C Programming
Lecture 6 : Operators

Lecture notes : courtesy of Ohio Supercomputing Center, and Prof. Woo and Prof. Chang

Expressions and Statements

- **Expression**
  - Combination of constants, variables, operators, and function calls
  - Ex)

```
a+b
3.0*x – 9.66553
tan(angle)
```

- **Statement**
  - An expression terminated with a semicolon
  - Ex)

```
 sum = x + y + z;
 printf(“Dragons!”);
```

Assignment Operator

- The equal sign = is an assignment operator
- Used to give a variable the value of an expression
  - Ex)

```
x=34.8;
sum=a+b;
slope=tan(rise/run);
midinit=‘J’;
j=j+3;
x=y=z=13.0;
```

- Initialization
  - Ex)

```
int i=0;
```

Arithmetic operators

- Binary operators
  - Addition : +
  - Subtraction : -
  - Multiplication : *
  - Division : /
  - Modulus : % // only works for integers values
- Unary operators
  - + , -
- Integer division
  - 1/2 = 0 (?) , 3/2 = 1 (?)

Arithmetic operators

- In binary operators
  - If two operands are int type : the result is int type
  - If one or two operands are floating-point type : the result is floating-point type
    - `2 + 3.14 ⇒ 2.0 + 3.14 = 5.14`
    - `12.0/5 ⇒ 12.0/5.0 = 2.4`

increment/decrement

- Increment operator ++
  - `i=i+1;`
    - `i++;         // postfix form`
    - `++i;         // prefix form`
- decrement operator –
  - **i=i-1;**
    - **i--;   // postfix form**
    - **--I;   // prefix form**

- **Difference between `i++` and `++i` ?**

prefix vs. postfix

- Difference shows up when the operators are used as a part of a larger expression
  - **++k** : k is incremented before the expression is evaluated.
  - **k++** : k is incremented after the expression is evaluated.
- Ex) difference?

```
int a;
int i=0, j=0;
a= (++i) + (++j);
```

```
int b;
int i=0, j=0;
b= (i++) + (j++);
```

Shorthand Operators

- General syntax
  - ***variable = variable op expression;***

    *i*s equivalent to

    variable op= expression;

- Common forms
  - **+=, -=, *=, /=, %=**

- Examples

```
j=j*(3+x); j *= 3+x;

a=a/(s-5); a /= s-5;
```

Precedence , Associativity of Operators

- **Operator Precedence**
  - determines the order in which operations are performed
  - operators with higher precedence are employed first.

precedence	operators
1st	unary + , unary -
2nd	binary * / %
3rd	binary + -

- **Operator Associativity**
  - if two operators in an expression have the same precedence, **associativity** determines the direction in which the expression will be evaluated.

```
* , / , %   : L -> R
+ , - (bin): L -> R
=           : R -> L
+ , - (unary) : R -> L
```

Precedence Examples

- **Evaluation Order**

```
    1 + 2 * 3 - 4
->  1 + 6 - 4
->  7 - 4
->  3
```

  - use **parenthesis** to force a desired order of evaluation
  - Ex)

```
(1 + 2) * (3 – 4)
```

Associativity Examples

- Left associativity

```
a / b * c  →  (a / b) * c
```

- Right associativity

```
- + - a  →  - (+ (- a))
```

Bitwise Operators

shift/logic	Op. name	usage	type	output
shift op.	left shift	`a<<n`	integer	Shift bits of `a` to left by n bit Newly created bits will be 0
	right shift	`a>>n`	integer	Shift bits of `a` to right by n bit Newly created bits will be 0
bit op.	bit AND	`a & b`	integer	AND of `a`'s and `b`'s each bit
	bit OR	`a | b`	integer	OR of `a`'s and `b`'s each bit
	bit XOR	`a ^ b`	integer	XOR of `a`'s and `b`'s each bit
	1's complement	`~a`	integer	1's complement of `a`

