Programmazione 2
e Laboratorio di Programmazione

Corso di Laurea in

Informatica

Università degli Studi di Napoli "Parthenope"

Anno Accademico 2023-2024

Prof. Luigi Catuogno

1

Informazioni sul corso

Docente	Luigi Catuogno **luigi.catuogno@uniparthenope.it**
Orario	Lun: 9:00-11:00 Mer: 11:00-13:00
Sede	Centro Direzionale Napoli **Aula Magna**
Ricevimento	Mer: 14:00-16:00 (previo appuntamento) Ufficio docente oppure Team: **cxxa3bo**

2

Libri di testo

Introduzione al linguaggio – costrutti e tecniche di base

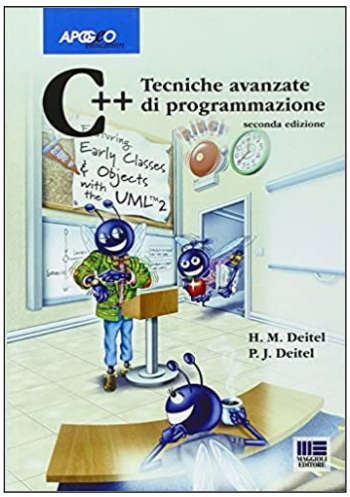
[FdP] H. M. Deitel, P. J. Deitel
C++ Fondamenti di programmazione
II ed. (2014) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8571-9

3

Libri di testo

Tecniche avanzate e strutture dati elementari

[TAP] H. M. Deitel, P. J. Deitel
C++ Tecniche avanzate di programmazione
II ed. (2011) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8572-6



4

Risorse on-line

Team del corso

Programmazione 2 AA 2023-24 - Prof. Catuogno
Comunicazioni, incontri e avvisi per il corso
Codice: **ftomzjx**

Piattaforma e-learning

Programmazione II e Laboratorio di Programmazione II - A.A. 2023-24
Materiale didattico, manualistica, esercitazioni.
URL: https://elearning.uniparthenope.it/course/view.php?id=2386

5

Dal C al C++

6

Operatori bitwise

7

Operatori *«bitwise»*

- Operano sui singoli bit della rappresentazione binaria degli operandi

Op.	descrizione
`&`	`and`
`|`	`or`
`^`	`or esclusivo (xor)`
`~`	`inversione (compl. a uno)`
`<<`	`scorrimento (shift) a sx`
`>>`	`scorrimento a dx`

8

Operatori «bitwise»: esempio &

```
unsigned char x=16,y=24,z;
```

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
x=	0	0	0	1	0	0	0	0

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
y=	0	0	0	1	1	0	0	0

z=x &y=

9

Operatori «bitwise»: esempio &

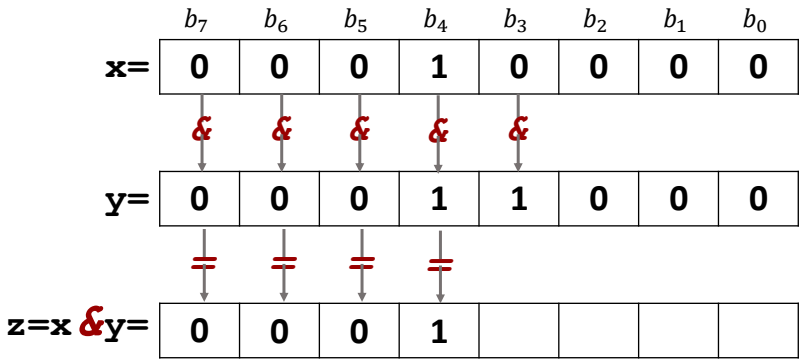
```
unsigned char x=16,y=24,z;
```

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
x=	0	0	0	1	0	0	0	0
	&	&						
y=	0	0	0	1	1	0	0	0
	=							
z=x &y=	0							

