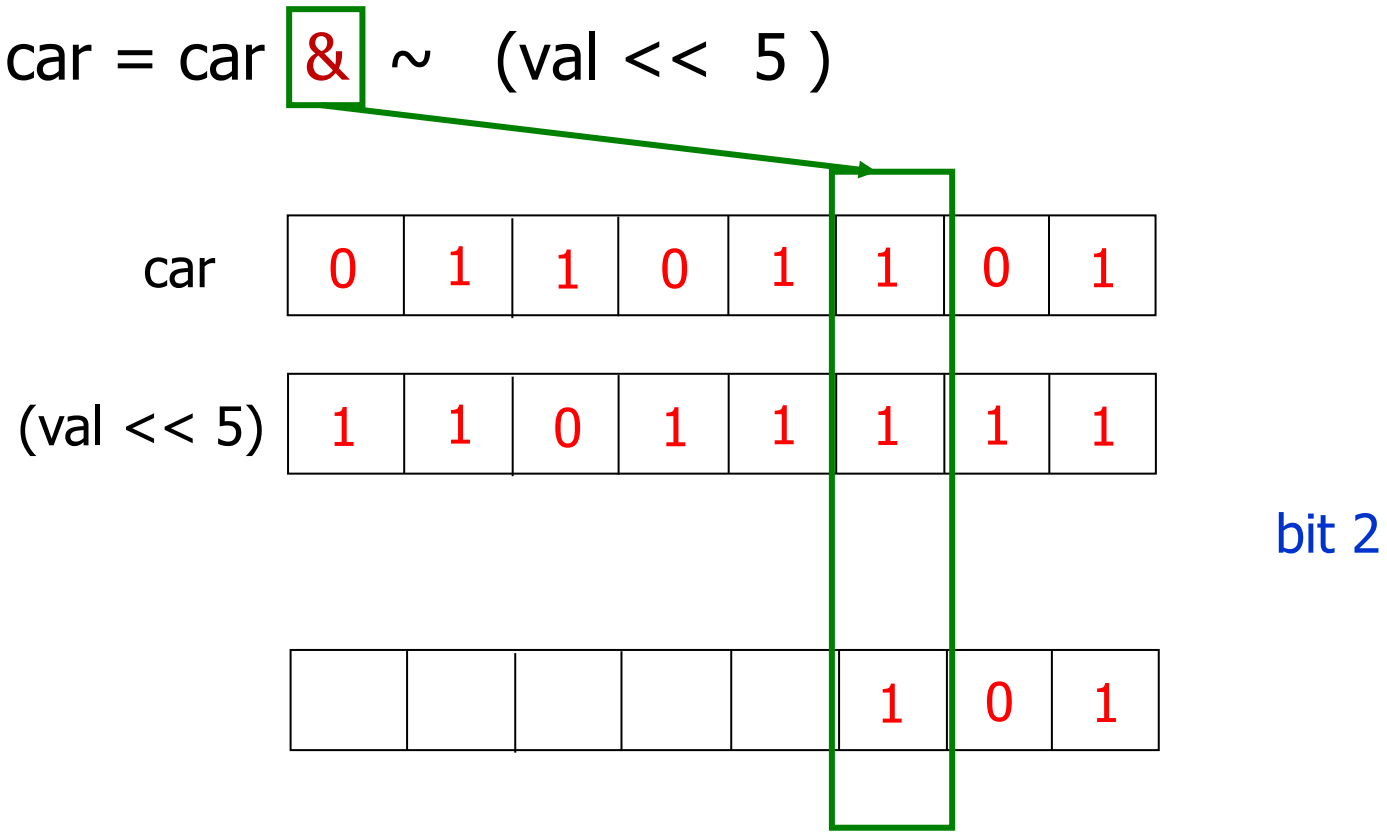
Operatore **AND** (**&**): esegue l'operazione logica **AND** tra i bit corrispondenti delle due variabili interessate.

Operatori di manipolazione dei bit



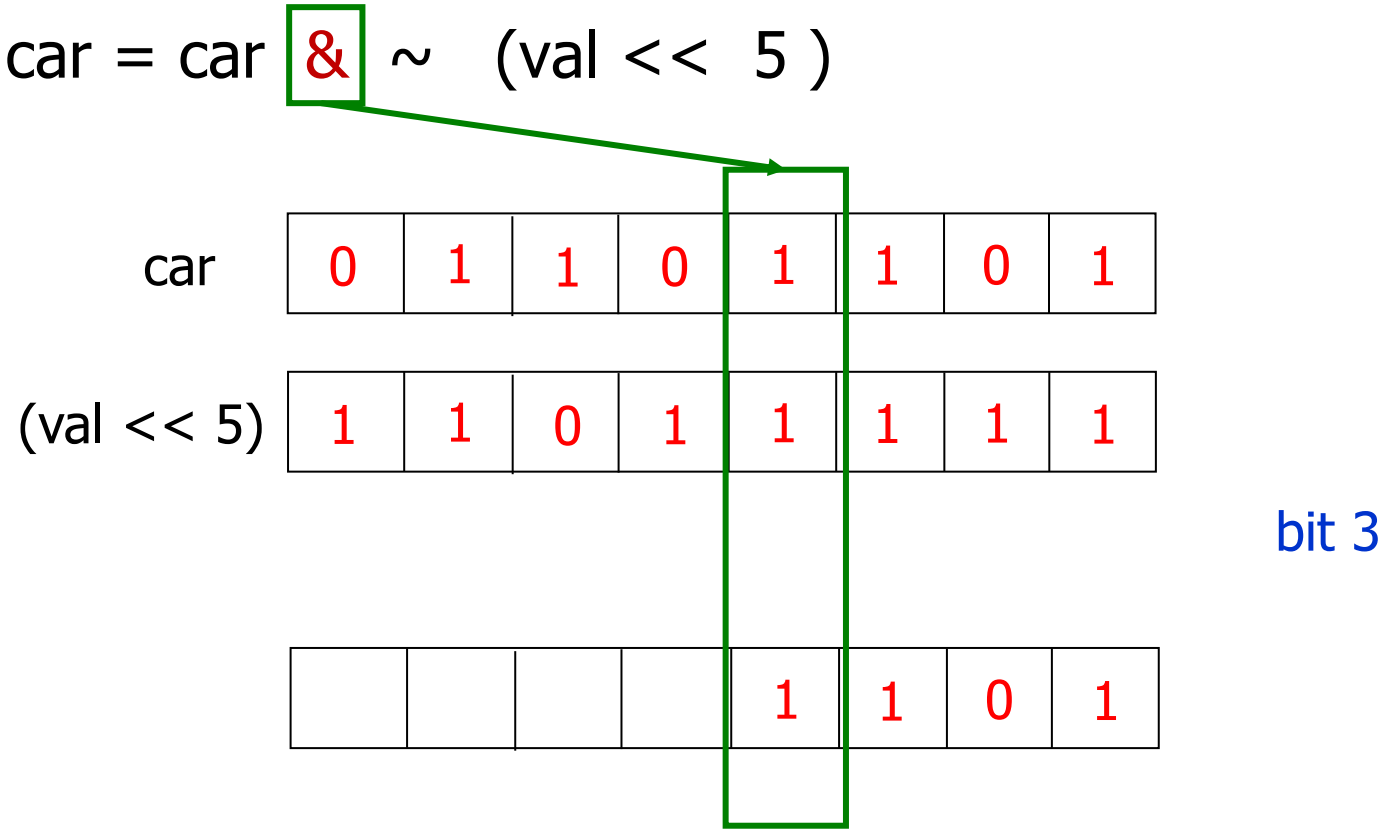
Operatore **AND** (**&**): esegue l'operazione logica **AND** tra i bit corrispondenti delle due variabili interessate.