Truth/False Table

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

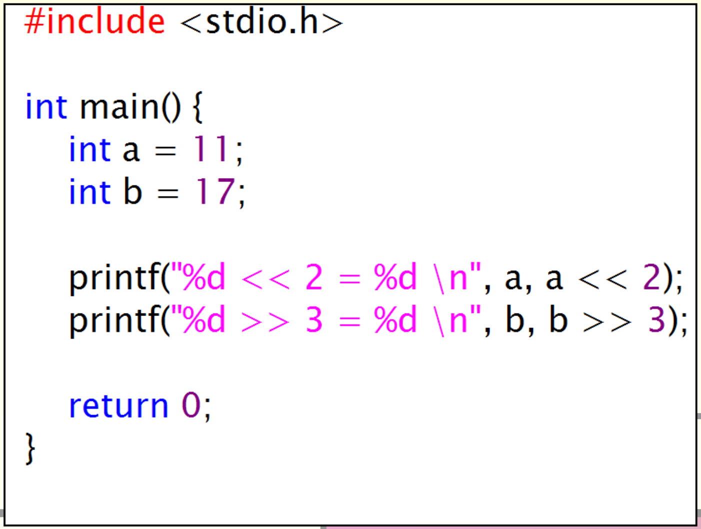
a	~a
0	1
1	0

Bitwise Operators Examples

- 11 = 0000 0000 0000 1011
- 17 = 0000 0000 0001 0001

- 11 << 2
- 0000 0000 0000 1011 << 2 = 0000 0000 0010 1100 = 44

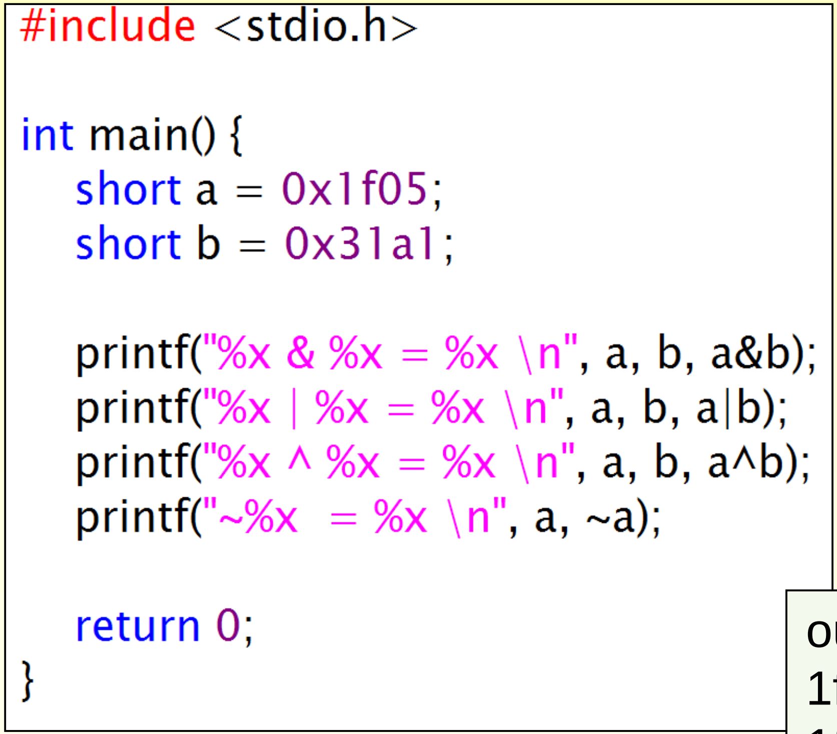
- 17 >> 3
- 0000 0000 0001 0001 >> 3 = 0000 0000 0000 0010 = 2



```c
#include <stdio.h>

int main() {
    int a = 11;
    int b = 17;

    printf("%d << 2 = %d \n", a, a << 2);
    printf("%d >> 3 = %d \n", b, b >> 3);

    return 0;
}
```

```
output:
11 << 2 = 44
17 >> 3 = 2
```

example



```c
#include <stdio.h>

int main() {
    short a = 0x1f05;
    short b = 0x31a1;

    printf("%x & %x = %x \n", a, b, a&b);
    printf("%x | %x = %x \n", a, b, a|b);
    printf("%x ^ %x = %x \n", a, b, a^b);
    printf("~%x  = %x \n", a, ~a);

    return 0;
}
```

```
output:
1f05 & 31a1 = 1101
1f05 | 31a1 = 3fa5
1f05 ^ 31a1 = 2ea4
~1f05  = ffffe0fa
```