10

Operatori «bitwise»: esempio &

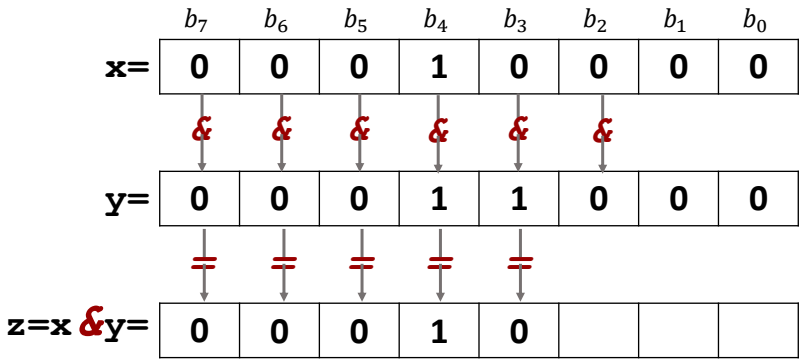
```
unsigned char x=16,y=24,z;
```



11

Operatori «bitwise»: esempio &

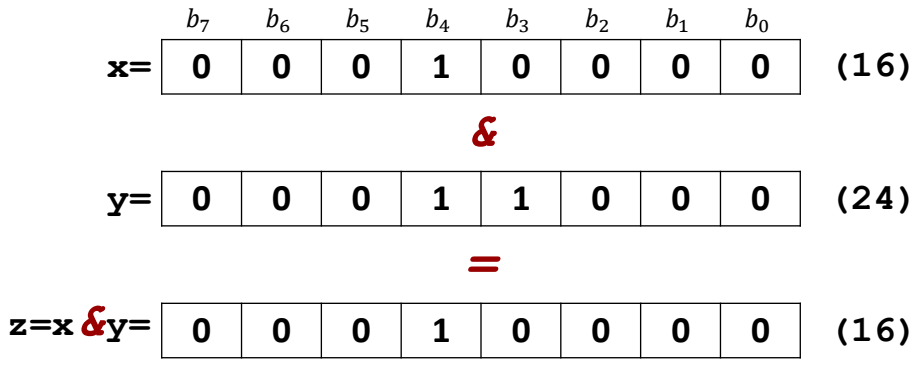
```
unsigned char x=16,y=24,z;
```



12

Operatori «bitwise»: esempio &

```
unsigned char x=16,y=24,z;
```



13

Operatori «bitwise»: esempio & *versus* &&

```
unsigned char x=16,y=24,z;
```

x=16

y=24

z=x&y *vale* **16**

z=x&&y *vale*?

14

Operatori «bitwise»: esempio & *versus* &&

```
unsigned char x=16,y=24,z;
```

x *vale* **16** $x \neq 0 \Rightarrow x$ è ***true***

y *vale* **24** $y \neq 0 \Rightarrow y$ è ***true***

z=x&y *vale* **16**

z=x&&y *vale* **true (1)**

15

Operatori «bitwise»: esempio |

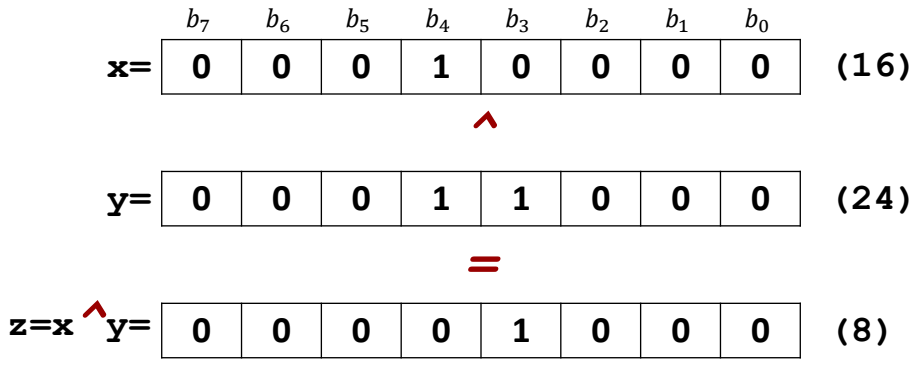
```
unsigned char x=16,y=24,z;
```

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
x=	0	0	0	1	0	0	0	0	(16)
				|					
y=	0	0	0	1	1	0	0	0	(24)
				=					
z=x|y=	0	0	0	1	1	0	0	0	(24)