Operatori di manipolazione dei bit



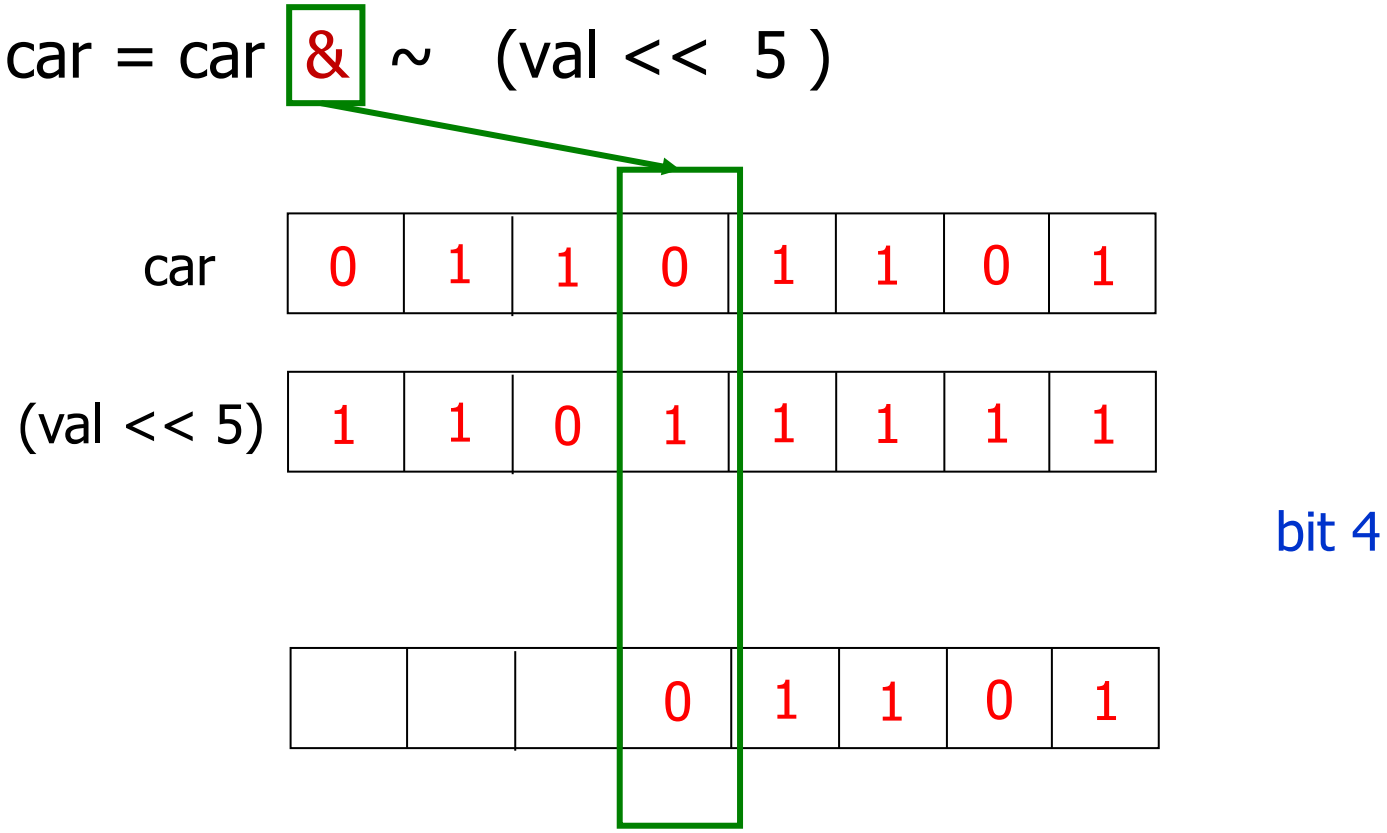
Operatore **AND** (**&**): esegue l'operazione logica **AND** tra i bit corrispondenti delle due variabili interessate.

Operatori di manipolazione dei bit



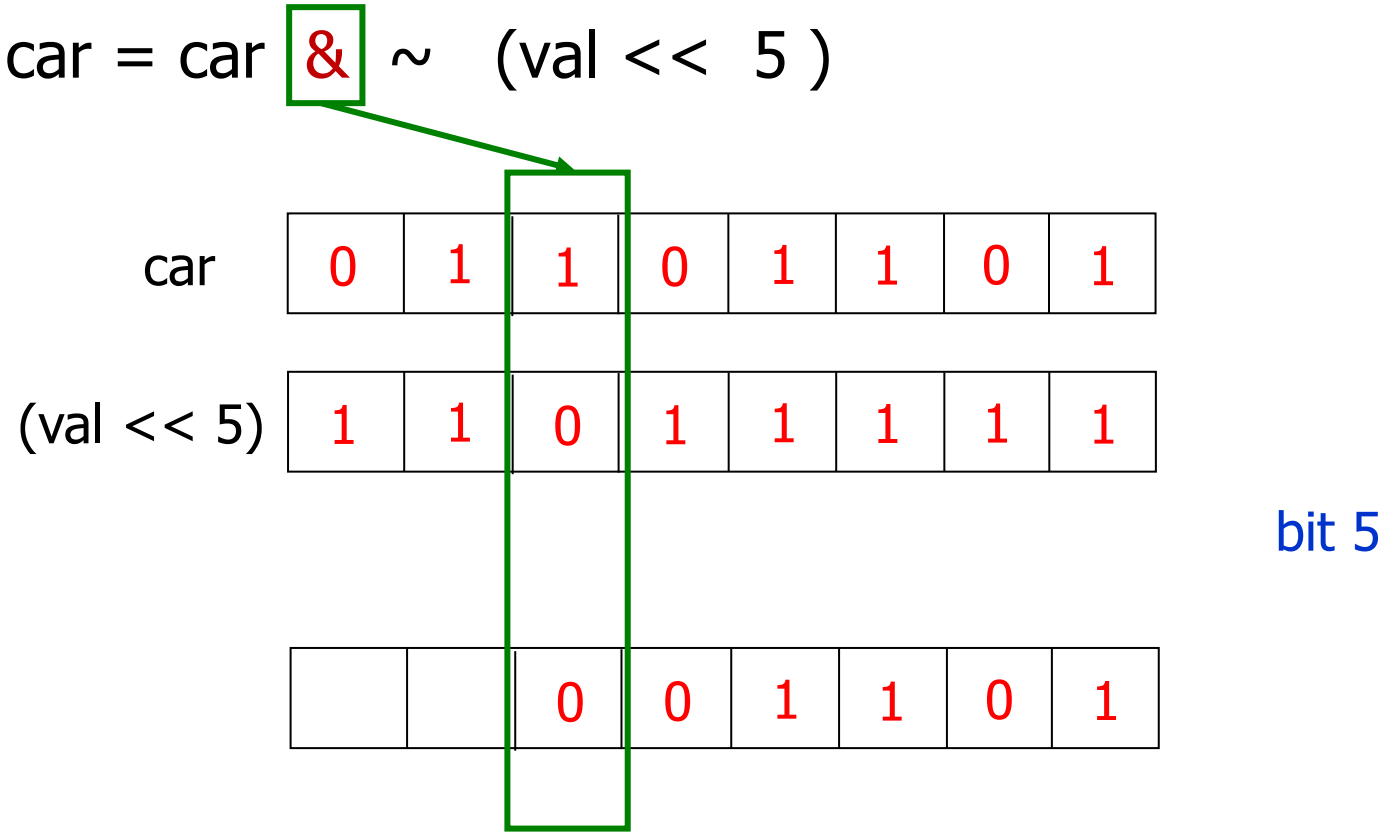
Operatore **AND** (**&**): esegue l'operazione logica **AND** tra i bit corrispondenti delle due variabili interessate.

