



## C-ohjelmointi: Bittioperaatiot

Päivi Kuuppelomäki  
 10.10.2008



## Bittioperaatiot

$n$  &  $|$   $\wedge$  vertaavat alkioita bitti kerrallaan ja palauttavat tämän tuloksen

$n \ll \gg$  siirtävät bittejä sanan sisällä

$n \sim$  yhden komplementti  
 0 -> 1 ja 1 -> 0

low-order byte

- &** bitwise and
- |** bitwise or
- ^** bitwise xor (exclusive or)
- <<** left shift
- >>** right shift
- ~** one's complement

$x$	...		B
<b>&amp;</b>	0...0	0...0	11111111
$x \& 0xff$	0...0		B



## & (bitti and)

∩ jos  $(x)_i == 1$  ja  $(y)_i == 1$  niin  $(x \& y)_i == 1$

∩ muuten  $(x \& y)_i == 0$

∩ Tätä käytetään usein bittien nollaukseen, esim:

∩  $x \& 0\text{xff}$  nolaa sanan kaikki muut bitit paitsi vähiten merkitsevän tavun (low-order byte):

low-order byte

	x	...		B
&				
	0xff	0...0	0...0	11111111
	x & 0xff	0 ... 0		B



## | (bitti or)

∩ jos  $(x)_i == 1$  tai  $(y)_i == 1$  niin  $(x | y)_i == 1$

∩ muuten  $(x | y)_i == 0$

∩ Tällä usein asetetaan tiettyjä bittejä ykkösiksi.

∩ Asetetaan x:n alimman tavun ylin bitti ykköseksi:

$x | 0\text{x80}$ ,

low-order byte

	x	C	B <sub>1</sub> B <sub>2</sub> B <sub>3</sub> ...
	0x80	0...0	10000000
	x   0x80	C	1B <sub>2</sub> B <sub>3</sub> ...



### **^ (bitti xor)**

n jos  $(x)_i == (y)_i$  niin  $(x \wedge y)_i == 0$   
n muuten  $(x \wedge y)_i == 1$

- n Bittioperaatio  $\wedge$  (xor, poissulkeva or) nollaa ne bittipositiot, joiden arvo on sama ,
- n ja asettaa ykköseksi ne bittipositiot, jotka olivat eriarvoiset.
- n Yksinkertainen testi sanojen samuudelle:  $x \wedge y$
- n palauttaa 0 jos  $x$  ja  $y$  samat (eli kaikki bitit vastasivat toisiaan).



### **~ (yhden komplementti)**

n **Yhden komplementti:**  $\sim x$

$$(\sim x)_i == 1 - (x)_i$$

- n Bitit vaihtuvat vastakkaisiksi.
- n Esimerkiksi. Halutaan vain nollata sanan  $x$  3 alinta bittiä ja jättää loput ennalleen:  $x \& \sim 7$   
(Niin miten tuo oikein toimii?)
- n Vastaavasti, kun halutaan asettaa kaikki bitit ykkösiksi  $\sim 0$



## << Siirto vasemmalle

n **Left shift:**  $i \ll j$

n Tuloksena on sana, jossa bitit ovat siirtyneet vasemmalle  $j$  paikkaa

n Sana on täydennetty oikealta tarvittavalla määrällä nollia.

n Tämä on näppärä kahden kertolaskuissa:

n  $x \ll= 1$  on sama kuin  $x *= 2$

n  $x \ll= 3$  on sama kuin  $x *= 8$ .



## >> Siirto oikealle

n **Right shift:**  $i \gg j$

n Tuloksena on sana, jossa bitit ovat siirtyneet  $j$  paikkaa oikealle.

n Etumerkittömillä tyypeillä (ja positiivisilla luvuilla) oikealta täydennetään aina nolilla.

n Etumerkillisillä ja negatiivisilla luvuilla täydennyksenä voi olla 0 tai 1 toteutuksesta riippuen. (MIKSI?)

n  $x \gg= 1$  is equivalent to  $x /= 2$

n  $x \gg= 2$  is equivalent to  $x /= 4$ .



## Esimerkki:

```
#include <stdio.h>
/* Bittipeliä*/
int main(void)
{
    enum {LL = 011 };
    int i, j;

    i = 0;
    j = i | LL;
    printf("i: %d, LL (okt):%o, i|LL: %d, oktaalina %o\n",
           i, LL, j, j);

    printf("1 & 6: %d, 1 && 6: %d\n",
           1 & 6, 1 && 6);

    printf("1<<3: %d, 8>>3: %d\n",
           1<<3, 8>>3);
    return 0;
}
```

Tulostaa:

```
i: 0, LL (okt):11, i|LL: 9, oktaalina 11
1 & 6: 0, 1 && 6: 1
1<<3: 8, 8>>3: 1
```