16

Operatori «bitwise»: esempio ^

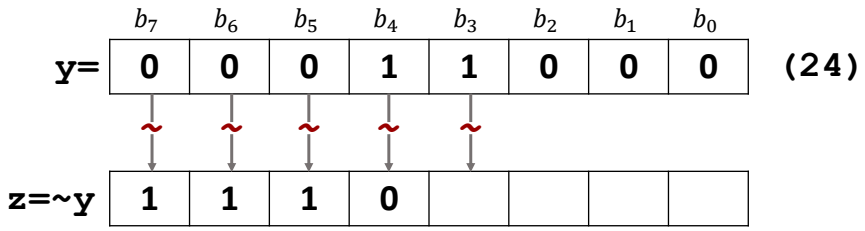
```
unsigned char x=16,y=24,z;
```



17

Operatori «bitwise»: esempio ~

```
unsigned char y=24,z;
```



18

Operatori «bitwise»: esempio ~

```
unsigned char y=24,z;
```

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
y=	0	0	0	1	1	0	0	0	(24)
				~					
z=~y	1	1	1	0	0	1	1	1	(231)

19

Operatori «bitwise»: esempio ~ *versus* !

```
unsigned char y=24,z;
```

y *vale* **24** $y \neq 0 \Rightarrow y$ è ***true***

z= ~ y *vale* **231**

z= ! y *vale ?*

20

Operatori «bitwise»: esempio ~ *versus* !

```
unsigned char y=24,z;
```

y *vale* **24** $y \neq 0 \Rightarrow y$ è ***true***

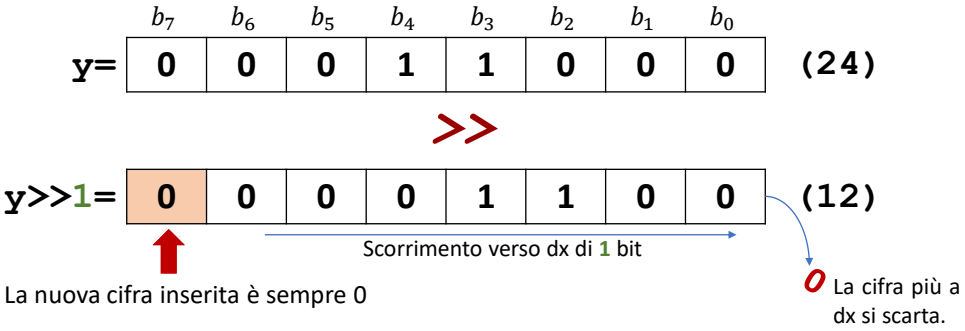
z= ~ y *vale* **231** $z \neq 0 \Rightarrow z$ è ***true***