Operatori di manipolazione dei bit



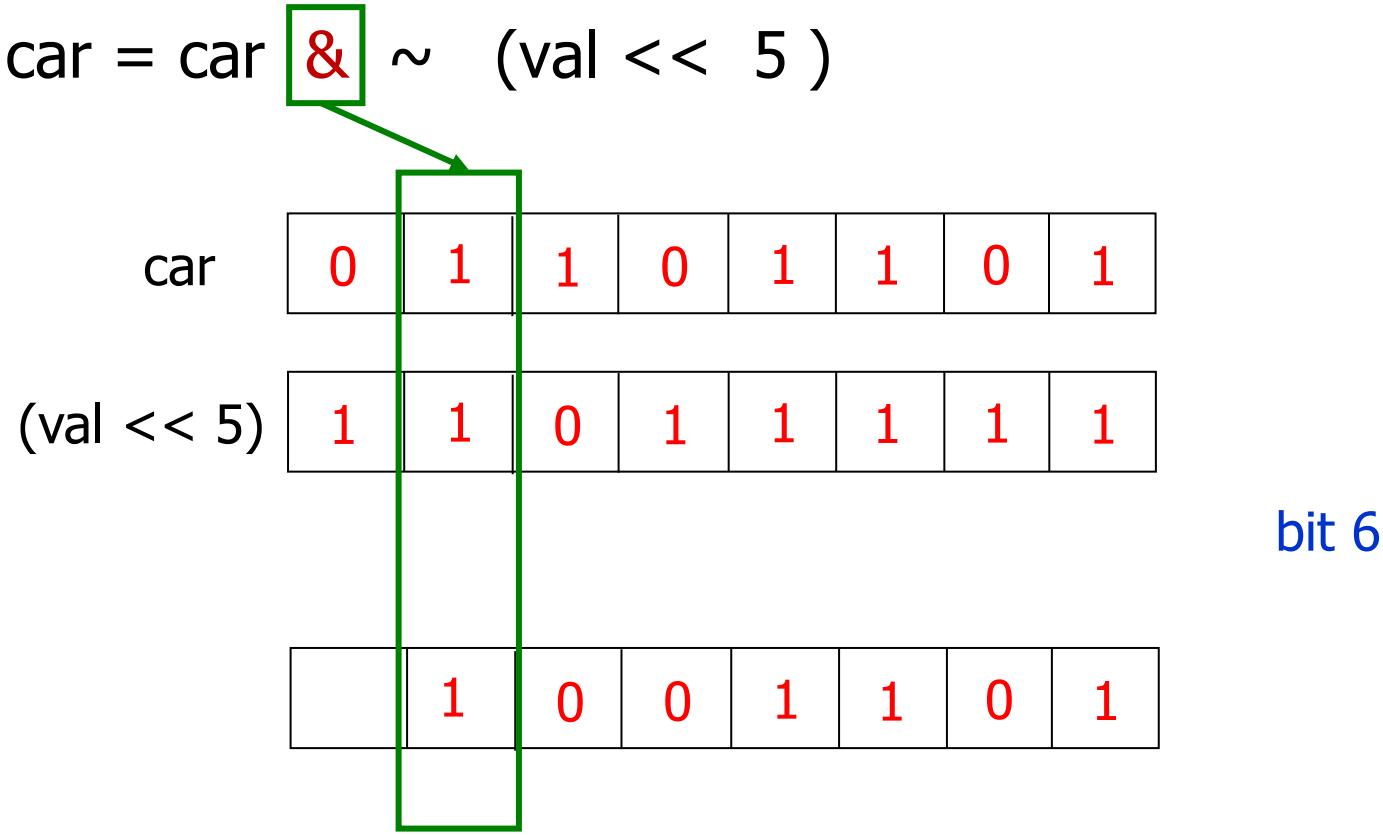
Operatore **AND** (**&**): esegue l'operazione logica **AND** tra i bit corrispondenti delle due variabili interessate.

Operatori di manipolazione dei bit



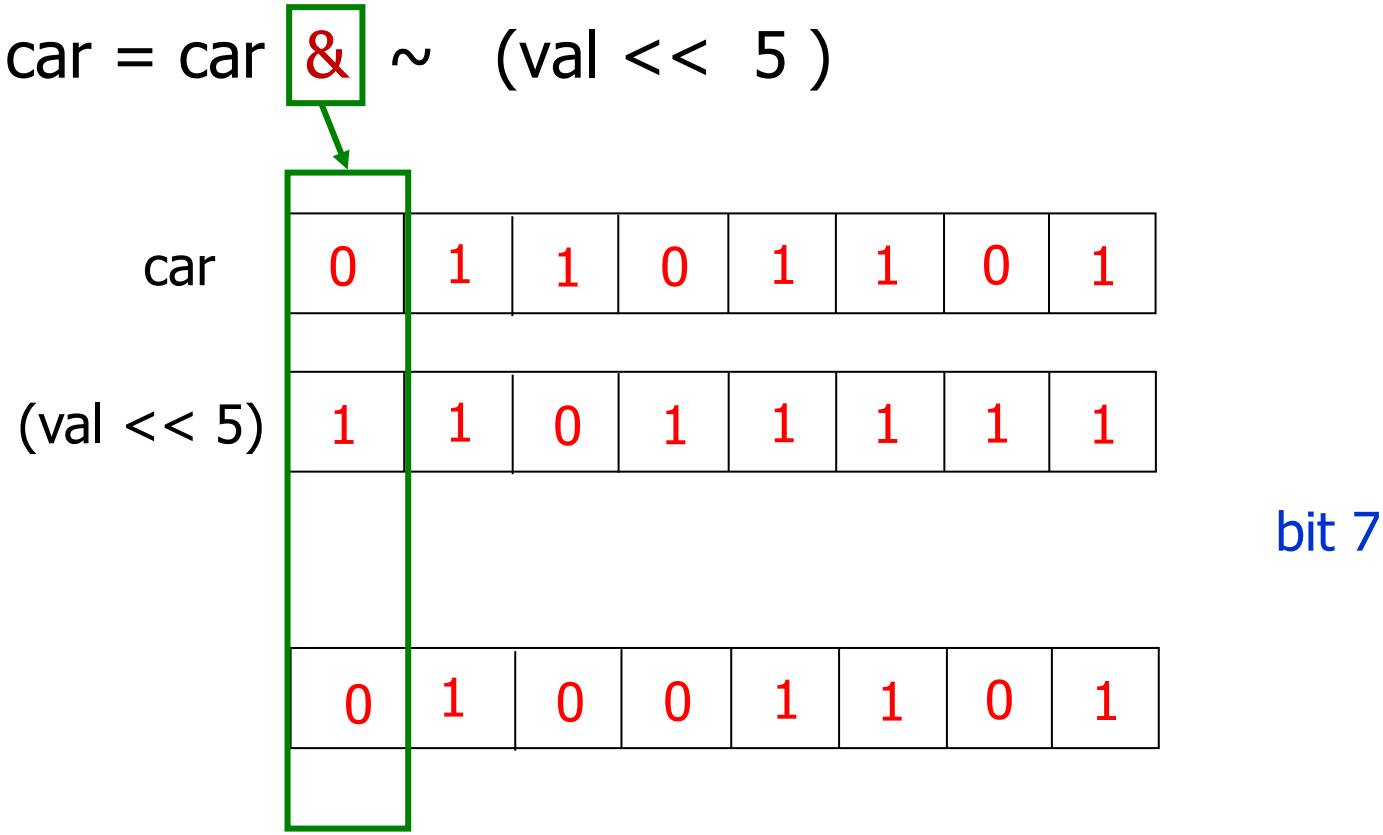
Operatore **AND** (**&**): esegue l'operazione logica **AND** tra i bit corrispondenti delle due variabili interessate.

