

Introduction to C, C++, and Unix/ Linux

CS 60

Lecture 5: Operators

Today

→ C operators

- Reading for Next class: K&R ch. 1 – 3 & 7.1-7.4

Note: Lexical elements of C

- **Keywords**
 - Reserved words that may not be used for anything else
 - **Identifiers**
 - Variable names, function names...
 - **Constants**
 - E.g., the number 5
 - **String constants**
 - E.g. “Hello, world\n”
 - **Operators**
 - E.g., +, -, =, ++
 - **Punctuators**
 - E.g., {} () ; ,
- These are the basic tokens that the compiler cares about

C operators

Specify action

- Arithmetic

+ - * / %

plus, minus, times, divide, modulus.
→ two operands

- Unary sign

+ -

positive, negative
→ one operand

- Logical

! || && == !=

not, or, and, equal to, not equal to

- Relational

> >= < <=

gt, geq, lt, leq

- Logical and relational operators return
 - 1 if TRUE
 - 0 if FALSE

Value of **x**?

x = (3 == 5) ;	0
x = !12 ;	0
x = !0 ;	1
x = 101 && 102 ;	1
x = 17 12 ;	1
x = (3 < 5) ;	1

C operators (cont.)

- Bitwise operators – bit by bit

&	AND
 	OR
^	exclusive OR
<<	left shift
>>	right shift (unsigned)
~	one's complement (flips the bits)

```
a = 11110000
b = 00010010
```

```
char a, b;
a = 0xf0;
b = 0x12;
```

a&b	0x10
a b	0xf2
a^b	0xe2
b<<1	0x24
b>>1	0x09
~a	0x0f

Common mistakes

- Don't confuse **&&** with **&**, or **||** with **|**
- Don't mistake **^** for power operator which can be computed with **math.h** (use **pow(x,y)**)

```
if (x & y) printf("Both");
```

```
if (x | y) printf("Yes");
```

```
x = 2^8;
```

NO!

x will be **10**

C operators (cont.)

```
int delete = 0;
if (delete = 1)
    DeleteAllFiles();
```

- Assignment

=

- Note that the assignment operator in C is not equivalent to the mathematical equals sign
 - $x + 2 = 0$ (Not legal in C)
 - $x = x + 1$ (Not allowed in mathematics)
- Also, remember that = is assignment, not logical equivalence (==)!!!!!!

Assignment

- Like most C expressions, `=` returns a value

```
x = (y = 2) + (z = 3)
```

```
x = 5
```

```
while (c=getvalue()) {...}
```

C operators (cont.)

- *op* = shortcut

`+=` `-=` `*=` `/=` `%=` `<<=` `>>=`
`&=` `|=` `^=`

`var op= expr` \iff `var = var op (expr)`

`x += 1;`

`x = x + 1;`

C operators (cont.)

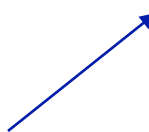
- Increment / decrement by 1 (unary operators)

var++ **++var**

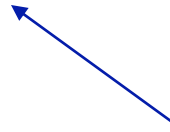
var-- **--var**

- Difference between **x++** and **++x** ?

Incremented **after**
expression is evaluated



Incremented **before**
expression is evaluated



```
num = i++;  →  num = i;  
              i += 1;
```

```
num = ++i;  →  i += 1;  
              num = i;
```

What is x?

```
num = 1;  
x = num++;
```

```
num = 1;  
x = ++num;
```

What is the value of **x**?

Moral: Don't do this!

```
int x, y;  
y = 3;  
x = ++y + ++y;
```

x is 10

```
y += 1;  
y += 1;  
x = y + y;
```

```
int x, y;  
y = 3;  
x = y++ + ++y;
```

x is 8

```
y += 1;  
x = y + y;  
y += 1;
```

It's not a good idea to use ++ or -- on the same variable more than once in a single expression

C operators (cont.)