z= ! y *vale* **false (0)**

21

Operatori «bitwise»: esempio >>

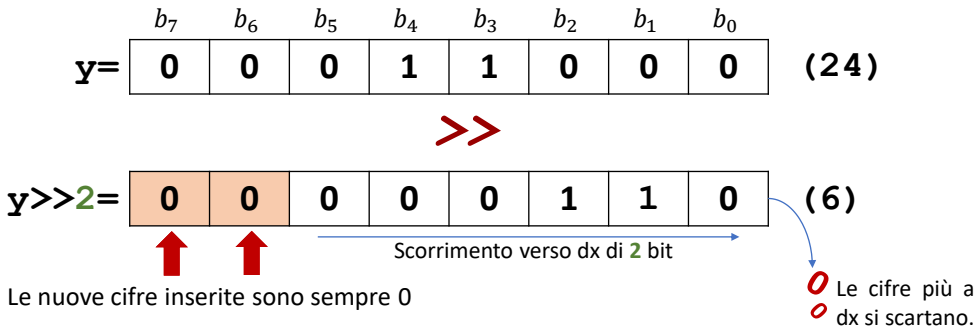
```
unsigned char y=24;
```



22

Operatori «bitwise»: esempio >>

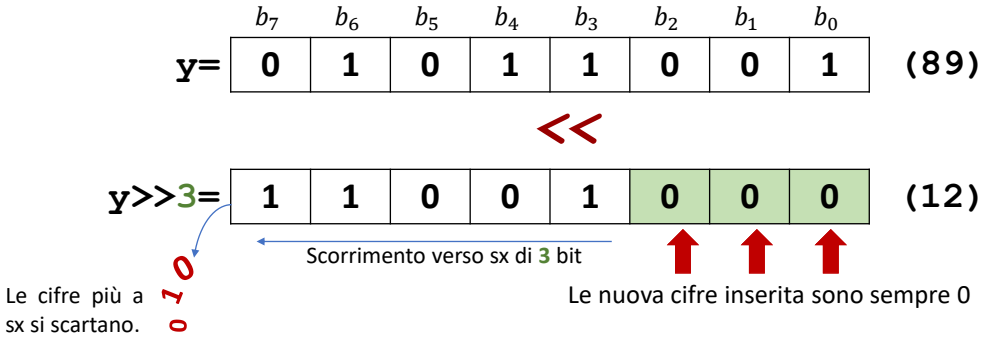
```
unsigned char y=24;
```



23

Operatori «bitwise»: esempio <<

```
unsigned char y=25;
```



24

Operatori *bitwise* composti

- Assegnano al una variabile un nuovo valore che dipende da quello precedente
- Si compongono di uno tra i seguenti operatori e il simbolo =

Op.	esempio	equivale a…
`&=`	`var1 &= <exp>`	`var1=var1 & <exp>`
`|=`	`var2 |= <exp>`	`var2=var2 | <exp>`
`^=`	`var3 ^= <exp>`	`var3=var3 ^ <exp>`
`<<=`	`var4 <<= <exp>`	`var4=var4 << <exp>`

`<<`	`>>`	`&`	`|`	`^`

25

Esempio: *da binario a stringa*

Si scriva la funzione C

```
void uint_to_string (unsigned int num, char *numstr)
```

che riempe l'array **`numstr`** con i caratteri che compongono la rappresentazione binaria di **`num`**.

26

Esempio: *da binario a stringa*

```
1  #include<stdio.h>
2
3  void uint_to_string(unsigned int num, char* numstr)
4  {
5      int i;
6      numstr[32]='\0';
7      for(i=31;i>=0;i--) {
8          if(num&1)
9              numstr[i]='1';
10         else
11             numstr[i]='0';
12         num>>=1;
13     }
14 }
…  …
```

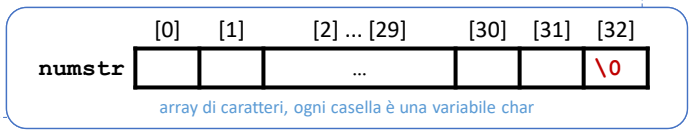
27

Esempio: *da binario a stringa*

```
1  #include<stdio.h>
2
3  void uint_to_string(unsigned int num, char* numstr)
4  {
5      int i;
6      numstr[32]='\0';
7      for(i=31;i>=0;i--) {
8          if(num&1)
9              numstr[i]='1';
10         else
11             numstr[i]='0';
12         num>>=1;
13     }
14 }
…  …
```

Mettiamo il *terminatore di stringa* alla fine dell'array.