Operatori di manipolazione dei bit



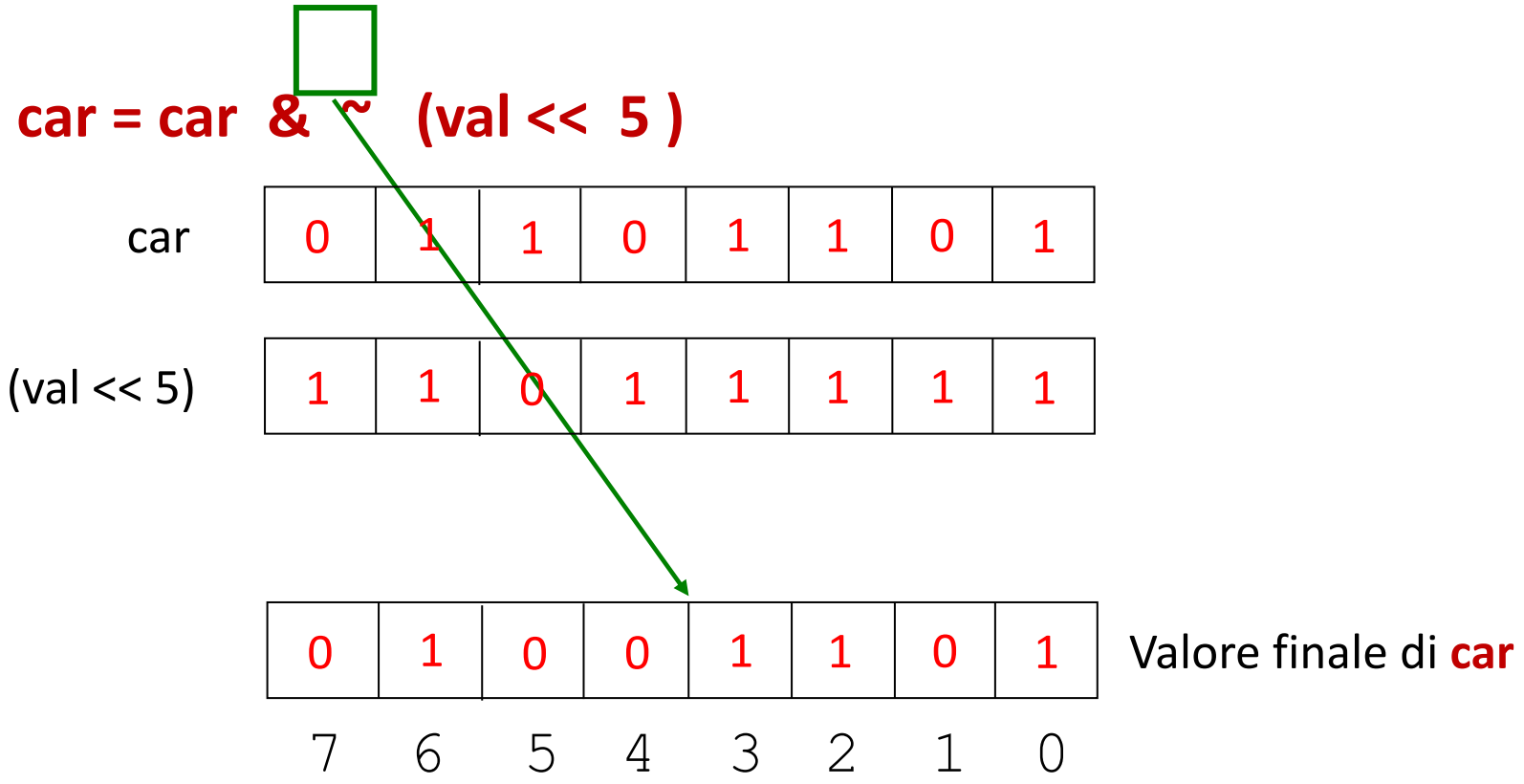
Operatore **AND** (**&**): esegue l'operazione logica **AND** tra i bit corrispondenti delle due variabili interessate.

Operatori di manipolazione dei bit



Operatore **AND** (**&**): esegue l'operazione logica **AND** tra i bit corrispondenti delle due variabili interessate.

Operatori di manipolazione dei bit



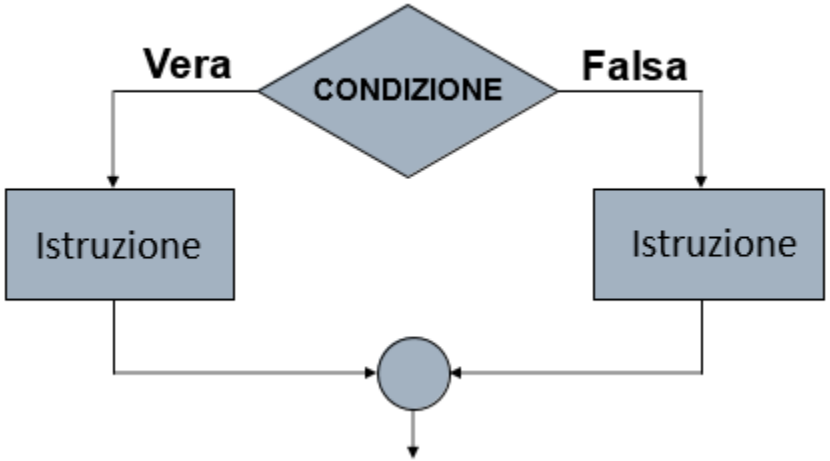
In pratica si è “spento” il bit in posizione 5 di **car**

Le strutture condizionali

Normalmente il flusso di esecuzione di un programma procede in modo sequenziale.

Talvolta l'algoritmo richieda di eseguire in alternativa un blocco di istruzioni piuttosto che un altro, in funzione del verificarsi di qualche condizione.

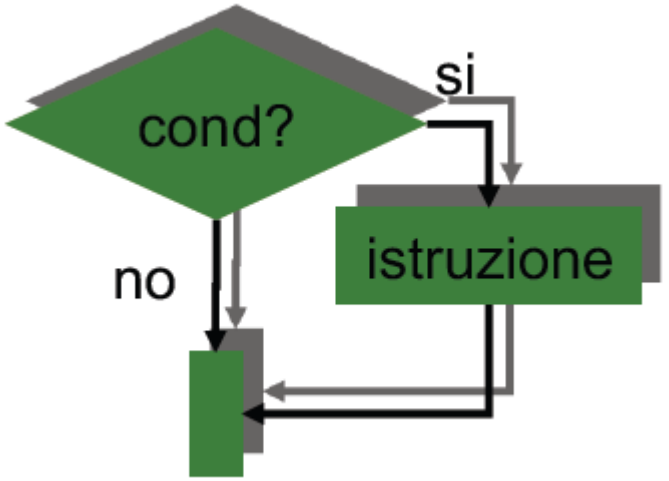
Il linguaggio C dispone di particolari istruzioni, dette **strutture di controllo**, che consentono di eseguire un blocco di istruzioni in modo condizionato, mantenendo la strutturazione e la leggibilità del programma.



Le strutture condizionali - If

L'istruzione di controllo **if** (o selezione) indica al calcolatore di eseguire una tra due istruzioni (semplici o blocchi) al verificarsi di una certa condizione:

if (**condizione**) **istruzione**;



dove

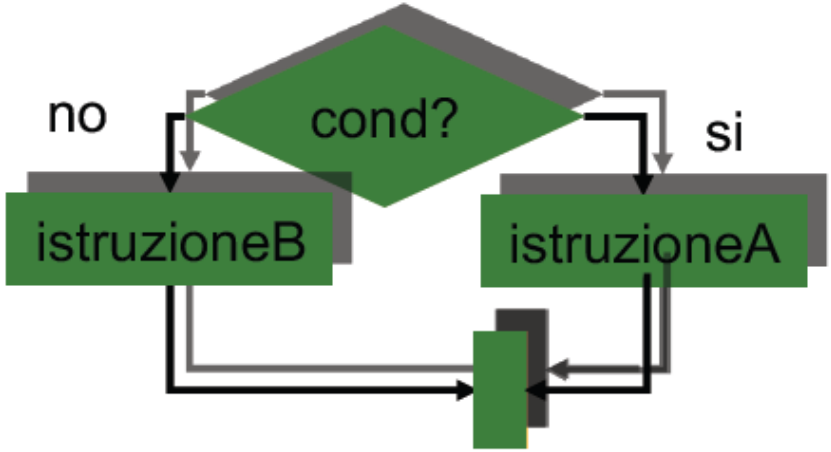
– condizione é una espressione logica

– se la condizione é **true** il programma esegue l'istruzione, altrimenti passa direttamente all'istruzione successiva

