

HELSINGIN YLIOPISTO  
HELSINGFORS UNIVERSITET  
UNIVERSITY OF HELSINKI

## C-ohjelmointi: Bittiopeeraatiot

Päivi Kuuppelomäki  
10.10.2008

### Bittiopeeraatiot

- $\&$  ja  $\wedge$  vertaavat alkioita bitti kerrallaan ja palauttavat tämän tuloksen
- $\ll$  ja  $\gg$  siirtävät bittejä sanan sisällä
- $\sim$  yhden komplementti  
0  $\rightarrow$  1 ja 1  $\rightarrow$  0

	low-order byte		
x	...		B
$\&$			
0xff	0...0	0...0	11111111
x & 0xff	0...0		B

$\&$  bitwise and  
 $|$  bitwise or  
 $\wedge$  bitwise xor (exclusive or)  
 $\ll$  left shift  
 $\gg$  right shift  
 $\sim$  one's complement

### $\&$ (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:  
 $x \& 0xff$  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:  
 $x | 0x80$ ,

	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> ...

### $\wedge$ (bitti xor)

- jos  $(x)_i == (y)_i$  niin  $(x \wedge y)_i == 0$
- muuten  $(x \wedge y)_i == 1$
- Bittiopeeraatio  $\wedge$  (xor, poissulkeva or) nolaa ne bittipositiot, joiden arvo on sama,
- ja asettaa ykköseksi ne bittipositiot, jotka olivat eriarvoiset.
- Yksinkertainen testi sanojen samuudelle:  $x \wedge y$
- palauttaa 0 jos x ja y samat (eli kaikki bitit vastasivat toisiaan).

### $\sim$ (yhden komplementti)

- Yhden komplementti:  $\sim x$
- $(\sim x)_i == 1 - (x)_i$
- Bitit vaihtuvat vastakkaisiksi.
- Esimerkiksi. Halutaan vain nolata sanan x 3 alinta bittiä ja jättää loput ennalleen:  $x \& \sim 7$   
(Niin miten tuo oikein toimii?)
- Vastaavasti, kun halutaan asettaa kaikki bitit ykkösiksi  $\sim 0$



### << Siirto vasemmalle

n **Left shift:** `i << 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 kertolaskussa:

n `x <<= 1` on sama kuin `x *= 2`

n `x <<= 3` on sama kuin `x *= 8`.



### >> Siirto oikealle

n **Right shift:** `i >> j`

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

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

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

n `x >>= 1` is equivalent to `x /= 2`

n `x >>= 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
```