Riempiamo l'array da destra a sinistra, seguendo l'ordine di estrazione delle cifre dall'intero **num**



28

Esempio: *da binario a stringa*

```
1  #include<stdio.h>
2
3  void uint_to_string(unsigned int num, char* numstr)
4  {
5      int i;
6      numstr[32]='\0';
7      for(i=31;i>=0;i--) {
8          if(num&1)
9              numstr[i]='1';
10         else
11             numstr[i]='0';
12         num>>=1;
13     }
14 }
…  …
```

variabile intera, ogni casella è un bit della rappresentazione binaria

	b_{31}	b_{30}	$b_{29} \dots b_2$	b_1	b_0
num	1	0	…	0	1
num&1	0	0	…	0	1

	0	1	2 … 29	30	31	32
numstr			…			\0

array di caratteri, ogni casella è una variabile char

29

Esempio: *da binario a stringa*

```
1  #include<stdio.h>
2
3  void uint_to_string(unsigned int num, char* numstr)
4  {
5      int i;
6      numstr[32]='\0';
7      for(i=31;i>=0;i--) {
8         if(num&1)
9              numstr[i]='1';
10         else
11             numstr[i]='0';
12         num>>=1;
13     }
14 }
…  …
```

variabile intera, ogni casella è un bit della rappresentazione binaria

	b_{31}	b_{30}	$b_{29} \dots b_2$	b_1	b_0
num	1	0	…	0	1
num&1	0	0	…	0	1

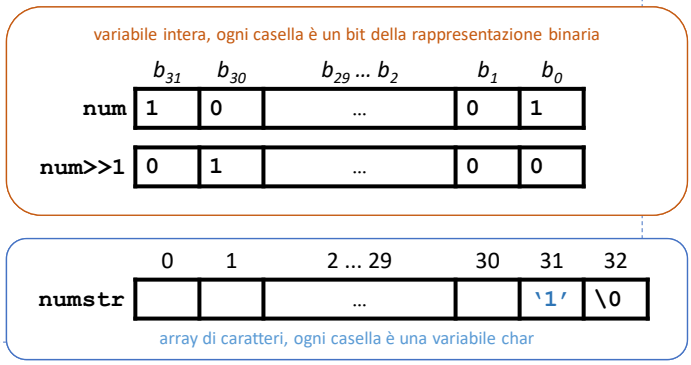
	0	1	2 … 29	30	31	32
numstr			…		'1'	\0

array di caratteri, ogni casella è una variabile char

30

Esempio: *da binario a stringa*

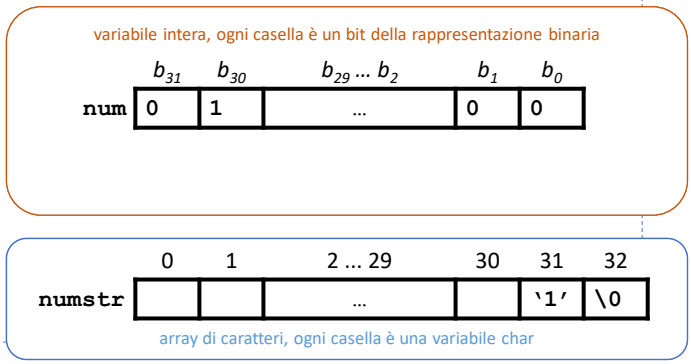
```
1  #include<stdio.h>
2
3  void uint_to_string(unsigned int num, char* numstr)
4  {
5      int i;
6      numstr[32]='\0';
7      for(i=31;i>=0;i--) {
8          if(num&1)
9              numstr[i]='1';
10         else
11             numstr[i]='0';
12         num>>=1;
13     }
14 }
…  …
```



31

Esempio: *da binario a stringa*

```
1  #include<stdio.h>
2
3  void uint_to_string(unsigned int num, char* numstr)
4  {
5      int i;
6      numstr[32]='\0';
7      for(i=31;i>=0;i--) {
8          if(num&1)
9              numstr[i]='1';
10         else
11             numstr[i]='0';
12         num>>=1;
13     }
14 }
…  …
```