Le strutture condizionali - If

Nel caso di due scelte alternative, all'istruzione di controllo **if** si può associare l'istruzione **else**:

```
if (condizione) istruzione_1;
else istruzione_2;
```

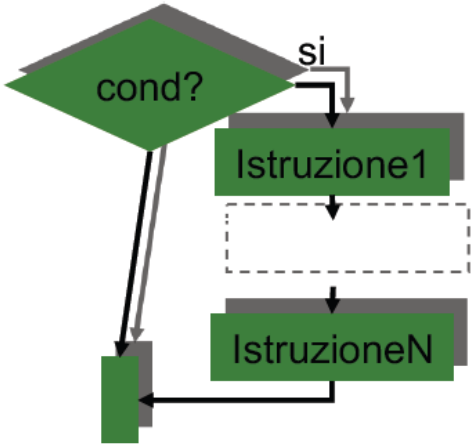


dove

- condizione é una espressione logica
- se la condizione é **true** il programma esegue **istruzione_1**
- altrimenti esegue **istruzione_2**

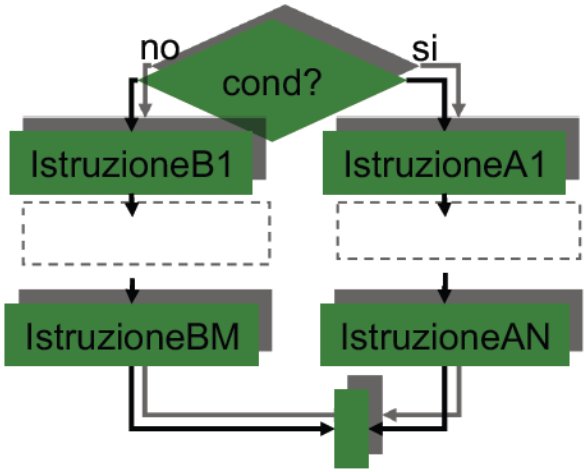
Le strutture condizionali - If

Se la condizione determina l'esecuzione di un **blocco di istruzioni**




```
if (condizione) {
   istruzione_1;
   ...
   istruzione_N;
};
```

Analogamente se determina l'esecuzione di uno di due blocchi di istruzioni




```
if (condizione) {
   istruzione_A1;
   ...
   istruzione_AN;
} else {
   istruzione_B1;
   ...
   istruzione_BN;
};
```

Esempio: equazione di primo grado

Dati due numeri qualsiasi a, b risolvere l'equazione di primo grado:

$$ax + b = 0$$

Algoritmo

leggi(**a**,**b**)
stampa(-**b**/**a**)

È corretto?

Infatti il problema dice che **a** può essere un **numero qualsiasi** e, se **a** fosse zero, si avrebbe un errore a **run time** poiché la divisione non è definita quando il divisore è zero.

Esempio: equazione di primo grado

Algoritmo

```
leggi(a,b)
if (a ≠ 0)
      x ← -b/a
      stampa(x)
else
      stampa(messaggio)
```

Programma

main.c

```c
#include <stdio.h>

int main()
{
  float x,a,b;

  printf("inserire i coefficienti di\nuna equazione di primo grado\n");
  printf("a(!0): ");
  scanf("%f",&a);
  printf("b: ");
  scanf("%f",&b);

  if (a!=0){ // parentesi graffa necessaria perché l'istruzione è composta
    x= -b/a;
    printf("soluzione x = %f\n",x);
  } // fine istruzione composta relative alla condizione (a!=0)
  else
    printf("l'equazione non è di primo grado\n");
}
```

Esecuzione

```
inserire i coefficienti di
una equazione di primo grado
a(!0): 4.5
b: 3.2
soluzione x = -0.711111
```

Esercizio

Algoritmo

leggi(**a**,**b**,**c**)
Massimo ← 0
Se **a** > Massimo
  Massimo ← **a**
Se **b** > Massimo
  Massimo ← **b**
Se **c** > Massimo
  Massimo ← **c**

stampa(Massimo)

Esecuzione

```
Il massimo fra 10, 12 e 1 è il valore 12
```

Programma (Esercizio_004_b.c)

main.c

```c
#include <stdio.h>

int main()
{
  int a;
  int b;
  int c;

  int Massimo = 0;

  // leggiamo i tre valori in input
  printf("inserisci il primo valore: ");
  scanf("%d", &a);
  printf("inserisci il secondo valore: ");
  scanf("%d", &b);
  printf("inserisci il terzo valore: ");
  scanf("%d", &c);

  if(a>Massimo)
    Massimo = a;

  if(b>Massimo)
    Massimo = b;

  if(c>Massimo)
    Massimo = c;

  printf("Il massimo fra %d, %d e %d è il valore %d\n", a, b, c, Massimo);

  return 0;
}
```

Esercizio

Algoritmo

leggi(**a**,**b**,**c**)
Se **a** > **b** e **a** > **c**
  Massimo ← **a**
Se **b** > **a** e **b** > **c**
  Massimo ← **b**
Altrimenti
  Massimo ← **c**

stampa(Massimo)

È corretto?

Algoritmo

leggi(**a**,**b**,**c**)
Se **a**≥**b** e **a**≥**c**
  Massimo ← **a**
Se **b**≥**a** e **b**≥**c**
  Massimo ← **b**
Se **c**≥**a** e **c**≥**b**
  Massimo ← **c**

stampa(Massimo)

Esecuzione

```
inserisci il primo valore: 10
inserisci il secondo valore: 10
inserisci il terzo valore: 1
Il massimo fra 10, 10 e 1 è il valore 1
```

Esercizio

Algoritmo

leggi(**a**,**b**,**c**)
Se **a**≥**b** e **a**≥**c**
  Massimo ← **a**
Se **b**≥**c** e **b**≥**c**
  Massimo ← **b**
 Se **c**≥**a** e **c**≥**b**
  Massimo ← **c**

stampa(Massimo)

Esecuzione

```
inserisci il primo valore: 10
inserisci il secondo valore: 10
inserisci il terzo valore: 1
Il massimo fra 10, 10 e 1 è il valore 10
```

Programma (Esercizio_004_c.c)

main.c

```c
#include <stdio.h>

int main()
{
  int a;
  int b;
  int c;

  int Massimo = 0;

  // leggiamo i tre valori in input
  printf("inserisci il primo valore: ");
  scanf("%d", &a);
  printf("inserisci il secondo valore: ");
  scanf("%d", &b);
  printf("inserisci il terzo valore: ");
  scanf("%d", &c);

  if(a>=b && a>=c)
    Massimo = a;

  if(b>=a && b>=c)
    Massimo = b;

  if(c>=a && c>=b)
    Massimo = c;

  printf("Il massimo fra %d, %d e %d è il valore %d\n", a, b, c, Massimo);

  return 0;
}
```

Le strutture condizionali - If

Errori tipici di programmazione:

```
if (N=1) {
...
};
```

Viene eseguito sempre

```
if (N==1) {
...
};
```

```
if (N==0);
{
...
};
```

Viene eseguito sempre

```
if (N==0)
  istruzione_1;
```

```
if (cond)
  istruzione_1;
  istruzione_2;
else
  istruzione_3;
```

Errore di compilazione:
Expected “;” before “else”

```
if (cond){
  istruzione_1;
  istruzione_2;
} else
  istruzione_3;
```

Le strutture condizionali – If/Else

Stampare il voto degli Esercizi

- Se il voto dello studente è ≥ 28
  Visualizza A
- Se il voto dello studente è ≥ 25 e < 28
  Visualizza B
- Se il voto dello studente è ≥ 22 e < 25
  Visualizza C
- Se il voto dello studente è ≥ 18 e <22
  Visualizza D
- Se il voto dello studente è < 18
  Visualizza non superato

