Structured Programming - Expressions

Donglong Chen

With Thanks to Dr. Xin Feng and Dr. Haipeng Guo

Outline

- Arithmetic expressions
- Relational expressions (conditions)
- Logical expressions (decisions)
- Special operators

Arithmetic Expressions

- Arithmetic operators
  - Binary operators
    - +, -, *: for all integer and float
      - E.g, a + b; 10 – 4, -5, 2.0 * 10
    - /
      - integer: give the int quotient. E.g., 10 / 3 = 3
      - Float: give the float quotient. E.g., 10.0 / 3 = 3.333333
    - % (modulo)
      - integer: give the remainder. E.g., 5 % 3 = 2
      - not applicable to float

Arithmetic Expressions

- An expression is a sequence of operands (constants or variables) and operators that reduces to a single value
  - E.g., a*b-c, (m+n)*(x+y), 6*2/3
- An expression is evaluated from left to right using the rule of precedence of operators
  - Precedence
    - Highest priority: ( )
    - High priority: * / %
    - Low priority: + -
  - What are the results of x and y
    - x = 9-12/3+3*2-1;
    - y = 9-12/(3+3)*(2-1);

https://en.cppreference.com/w/c/language/operator_precedence

Exercises

Compare the results of y

```
int i = 5, j = 1;
float x = 1.0, y;
y = x / i;
```

```
int i = 5, j = 1;
float x = 1.0, y;
y = j / i;
```

```
int i = 5, j = 1;
float x = 1.0, y;
y = (float)j / i;
```

Exercises

Compare the results of y

```
int i = 5, j = 1;
float x = 1.0, y;
y = x / i;              /* y = 1.0/5 = 0.2  */
```

```
int i = 5, j = 1;
float x = 1.0, y;
y = j / i;              /* y = 1/5 = 0.0 */
```

```
int i = 5, j = 1;
float x = 1.0, y;
y = (float)j / i;       /* y = 1.0/5 = 0.2  */
```

https://www.tutorialspoint.com/cprogramming/c_type_casting.htm

Relational Expressions (Conditions)

- Expressions with relational operators: <, <=, >, >=, ==, !=
- Allows you to compare variables and values
- The value of a relational expression is either 0 (false) or 1 (true)
- 4 < 5: true; 4 > 5: false; 4!=5: true.
- x > 10 : unknown, depending on the value of x

Relational Expressions

Operator	Description	Example
>	greater than	5 > 4
>=	greater than or equal to	mark >= score
<	less than	height < 75
<=	less than or equal to	height <= input
==	equal to	score == mark
!=	not equal to	5 != 4

Logical Expressions (Decisions)

- Comprise relational expressions (conditions) and logical operators
  - Logical operators
    - && (two ampersands): means *and*.
    - || (two vertical bars): means *or*.
    - ! (an exclamation point): means *not*.
- Allows you to verify more than one condition
- A logical expression is also called a decision
  - E.g., (x >= 0.0) && (x <= 1.0)
- When there is no logical operator in a decision, it is a condition.

Exercise

- Translate the following English questions into C decisions.
  - The height is not equal to zero
  - The temperature (variable: temp) is greater than 32.0 and less than 212.0
  - The absolute value of *pos* is greater than 5.0

Assignment Operators

- Assignment operators
  - =
    - E.g., a = 10; c = ‘c’;
  - *op*=
    - *op* can be any of +, -, *, /, and %
    - a *op*= b is equivalent to a = a *op* b
    - E.g.,
      - a += 10 is equivalent to a = a + 10
      - a /= b + c is equivalent to a = a / (b + c)

Examples

- Example 1

```
x = ( y = 3 ) +1;      /* y is assigned 3 */
                       /* the value of (y = 3) is 3 */
                       /* x is assigned 4 */
```

- Example 2

```
x = 5;            /* x is assigned 3 */
y = 3;            /* y is assigned 3 */
x += y + 1;       /* x = x + (y+1) */
                        // x = 9
```

Exercise

- Can you explain these expressions
  - x = (y = 5) + 3
  - x = y = 5 + 3
  - x == (y =5)

Arithmetic Expressions

- Arithmetic operators
  - unary operators
    - -
      - E.g., -5, x = -y
    - ++ (increment), -- (decrement)
      - Has only one operand. Only applicable to integers
      - E.g.,
        - » x++ is equivalent to x = x + 1
        - » ++x is equivalent to x = x + 1
        - » x-- is equivalent to x = x – 1
        - » --x is equivalent to x = x – 1

Rules for ++ and --

- ++ and -- are unary operators and they require variable as their operand
- postfix ++ (or --) (e.g. x++) : if it is used with a variable in an expression, the expression is evaluated first using the original value of the variable and then the variable is incremented (or decremented) by one
- prefix ++ (or --) (e.g. ++x) : if it is used with a variable in an expression, the variable is incremented (or decremented) by one first and then the expression is evaluated using the new value of the variable

Exercise

```
m = 5;
y = ++m;
x = m++;
```

What are the final values of x, y, and m?