32

Esempio: *da binario a stringa*

```
1  #include<stdio.h>
2
3  void uint_to_string(unsigned int num, char* numstr)
4  {
5      int i;
6      numstr[32]='\0';
7      for(i=31;i>=0;i--) {
8          if(num&1)
9              numstr[i]='1';
10         else
11             numstr[i]='0';
12         num>>=1;
13     }
14 }
…  …
```

variabile intera, ogni casella è un bit della rappresentazione binaria

	b_{31}	b_{30}	$b_{29} \dots b_2$	b_1	b_0
num	0	1	…	0	0
num&1	0	1	…	0	0

	0	1	2 … 29	30	31	32
numstr			…		'1'	\0

array di caratteri, ogni casella è una variabile char

33

Esempio: *da binario a stringa*

```
1  #include<stdio.h>
2
3  void uint_to_string(unsigned int num, char* numstr)
4  {
5      int i;
6      numstr[32]='\0';
7      for(i=31;i>=0;i--) {
8          if(num&1)
9              numstr[i]='1';
10         else
11             numstr[i]='0';
12         num>>=1;
13     }
14 }
…  …
```

variabile intera, ogni casella è un bit della rappresentazione binaria

	b_{31}	b_{30}	$b_{29} \dots b_2$	b_1	b_0
num	0	1	…	0	0
num&1	0	1	…	0	0

	0	1	2 … 29	30	31	32
numstr			…	'0'	'1'	\0

array di caratteri, ogni casella è una variabile char

34

Esempio: *da binario a stringa*

```
1  #include<stdio.h>
2
3  void uint_to_string(unsigned int num, char* numstr)
4  {
5      int i;
6      numstr[32]='\0';
7      for(i=31;i>=0;i--) {
8          if(num&1)
9              numstr[i]='1';
10         else
11             numstr[i]='0';
12         num>>=1;
13     }
14 }
…  …
```

variabile intera, ogni casella è un bit della rappresentazione binaria

	b_{31}	b_{30}	b_{29}	…	b_2	b_1	b_0
num	0	1	1	…	0	0	1

	[0]	[1]	[2]	…	[29]	[30]	[31]	[32]
numstr	'0'	'1'	'1'	…	'0'	'0'	'1'	\0

array di caratteri, ogni casella è una variabile char

35

Esempio: *da binario a stringa*

```
16 int main()
17 {
18     unsigned int num=0;
19     char uint_digits[33];
20     printf("size of num: %d bytes (%d bits)\n\n",sizeof(num),sizeof(num)*8);
21
22     do {
23         printf("enter 'num'(0 per uscire): ");
24         scanf("%d",&num);
25         if (num==0)
26             break;
27         uint_to_string(num,uint_digits);
28         printf("uint_digits: %s\n\n",uint_digits);
29     } while(1);
30 }
…  …
```

36

Esercizio: *da stringa a binario*

Si scriva la funzione C

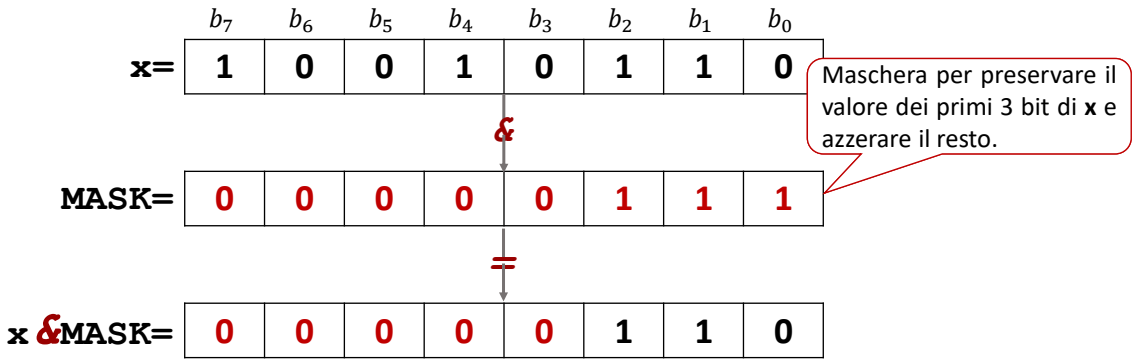
```
int string_to_uint (char *numstr)
```

che restituisce l'intero la cui rappresentazione binaria è contenuta nell'array **`numstr`**.

37

Operazioni con le «*maschere*» di bit

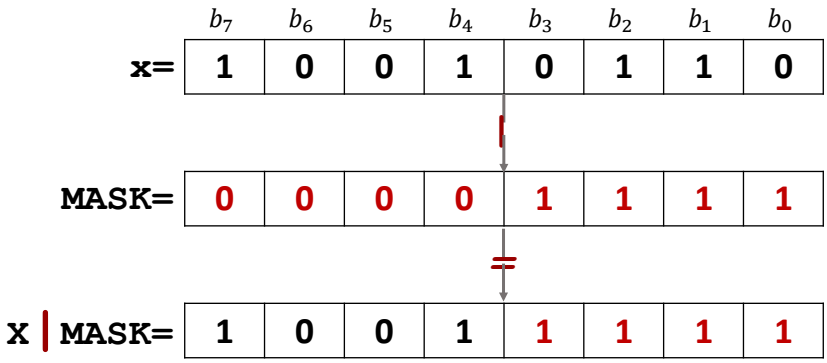
Azzerare tutti i bit tranne alcuni:



38

Operazioni con le *«maschere»* di bit

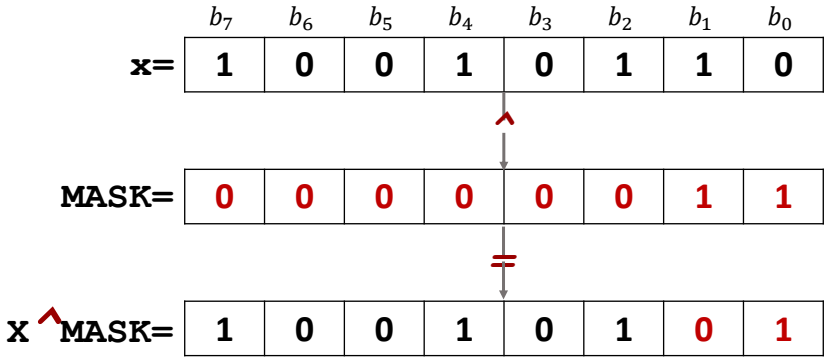
Porre a 1 tutti i bit tranne alcuni:



39

Maschere di bit

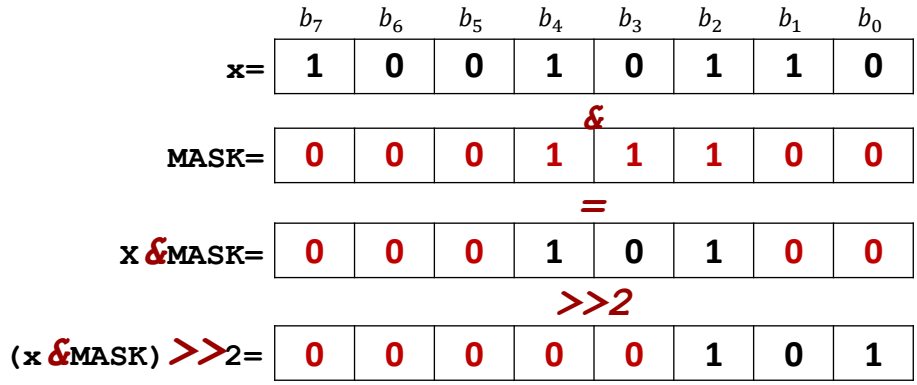
"cambiare" lo stato di certi bit e lasciarne invariati altri:



40

Operazioni con le «*maschere*» di bit

Estrarre il valore di una *sottoparola* di x (e.g. I bit da 2 a 4):



Esercizi: *operatori bitwise*

Si scrivano le seguenti funzioni C

1. `int bit1count(int num)`

   che restituisce il numero di bit a 1 nella rappresentazione binaria di **`num`**;