Stampare il voto degli Esercizi

```
if (voto >= 28)
   printf("A");
else{                          //apro livello 1
    if (voto >= 25  && voto < 28)
L1     printf("B");
     else{                     //apro livello 2
       if (voto >= 22 && voto < 25)
L2       printf("C");
       else{                   //apro livello 3
         if(voto >= 18 && voto < 22)
L3         printf("D");
         else{                 //apro livello 4
           if(voto < 18)
L4           printf("non superato");
           }                   // chiudo livello 4
       }                       // chiudo livello 3
     }                         // chiudo livello 2
}                              // chiudo livello 1
```

Le strutture condizionali – If/Else

Stampare il voto degli Esercizi

- Se il voto dello studente è ≥ 28
  Visualizza A
- Se il voto dello studente è ≥ 25 e < 28
  Visualizza B
- Se il voto dello studente è ≥ 22 e < 25
  Visualizza C
- Se il voto dello studente è ≥ 18 e <22
  Visualizza D
- Se il voto dello studente è < 18
  Visualizza non superato

Stampare il voto degli Esercizi

```
if (voto >= 28)
   printf("A");
else{                          //apro livello 1
L1   if (voto >= 25)
        printf("B");
     else{                     //apro livello 2
        if (voto >= 22)
L2         printf("C");
        else{                  //apro livello 3
           if(voto >= 18)
L3            printf("D");
           else
              printf("non superato");
        }                      // chiudo livello 3
     }                         // chiudo livello 2
}                              // chiudo livello 1
```

Le strutture condizionali – If/Else

Stampare il voto degli Esercizi

- Se il voto dello studente è ≥ 28
  Visualizza A
- Se il voto dello studente è ≥ 25 e < 28
  Visualizza B
- Se il voto dello studente è ≥ 22 e < 25
  Visualizza C
- Se il voto dello studente è ≥ 18 e <22
  Visualizza D
- Se il voto dello studente è < 18
  Visualizza non superato

Stampare il voto degli Esercizi

```
if (voto >= 28)
   printf("A");
else if (voto >= 25)
        printf("B");
     else if (voto >= 22)
             printf("C");
          else if(voto >= 18)
                  printf("D");
          else
                  printf("non superato");
```

Le strutture condizionali – If/Else

```
if(x > 5)
  if(y > 5)
    printf("x e y sono > 5");
  else
    printf("x è minore di 5");
```

Cosa stampa se x=6 e y=7?

Esecuzione

```
x e y sono > 5
```

Programma

main.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int x=6;
6    int y=7;
7
8    if (x > 5)
9      if (y > 5)
10       printf("x e y sono > 5");
11     else
12       printf("x è minore di 5");
13
14   return 0;
15 }
```

Cosa stampa se x=5 e y=7?

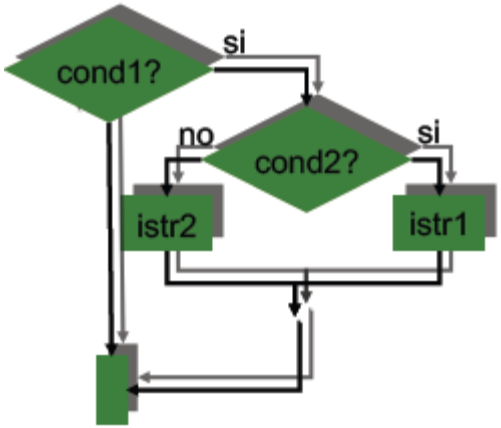
Esecuzione

Le strutture condizionali – If/Else

Se l'istruzione controllata da un **if** consiste a sua volta in un altro **if** (istruzioni **if** nidificate), ogni eventuale **else** si riferisce sempre all'**if** immediatamente superiore (in assenza di parentesi graffe).

Esempio:

```
If (cond1)
   if (cond2) istr1;
   else istr2;
```



Suggerimento:

– Mettere sempre le parentesi anche in caso di istruzioni semplici

1) Consente di controllar meglio il risultato di annidamenti di **if**

2) Riduce gli errori in caso di aggiunte di altre istruzioni come caso dell'**if** (o dell'**else**)

Le strutture condizionali – If/Else

IF o IF…… ELSE?

```
if (a<b)
  printf(”MASSIMO=%d\n”,b);
else
  printf(”MASSIMO=%d\n,a);
```

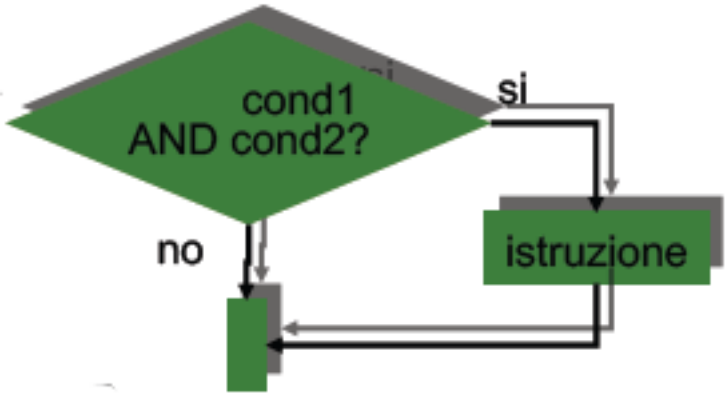
Il caso **a=b** fa stampare **a** come massimo

```
if (a<b)
  printf(”MASSIMO=%d\n”,b);
else if (b<a)
  printf(”MASSIMO=%d\n,a);
```

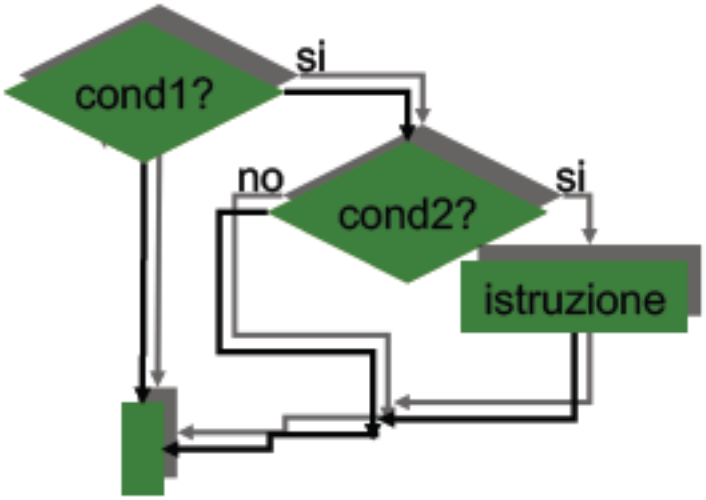
Il caso **a=b** non produce alcuna stampa

Le strutture condizionali – If

Se un blocco di istruzioni è controllato da un **if** con una congiunzione di due (o più) condizioni (**AND** logico)



… lo stesso blocco di istruzioni, in modo equivalente, può essere controllato da due (o più) **if** annidati ciascuno che verifica una delle condizioni



In generale:

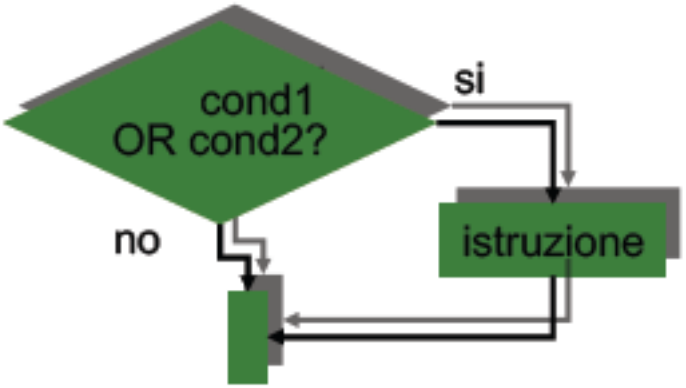
```
if (cond1 && cond2 && ... && condN)
  istr1;
```

equivale a:

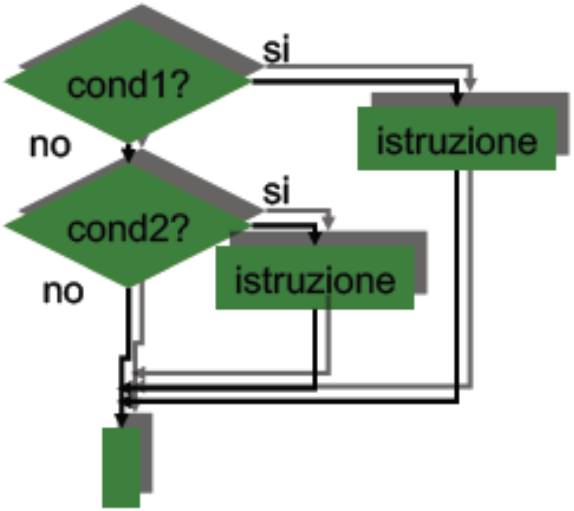
```
if (cond1)
  if (cond2)
    ...
      if (condN)
        istruzione;
```

Le strutture condizionali – If

Se un blocco di istruzioni è controllato da un **if** con una disgiunzione di due (o più) condizioni (**OR** logico)



… lo stesso blocco di istruzioni, in modo equivalente, può essere controllato da due (o più) **if** annidati nel ramo **else** ciascuno che verifica una delle condizioni



In generale:

