UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*

Comparison operators

Douglas Wilhelm Harder, M.Math.
Prof. Hiren Patel, Ph.D.
hiren.patel@uwaterloo.ca dwharder@uwaterloo.ca

Outline

- In this lesson, we will:
  – Review comparison symbols from secondary school mathematics
  – Describe the six binary comparison operators
  – Understand how they differ in purpose
    - They evaluate to `true` or `false` (`1` or `0`)
  – Look at some common errors
  – Upcasting of the operands

Comparison operators

- Previously, we saw that the literals `true` and `false` actually evaluate to the values `1` and `0`, respectively

- We will now look at six comparison operators that compare integers and floating-point numbers
  – From your secondary school mathematics, given two integers or real numbers, you can always compare their values

$$= \qquad \neq \qquad < \qquad \leq \qquad > \qquad \geq$$

  – For example,

$$1.\bar{9} = 2 \qquad \sin(x) \leq 1 \qquad \pi \neq \frac{22}{7}$$

$$x^2 \geq 0 \qquad \pi^e < e^\pi \qquad \frac{10\left(e^\pi - \ln(3)\right)}{\ln(2)} < 318$$

Comparison operators

- In your mathematics courses, you used comparison operators as statements:

  If $x > 2$, then $x^2 > 4$.

- In general, you made statements like "$x < y$ is false."
  – Instead, you would now write $x \nless y$ or $x \geq y$

- You may have even defined the absolute-value as:

$$|x| = \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases}$$

Comparison operators

- In C++, we must make decisions on yes-no, or true-false questions
  - These are collectively called *Boolean-valued queries*
  - C++ has six Boolean-valued operators for comparing items

Comparison Operator	Description
`==`	equal to
`!=`	not equal to
`<`	less than
`<=`	less than or equal to
`>=`	greater than or equal to
`>`	greater than

- Remember, we are restricted to symbols on the keyboard
  - In C++, enunciate '!' as "not" when you say it or think it

Comparison operators

- Each takes two operands, and evaluates to either true or false depending on whether or not the operands satisfy the condition

```
std::cout <<     (3 < 4)      << std::endl;  // prints 1
std::cout <<     (4 < 4)      << std::endl;  // prints 0
std::cout <<     (3 != 4)     << std::endl;  // prints 1
std::cout <<   (3.1 != 3.100) << std::endl;  // prints 0
std::cout << (-42.3 == -42.3) << std::endl;  // prints 1
std::cout <<    (-7 == 7)     << std::endl;  // prints 0
```

Common mistakes

- The most common mistake is to write = when you mean ==
  - The = operator is reserved for assignment (to be covered soon)

- Another very common mistake is to write =< or => when you mean <= or >=, respectively
  - Remember to write it as you say it:

    3 is less than or equal to 4
    3 < = 4

  or 3 <= 4

Common mistakes

- Consider these two mistakes:

```
#include <iostream>

int main();

int main() {
    std::cout << (3 =< 4) << std::endl;
    std::cout << (3 => 4) << std::endl;

    return 0;
}
```

Write it as you say it: 3 is less than or equal to 4

- The error messages can be a little opaque, but you can see the issue:

```
example.cpp: In function 'int main()':
example.cpp:6:22: error: expected primary-expression before '<' token
     std::cout << (3 =< 4) << std::endl;
                      ^
example.cpp:7:22: error: expected primary-expression before '>' token
     std::cout << (3 => 4) << std::endl;
                      ^
```

Complementary operators

- If x and y are different values, note these relationship:

```
 x < y          x >= y
x == y          x != y
 x > y          x <= y
```

- The result in the first column will always be the *opposite* of the result in the second:
  - The opposite of `true` is `false`, and the opposite of `false` is `true`
  - In Boolean logic, we call this *negation*
- The term *complementary* means "that which completes"
  - There is the sine of an angle, but it is only with the complement of the sine—the cosine—that we know where we are on the circle

Sequences of comparison operators

- New programmers may try the following:

```
std::cout << (3 < 4 < 5) << std::endl;
```

- The output is `1`, so was this not successful? Try it with:

```
std::cout << (-5 < -4 < -3) << std::endl;
```

- This fails: why?
  - Just like with `3 + 4 + 5`, the above are evaluated as

```
 (3 < 4) < 5
(-5 < -4) < -3
```

  - In both cases, the first evaluates to `true`, or `1`

```
1 < 5
1 < -3
```

  - The first is `true`, the second is `false`

Upcasting

- If one operand is an integer and the other is a floating-point number, the integer is converted to a floating-point number first:

```
std::cout <<  (3 < 3.0)      << std::endl;  // prints 0
std::cout <<  (3 != 3.00000) << std::endl;  // prints 0
std::cout << (42 == 42.0)    << std::endl;  // prints 1
std::cout <<  (0 <= 0.0)     << std::endl;  // prints 1
```

- Remember that integers are stored differently from floating-point numbers—more on this later
- All integers between $-2^{53} + 1$ and $2^{53} - 1$ can be perfectly represented using a double-precision floating-point number (`double`)
  - The value 2^{53} is approximately 9 quadrillion

Summary

- Following this lesson, you now:
  - Understand the six comparison operators: `==  !=  <  <=  >=  >`
  - Based on the operands, these evaluate to either `true` or `false` (`1` or `0`)
  - You must avoid using `=` when you want to compare the operands
  - You cannot use `=<` or `=>` when you mean `<=` or `>=`
  - If one operand is an integer and the other is a floating-point number, the integer is cast as a floating-point number

References

[1] No references?

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer