3.x — Chapter 3 comprehensive quiz

BY ALEX ON SEPTEMBER 11TH, 2011 | LAST MODIFIED BY ALEX ON JANUARY 18TH, 2018

Quick review

Always use parentheses to disambiguate the precedence of operators if there is any question or opportunity for confusion.

The arithmetic operators all work like they do in normal mathematics. The Modulus (%) operator returns the remainder from an integer division. Beware about rounding or sign errors when the operands of integer division and modulus are negative.

The increment and decrement operators can be used to easily increment or decrement numbers. Beware of side effects, particularly when it comes to the order that function parameters are evaluated. Do not use a variable that has a side effect applied more than once in a given statement.

Relational operators can be used to compare floating point numbers. Beware using equality and inequality on floating point numbers.

Logical operators allow us to form compound conditional statements. Bitwise operators allow us to modify or query individual bits.

Comprehensive quiz

1) Evaluate the following:

a) (5 > 3 && 4 < 8)
b) (4 > 6 && true)
c) (3 >= 3 || false)
d) (true || false) ? 4 : 5

Hide Solution

a) (5 > 3 && 4 < 8) becomes (true && true), which is true.
b) (4 > 6 && true) becomes (false && true), which is false.
c) (3 >= 3 || false) becomes (true || false), which is true.
d) (true || false) ? 4 : 5 becomes (true ? 4 : 5), which is 4.

2) Answer the following:

a) 7 / 4
b) 14 % 5

Hide Solution

a) 7 / 4 = 1 remainder 3, so this equals 1.
b) 14 % 5 = 2 remainder 4, so this equals 4.

3) Convert the following from binary to decimal:

a) 1101
b) 101110

Hide Solution

a) 1101 is ((1 * 8) + (1 * 4) + (0 * 2) + (1 * 1)) = 8 + 4 + 1 = 13
b) 101110 is ((1 * 32) + (0 * 16) + (1 * 8) + (1 * 4) + (1 * 2) + (0 * 1)) = 32 + 8 + 4 + 2 = 46

4) Convert the following from decimal to binary:

a) 15
b) 53

Hide Solution

a) Using method 1:
15 / 2 = 7 r1
7 / 2 = 3 r1
3 / 2 = 1 r1
1 / 2 = 0 r1

Reading the remainders from bottom to top: 1111

Using method 2:

Is 15 >= 8? Yes, with 7 left over.
Is 7 >= 4? Yes, with 3 left over.
Is 3 >= 2? Yes, with 1 left over.
Is 1 >= 1? Yes.
So this number is 1111 in binary.

b) Using method 1:
53 / 2 = 26 r1
26 / 2 = 13 r0
13 / 2 = 6 r1
6 / 2 = 3 r0
3 / 2 = 1 r1
1 / 2 = 0 r1

Reading the remainders from bottom to top: 110101

Using method 2:

Is 53 >= 32? Yes, with 21 left over.
Is 21 >= 16? Yes, with 5 left over.
Is 5 >= 8? No.
Is 5 >= 4? Yes, with 1 left over.
Is 1 > 2? No.
Is 1 >=1? Yes.

So this number is 110101 in binary.

5) Why should you never do the following:

a) int y = foo(++x, x);
b) int x = 7 / -2; // (prior to C++11)
c) int x = -5 % 2; // (prior to C++11)
d) float x = 0.1 + 0.1; if (x == 0.2) return true; else return false;
e) int x = 3 / 0;

Hide Solution

a) Because operator++ applies a side effect to x, we should not use x again in the same expression. In this case, the parameters to function foo() can be evaluated in any order, so it's indeterminate whether x or ++x gets evaluated first. Because ++x changes the value of x, it's unclear what values will be passed into the function.
b) Prior to C++11, it's unclear whether the compiler will round this up to -3 or down to -4.
c) Prior to C++11, it's unclear whether this will result in 1 or -1 (note: You can still use % 2 for even/odd checking).
d) Floating point precision issues may cause this to evaluate as false even though it looks like it should be true.
e) Divide by 0 will crash the program.

 **4.1 -- Blocks (compound statements)**

 **Index**