```
if (cond1 || cond2 || ... || condN)
  istr1;
```

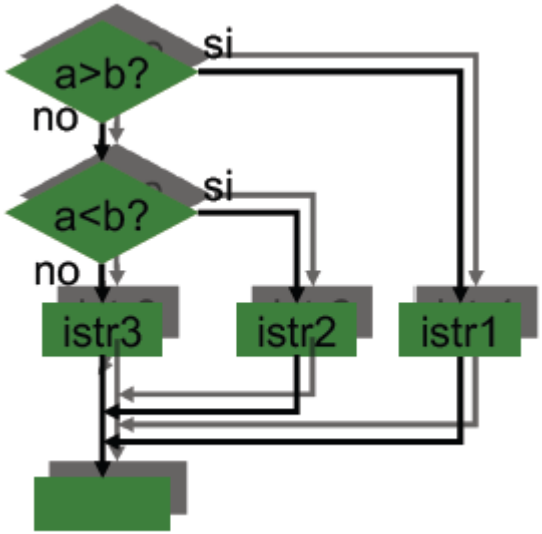
equivale a:

```
if (cond1) istruzione;
  else if (cond2) istruzione;
    ...
      else if (condN)
        istruzione;
```

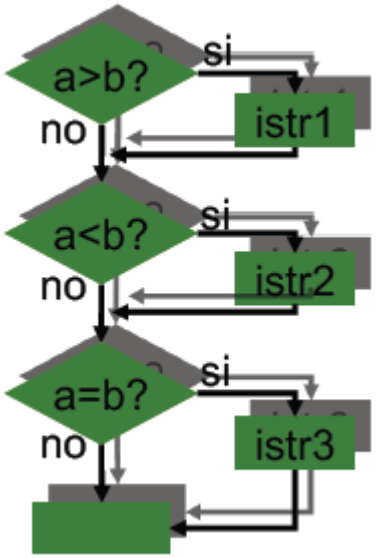
Le strutture condizionali – If

Esempio:

```
if (a>b)
  istr1;
else
  if(a<b)
    istr2;
  else
    istr3;
```



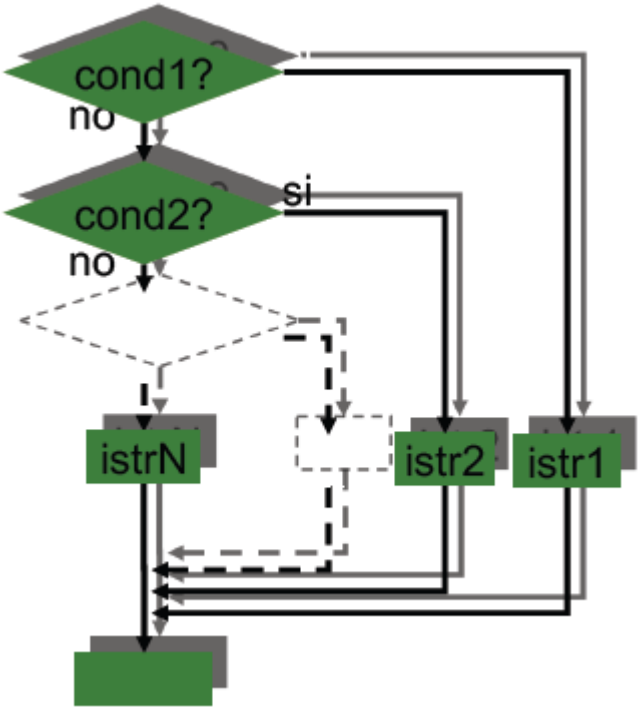

```
if (a>b)
  istr1;
if(a<b)
  istr2;
if(a==b)
  istr3;
```



Le strutture condizionali – If

Se bisogna effettuare **n** test distinti ad ognuno dei quali corrisponde una azione alternativa occorre adoperare **n-1** **else** annidati secondo il seguente schema:

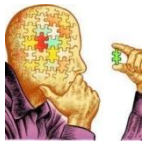
```
if (cond1)
   istr1;
else if (cond2)
   istr2;
else if (...)
   ...
else
   istrN;
```



Le strutture condizionali – If

La nidificazione dell'istruzione **if** è spesso fonte di errori

Esempio:
assegnati tre numeri **n1**, **n2**, e **n3** riscriverli in modo ordinato



Ragionamento

Se due numeri **n1** e **n2** non sono nell'ordine richiesto, è sufficiente scambiarli di posizione

Idea

Confronto/Scambio **n1** e **n2**, successivamente
Confronto/Scambio **n1** e **n3**, successivamente
Confronto/Scambio **n2** e **n3**

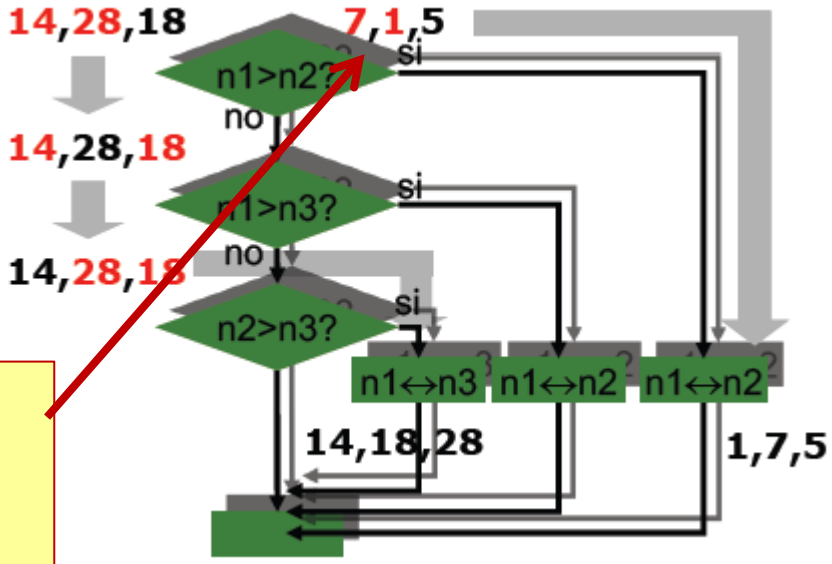
Le strutture condizionali – If

La nidificazione dell'istruzione **if** è spesso fonte di errori

Algoritmo

