C under Linux

Dr. Naeem Odat

Department of Computer and Communications Engineering

C Operators II

Operators and typecasting

Test, clear and set bits

Testing bits

- A 1 in the bit position of interest is AND'ed with the operand.
- The result is non-zero if and only if the bit of interest is 1
- Example:

```
if ((bits & 64) != 0) /* Check to see if bit 6 is set. */
if  (bits & (1 << 6)) /* check to see if bit 6 is set */
```

$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \& 01000000 => 0 b_6 000000$

```
#include <stdio.h>
int main(){/* To test whether a bit in a number is ON or OFF  */
    int i = 65, j;
    printf("\nvalue of i = %d\n", i);
    j = i & 32 ;
    if ( 0 == j)
        printf("fifth bit is OFF\n");
    else
        printf("fifth bit is ON\n");
    return 0;
}
```

Test, clear and set bits

Clear bits

- Clearing a bit to 0 is accomplished with the bitwise-AND operator:

```
bits=bits& ~(1 << 7) ; /* clears bit 7 */
```

Set bits

- Setting a bit to 1 is easily accomplished with the bitwise-OR operator:

```
bits = bits | (1 << 7) ; /* sets bit 7 */
```

Increment/decrement operators

Increment/decrement operators

- Increment operator ++ adds 1 to its operand, while decrement operator - - subtracts 1
- The unusual aspect is that ++ and - - may be used either as prefix operators (before the variable : ++n) or postfix (after the variable: n++)

Example

```
#include <stdio.h>
int main(){
    int nNum1 = 10;
    printf("++nNum1 = %d\n", ++nNum1);/*Pre increment*/
    printf("nNum1++ = %d\n", nNum1++);/*Post Increment*/
    printf("nNum1 = %d\n", nNum1);
    return(0);
}
```

Assignment operators

Other than =

+=, -=, *=, /= , %=, etc.

Typecasting

Typecasting

- In general, a "narrower" operand is automatically converted into a "wider" one without loosing information.
- Such as converting an integer to a floating point in an expression like f + i.
- To force conversion use cast.
- Syntax:

```
(type-name) expression
```

- Example:

```
float f = 10.5;
int i;
i = (int) f;
```

Operator precedence and associativity

Operators	**Associativity**
() [] -> . ++ -- (Postfix)	Left to right
! ~ ++ -- - * & (*type*) *sizeof*() (Unary)	Right to left
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Left to right
== !=	Left to right
&	Left to right
^	Left to right
|	Left to right
&&	Left to right
||	Left to right
? :	Right to left
= += -= *= /= %= &= ^= |= <<= >>=	Right to left
,	Left to right