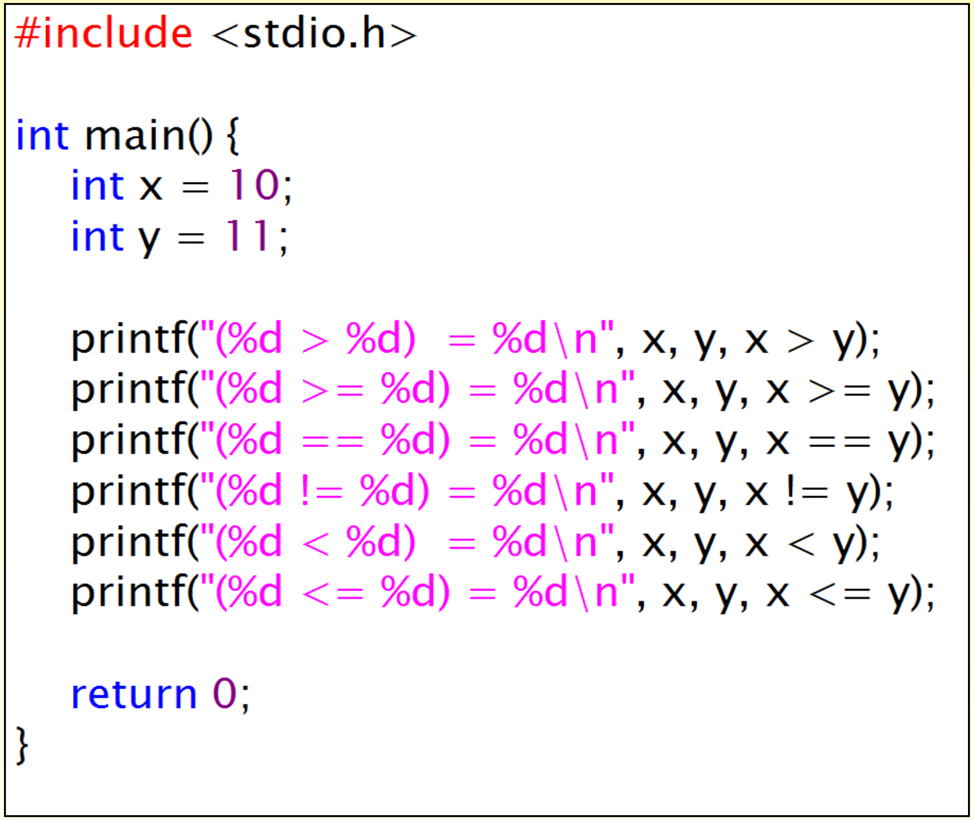
example

expression	value	result
a	0x1f05	0001 1111 0000 0101
b	0x31a1	0011 0001 1010 0001
~a	0xe0fa	1110 0000 1111 1010
a & b	0x1101	0001 0001 0000 0001
a | b	0x3fa5	0011 1111 1010 0101
a ^ b	0x2ea4	0010 1110 1010 0100

Relational Operators

meaning	연산자	자료형	결과값
Equal	a == b	integer or floating point	1(=true) if a is equal to b otherwise 0(=false)
not equal	a != b	integer or floating point	1(=true) if a is not equal to b otherwise 0(=false)
less than	a < b	integer or floating point	1(=true) if a is less than b otherwise 0(=false)
less than or equal to	a <= b	integer or floating point	1(=true) if a is less than or equal to b otherwise 0(=false)
greater than	a > b	integer or floating point	1(=true) if a is greater than b otherwise 0(=false)
greater than or equal to	a >= b	integer or floating point	1(=true) if a is greater than or equal to b otherwise 0(=false)

example



```c
#include <stdio.h>

int main() {
    int x = 10;
    int y = 11;

    printf("(%d > %d)  = %d\n", x, y, x > y);
    printf("(%d >= %d) = %d\n", x, y, x >= y);
    printf("(%d == %d) = %d\n", x, y, x == y);
    printf("(%d != %d) = %d\n", x, y, x != y);
    printf("(%d < %d)  = %d\n", x, y, x < y);
    printf("(%d <= %d) = %d\n", x, y, x <= y);

    return 0;
}
```

```
output:
(10 > 11)  = 0
(10 >= 11) = 0
(10 == 11) = 0
(10 != 11) = 1
(10 < 11)  = 1
(10 <= 11) = 1
```

Logical Operators

op name	expression	meaing
logical NOT	! a	If a is false, then 1(=true), otherwise 0(=false)
logical AND	a && b	If both a and b are true, then 1(=true), otherwise 0(=false)
logical OR	a || b	If either a or b is true, then 1(=true), otherwise 0(=false)

example

```c
#include <stdio.h>

int main()
{
    int score;

    printf("Score?");
    scanf("%d",&score);
    if (score >= 90 && score <=100)
          printf("your grade is A.\n");
    if (score >= 80 && score < 90)
          printf("your grade is B.\n");
    if (score >= 70 && score < 80)
          printf("your grade is C.\n");
    if (score >=60 && score < 70)
          printf("your grade is D.\n");
    if (score < 60)
          printf("your grade is F.\n");

    return 0;
}
```

Automatic Type Conversion

- What happens when expression has mixture of different data types.
- Ex)

```
double x=1.2;
float y=0.0;
int i=3;
int j=0;

j=x+i;   /* (temporary copy of)i will be converted to double type
            before ‘+’ operation.
            the value of i in memory is unchanged */
y=x+i;

printf(“j=%d , y=%f\n”,j,y);
```

Automatic Type Conversion

- “lower” types are promoted to “higher” types. The expression itself will have the type of its highest operand. The **type hierarchy is as** follows
  - **`long double`**
  - **`double`**
  - **`float`**
  - **`int`**
  - **`short , char`**

- If either operand is `long double`, convert the other to `long double`
- Otherwise, if either operand is `double`, convert the other to `double`
- Otherwise, if either operand is `float`, convert the other to `float`
- Otherwise, convert `char` and `short` to `int`

Automatic Type Conversion with assignment operator

- Example

```
double x=5.5;
int y=3;

y=x;          /* x will be converted to int type */

x=y;          /* y will be converted to double type */
```

Type casting

- Programmers can enforce type conversion to a variable

```
Ex1)
   double x=3.5;
   double y=2.7;
   double below_point;

   below_point = x*y - (int)(x*y) ;

Ex2)
   double x=3.5;
   printf(“integer number of x = %d\n”,(int)x);
```