- Conditional operator

? :

- It is *ternary* – it takes three arguments

expr1 ? expr2 : expr3

- This does the work of an *if-then-else* construct

```
if (x < 0)
    y = -x;
else
    y = x;
```



```
y = (x < 0) ? -x : x
```

```
printf("%d day%s\n", x,
       (x > 1) ? "s" : "");
```

"1 day" "2 days" "3 days"

C operators (cont.)

Often used in **for** loop initialization

- Comma operator (,)
expr1, expr2;

Statements separated by commas are evaluated left to right

The type and value of the result are that of the rightmost statement

i=0, j=i+100; x=(i=0, j=1);

Operator precedence

- It's important to know the rules for precedence and associativity of operators
 - See Table 2.1 in K&R
- Some operators have higher precedence than others
- Most are associated left to right, but some are right to left

$x \ll y \ll z;$

$(x \ll y) \ll z;$

$x = y = z;$

$x = (y = z);$

Precedence



Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary prefix	+ - ! ~ ++ -- (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	? :	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right

Example 1

```
int x
```

```
x = 3 + 6 / 2
```

```
x = 1 == 2 && 3 == 1
```

Example 2

- What is x?

```
int i=1;  
int j = 2;  
x = i<<8/2==j;  
      ↑  ↑  ↑
```

x is 0

(unless j=16, then x is 1)

Same as

```
x = ((i<<(8/2))==j);
```

Example 3

$j *= k + 2;$

?

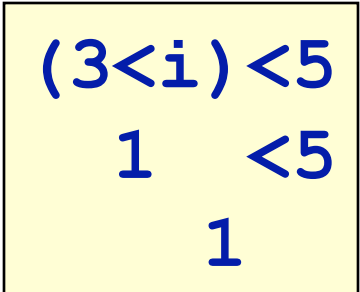
$j = j * k + 2;$

$j = j * (k + 2);$

Example 4

- BUY or SELL?

```
int i=7;  
if (3<i<5)  
    printf("BUY");  
else  
    printf("SELL");
```



(3<i)<5
1 <5
1



BUY

General warning



- Avoid the temptation to write obscure C code
- It's easy to do, but if the code is not easy to read and comprehend, it is a ticking bomb!
- Some C constructs (like `++`, `?:`, `,`) and preprocessor macros are powerful but can be dangerous
 - Use them wisely!


```

m(f,a,s)char*s;
{char c;return f&1?a!=*s++?m(f,a,s):s[11]:f&2?a!=*s++?1+m(f,a,s):1:f&4?a--?
putchar(*s),m(f,a,s):a:f&8?*s?m(8,32,(c=m(1,*s++,"Arjan Kenter. \no$../\\"", m(4,m(2,*s+
+,"POCnWAUvBVxRsoqatKJurgXYyDQbzhLwkNjdMTGeISchFmpIiZEF"),&c),s)): 65:
(m(8,34,"rgeQjPruaOnDaPeWrAaPnPrCnOrPaPnPjPrCaPrPnPrPaOrvaPndeOrAnOrPnOrP\
nOaPnPjPaOrPnPrPnPrPtPnPrAaPnBrnnsrnnBaPeOrCnPrOnCaPnOaPnPjPtPnAaPnPrPnPrCaPn\
BrAnxrAnVePrCnBjPrOnvrCnxrAnxrAnsrOnvjPrOnUrOnornnsrnnorOtCnCjPrCtPnCrnnirWtP\
nCjPrCaPnOtPrCnErAnOjPrOnvtPnnrCnNrnRePjPrPtnrUnnrntPnbtPrAaPnCrnnOrPjPrRtPn\
CaPrWtCnKtPnOtPrBnCjPronCaPrVtPnOtOnAtnrxaPnCjPrqannaPrtaOrsaPnCtPjPratPnnaPrA\
aPnAaPtPnnaPrvaPnnjPrKtPnWaOrWtOnnaPnWaPrCaPnntOjPrprtOnWanrOtPnCaPnBtCjPrYtOn\
UaOrPnVjPrwtnnxjPrMnBjPrTnUjP"),0);}
main(){return m(0,75,"mIWltouQJGsBniKYvTxODAFbUcFzSpMwNCHEgrdLaPkyVRjXeqZh");}

```

Example: What will this program output?