2. `int bit_value(int num,int n)`

   che restituisce il valore del bit **`n`**-esimo di **`num`**;

3. `int subword_value(int num,int n,int len)`

   che restituisce il valore della sottosequenza di **`num`** lunga **`len`** bit che inizia dal bit **`n`**-esimo;

Altri operatori

43

Operatore «virgola»

- L'operatore «virgola» ha la seguente sintassi:

$$lval = \langle exp_1 \rangle, \langle exp_2 \rangle, \ldots, \langle exp_n \rangle;$$

- Le espressioni exp_i sono valutate da sinistra verso destra
  - Inclusi assegnamenti, incrementi etc.
- Il valore di ciascuna espressione (tranne quella più a destra) è scartato
- `lval` assume il valore dell' espressione «più a destra»
  - È indispensabile che ci sia concordanza/compatibilità tra i tipi di `lval` e exp_n

44

Operatore «virgola»: un tipico esempio…

- Spessp, nelle espressioni con l'operatore virgola, quello che interessa è il cosiddetto «effetto collaterale», cioè : *la variazioni del valore di alcuni degli operandi,* piuttosto che il risultato finale*;*
  - Come in: **`a=b+c, c++, a/=7`**…
- Possiamo utilizzare la virgola nei cicli **`for`**, quando vogliamo che più variabili indice siano utilizzate contemporaneamente *(purchè inizializzazione e «aggiornamento» avvengano sempre simultaneamente)*

45

Operatore «virgola»: un tipico esempio…

```
#include<stdio.j>

int main()
{
        int i, j, sequenza[10]={1,2,5,0,1,1,0,5,2,1};
        int palindroma=1;

        printf("La sequenza ");

        for(i=0,j=9; j>i; i++,j--)
                if(sequenza[i]!=sequenza[j]){
                                palindroma=0;
                                break;
                        }
        if (palindroma)
                printf("e' ");
        else
                printf("non e' ");
        printf("palindroma.\n");
}
```

```
La sequenza e' palindroma.
```

46

Operatore «condizionale» ?:

- L'operatore «condizionale» ? : è un operatore ternario e ha la seguente sintassi:

$$exp = < cond > ? < sub\_exp_1 > : < sub\_exp_2 >$$

- L'operatore valuta la condizione $cond$ (a valori «booleani»)
  - Se è vera, si procede alla valutazione di sub_exp_1
  - Se è falsa si procede alla valutazione di sub_exp_2
- exp assume il valore della sub espressione valutata;

47

Operatore «condizionale» ?:

```
int main()
{
        int a=0,b=0,c=0;
        int cond=0;

        a=cond ? b=10 : c=20;
        printf("a=%d ",a);
        printf("b=%d, c=%d\n",b,c);
        cond=1;
        a=b=c=0;
        a=cond ? b=10 : c=20;
        printf("a=%d ",a);
        printf("b=%d, c=%d\n",b,c);
}
```

```
a=20, b=0, c=20
a=10, b=10, c=0
```

Nel primo caso, b è uguale a zero poiché, essendo falsa la condizione, la prima espressione non è valutata e quindi neppure l'assegnamento ha luogo.

Nel secondo caso, è la seconda espressione (l'assegnamento c=20) a non essere valutata, pertanto, il valore della variabile c resta invariato;

48

Precedenza tra gli operatori (1/2)

Operatori	Descrizione
++, --	incremento/decremento postfisso
++, --	incremento/deceremento prefisso
+, -, !	operatori unari
(type)	conversione esplicita di tipo (cast)
*, /, %	aritmetici moltiplicativi
+, -	aritmetici additivi
<<, >>	scorrimento (bitwise)
<, <=,>,>=, ==	relazionali

49

Precedenza tra gli operatori (2/2)

Operatori	Descrizione
&	and bitwise
^	xor bitwise
|	or bitwise
&&	and logico
||	or logico
?:	condizionale
+=, -=, …	composti
,	virgola (valutazione sequenziale)

50