```
leggi(n1,n2,n3)
if(n1>n2)
  scambia(n1,n2)
else if(n1>n3)
       scambia(n1,n3)
     else if(n2>n3)
            scambia(n2,n3)
stampa(n1,n2,n3)
```

È corretto?



Non funziona perché effettua solo uno scambio, ma in genere può esserne necessario più di uno

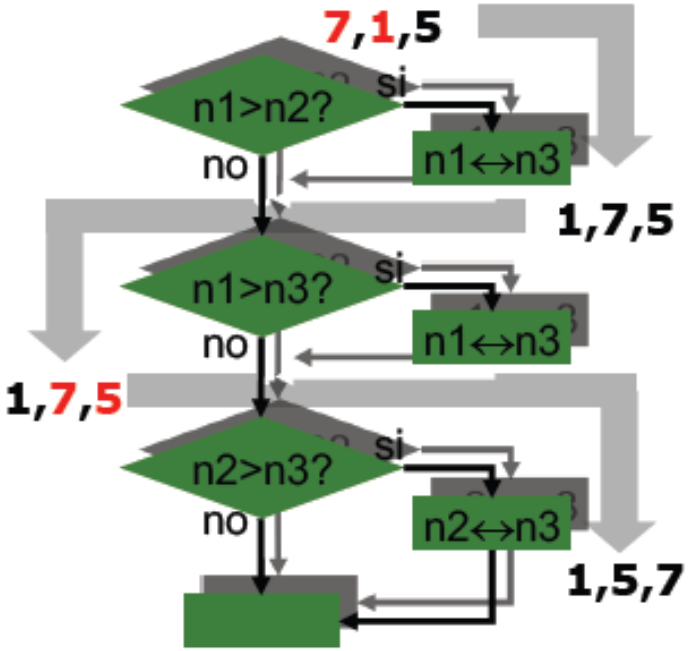
Le strutture condizionali – If

La nidificazione dell'istruzione **if** è spesso fonte di errori

Algoritmo

```
leggi(n1,n2,n3)
if(n1>n2)
   scambia(n1,n2)
if(n1>n3)
   scambia(n1,n3)
if(n2>n3)
   scambia(n2,n3)
stampa(n1,n2,n3)
```

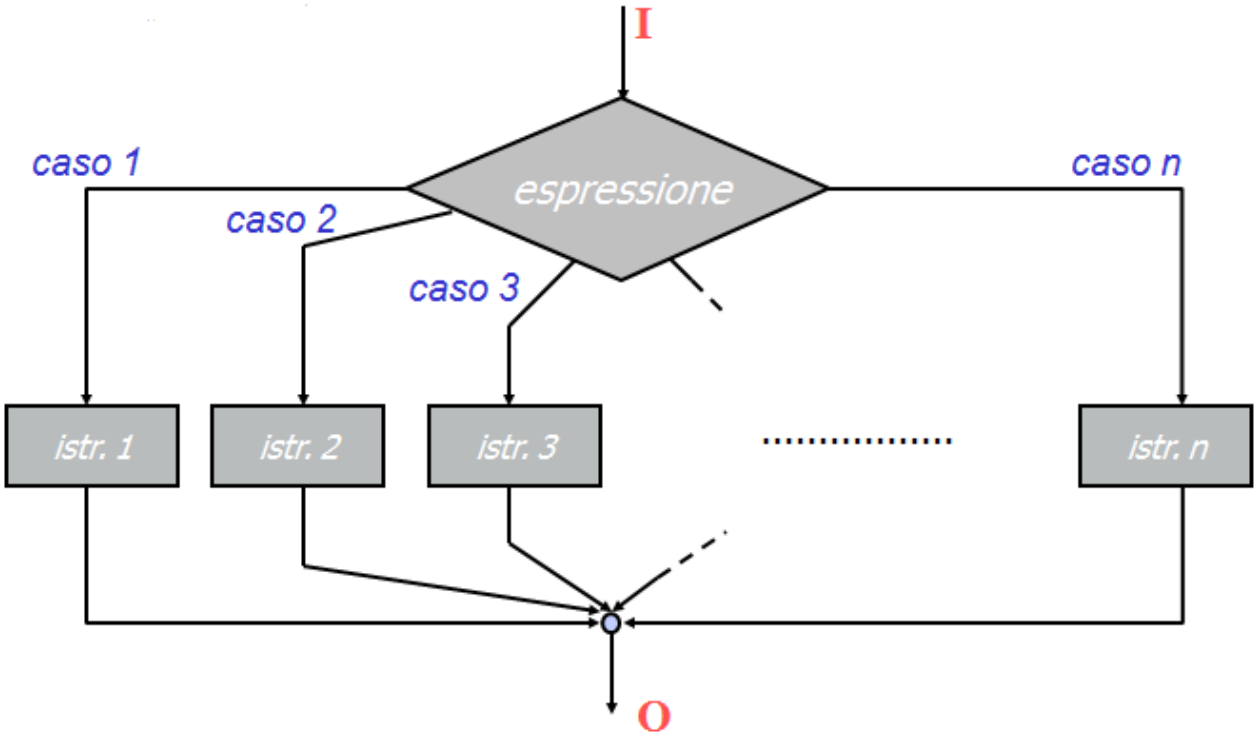
È corretto?



Le strutture condizionali – switch

Quando la condizione da imporre non è intrinsecamente binaria ma si deve distinguere tra più casi ed eseguire le istruzioni appropriate per ogni singolo caso, l'**if annidato** può non essere conveniente per complessità e leggibilità.

Per questi casi c'è l'istruzione **switch**



Le strutture condizionali – switch

Sintassi:

```
switch (espressione)
      {
     case <costante 1>:
                    <istruzione 1>
           break;
     case <costante 2>:
                    <istruzione 2>
           break;
        ...
        ...
     [ default:
                  < istruzione default> ]
     }
```

espressione: espressione a valore numerico di tipo **int** o **char**, ma non **float** o **double**

<**costante1**>, <**costante1**>, ... sono costanti dello stesso tipo dell'espressione

<**istruzione** 1>, <**istruzione** 2>, ... sono sequenze di istruzioni (senza graffe!)

Significato:

In base al valore di **espressione**, esegui le istruzioni del **case** corrispondente. Nel caso nessun **case** venga intercettato, esegui le istruzioni corrispondenti al caso **default** (se esiste).

Le strutture condizionali – switch

I vari **case** devono rappresentare condizioni

MUTUAMENTE ESCLUSIVE!

I vari **case** vengono eseguiti in sequenza, uno dopo l'altro: per evitarlo è necessario interrompere il flusso usando l'istruzione **break** alla fine di ogni blocco di istruzioni (il flusso prosegue con la prima istruzione successiva alla **switch**).

Se il blocco di **default** non è presente e nessun **case** viene soddisfatto, l'esecuzione procede con la prima istruzione che segue la **switch**.

Le strutture condizionali – switch

Esempio:

```
main()
{
  int x;
  printf (“\nIntroduci un numero intero: “);
  scanf (“%d”, &x);

  switch (x) {
        case 2:
              printf(“\nHai introdotto 2“);
              break;
        case 1:
              printf(“\nHai introdotto 1“);
              break;
        default:
              printf(“\nHai introdotto un numero diverso da 1 e 2“);
  }
}
```

I **case** possono essere indicati nell’ordine che si desidera e non in ordine strettamente crescente o decrescente.

Esercizi

1. Determinare se un numero è intero.

Suggerimenti:

- Per risolvere il problema pensiamo di confrontare il numero letto con la sua parte intera: se i due valori sono uguali allora il numero è intero, altrimenti non lo è
- La parte intera si calcola con floor(n) (si deve includere anche il file <math.h>)

2. Determinare se un numero intero è pari.

Suggerimenti:

- Per risolvere il problema pensiamo di calcolare il resto modulo 2; se tale resto è zero allora il numero è pari altrimenti non lo è

3. Scrivere un programma

bitn che prenda in input un valore intero **N** e un intero **i** e stampi a video il valore dell'**i**-esimo bit di **N**