Conditional Operators

- Format
  - exp1 ? exp2 : exp3
  - If exp1 is true,
    - Value of exp1 ? exp2 : exp3 is exp2 (exp3 is not evaluated)
  - If exp1 is false,
    - Value of exp1 ? exp2 : exp3 is exp3 (exp2 is not evaluated)

Examples

Example 1

- x = 4;
- y = 5;
- z = (x > y) ? x : y;

Example 2

- x = 5;
- y = 4;
- z = (x > y) ? x : y;

What is the value of z in these two examples?

Bitwise Operators

- Work on binary system of all integer types
  - Operator Meaning

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise Exclusive OR
~	bitwise complement
<<	shift left, zero pad on LSB
>>	shift right, zero pad on MSB

Bitwise Operators

```
x = 0xFFF0;
y = 0x002F;
```

```
x & y = 0x0020

x:      1111  1111  1111 0000
y:      0000 0000 0010 1111
        ---------------------
        0000 0000 0010 0000
```

0	1	0	1
0	0	1	1
0	0	0	1

&

```
x | y = 0xFFFF

x:      1111  1111  1111 0000
y:      0000 0000 0010 1111
        ---------------------
        1111  1111  1111 1111
```

0	1	0	1
0	0	1	1
0	1	1	1

|

Bitwise Operators

x = 0xFFF0;
y = 0x002F;

x ^ y = 0xFFDF

```
x:    1111 1111 1111 0000
y:    0000 0000 0010 1111
      ---------------------
      1111 1111 1101 1111
```

0	1	0	1	^
0	0	1	1	
0	1	1	0	

~y = 0xFFD0

```
y:    0000 0000 0010 1111
      ---------------------
      1111 1111 1101 0000
```

1	0	~
0	1	

Shift, Multiplication and Division

- 14: 0000 1110 ($2^3+2^2+2^1$)
- 14<<1 (shift one bit left: 0001 1100) ($2^4+2^3+2^2 = 28$)
- 14>>1 (shift one bit right: 0000 0111) ($2^2+2^0 = 7$)

Shift, Multiplication and Division

- Multiplication and division are often slower than shift.
- Multiplying 2 can be replaced by shifting 1 bit to the left.

```
n = 10
printf("%d = %d" , n*2, n<<1);
printf("%d = %d", n*4, n<<2);
```

- Division by 2 can be replaced by shifting 1 bit to the right.

```
n = 10
printf("%d = %d" , n/2, n>>1);
printf("%d = %d", n/4, n>>2);
```

Comma Operator

- An expression can be composed of multiple subexpressions separated by commas.
  - Subexpressions are evaluated left to right.
  - The entire expression evaluates to the value of the rightmost subexpression.

Example

```
x = (a++, b++);
```

Evaluation steps:

1. a is incremented
2. b is assigned to x
3. b is incremented

What if the parenthesis are missed?

Operator Precedence

	Operator	Precedence level
–	( )	1
–	~, ++, --, unary -	2
–	*, /, %	3
–	+, -	4
–	<<, >>	5
–	<, <=, >, >=	6
–	==, !=	7
–	&	8
–	^	9
–	|	10
–	&&	11
–	||	12
–	=, +=, -=, etc.	14
–	,	15

Exercise

```
#include <stdio.h>
int main ()
{
   int w = 10,x = 20, y = 30, z = 40;
   int temp1, temp2;

   temp1 = x * x /++y + z / y;
   printf("temp1= %d;\nw= %d;\nx= %d;\ny= %d;\nz=
          %d\n", temp1, w,x,y,z);
   y=30;
   temp2 = x * x /y++ + z / y;
   printf("temp2= %d;\nw= %d;\nx= %d;\ny= %d;\nz=
          %d\n",temp2, w,x,y,z);
   return 0;
}
```

What is the output of the program?

Summary

- Arithmetic expression
- Logical expression
- Bitwise operation
- Precedence of operations