

## Topic 2

---

1. The `if` statement
2. Comparing numbers and strings
3. Multiple alternatives
4. Nested branches
5. Problem solving: flowcharts
6. Problem solving: test cases
7. Boolean variables and operators
8. Application: input validation
9. Chapter summary

# Relational Operators: Table 1

C++	Math Notation	Description
>	>	Greater than
>=	≥	Greater than or equal
<	<	Less than
<=	≤	Less than or equal
==	=	Equal
!=	≠	Not equal

*Relational operators are used to compare numbers and strings, inside the () of if().*

## Relational Operator Examples: Table 2 (Part 1)

Expression	Value	Comment
<code>3 &lt;= 4</code>	true	3 is less than 4; <= tests for “less than or equal”.
<code>3 =&lt; 4</code>	<b>Error</b>	The “less than or equal” operator is <=, not =<. The “less than” symbol comes first.
<code>3 &gt; 4</code>	false	> is the opposite of <=.
<code>4 &lt; 4</code>	false	The left-hand side of < must be strictly smaller than the right-hand side.
<code>4 &lt;= 4</code>	true	Both sides are equal; <= tests for “less than or equal”.

## Relational Operator Examples: Table 2 (Part 2)

Expression	Value	Comment
<code>3 == 5-2</code>	true	<code>==</code> tests for equality.
<code>3 != 5-1</code>	true	<code>!=</code> tests for inequality. It is true that 3 is not 5 - 1.
<code>3 = 6 / 2</code>	<b>Error</b>	<b>Use <code>==</code> to test for equality.</b>
<code>1.0 / 3.0 == 0.3333333333</code>	false	Although the values are very close to one another, they are not exactly equal. See Common Error 3.3.
<code>"10" &gt; 5</code>	<b>Error</b>	<b>You cannot compare a string to a number.</b>

# Relational Operators – Some Notes

- The == operator is initially confusing to beginners.
- In C++, = already has a meaning, namely assignment
- The == operator denotes equality testing:

```
floor = 13; // Assign 13 to floor
// Test whether floor equals 13
if (floor == 13)
```

- You can compare strings as well:

```
if (input == "Quit") ...
```

# Common Error – Confusing = and ==

- The C++ language allows the use of = inside tests.
- To understand this, we have to go back in time.
- The creators of C, the predecessor to C++, were very frugal thus C did not have true and false values.
- Instead, they allowed any numeric value inside a condition with this interpretation:
  - 0 denotes false
  - any non-0 value denotes true.
- In C++ you should use the `bool` values `true` and `false`

# Confusing = and ==

- Furthermore, in C and C++ assignments have values.
- The *value* of the assignment expression `floor = 13` is *13*.
- These two features conspire to make a horrible pitfall:

```
if (floor = 13) ...
```

is legal C++.

- The code sets `floor` to 13, and since that value is not zero, the condition of the `if` statement is *always true*.

SO... *Use only == inside tests.*

*Use = outside tests.*

# Kinds of Error Messages

---

- Error messages are fatal.
  - The compiler will not translate a program with one or more errors.
- Warning messages are advisory.
  - The compiler will translate the program, but there is a good chance that the program will not do what you expect it to do.
  - So check the warnings, and fix your code if possible to eliminate the warnings



# Common Error – Exact Comparison of Floating-Point Numbers

- *Roundoff errors*
  - Floating-point numbers have only a limited precision.
  - Calculations can introduce roundoff errors.
  - *Given  $r=2$ ,*

Does  $\quad == 2$  ?

$$\left(\sqrt{r}\right)^2$$

Let's see (by writing code, of course) ...

# Exact Comparison of Floating-Point Yields Unexpected Value

```
double r = sqrt(2.0);
if (r * r == 2)
{
    cout << "sqrt(2) squared is 2" << endl;
}
else
{
    cout << "sqrt(2) squared is not 2 but "
        << setprecision(18) << r * r << endl;
}
```

This program displays:

```
sqrt(2) squared is not 2 but 2.00000000000000000044
```

# How to Compare Floating-Point Numbers

## *Roundoff errors – a solution*

Close enough will do.

$$|x - y| < \varepsilon$$

$\varepsilon$  is the Greek letter epsilon, a letter used to denote a very small quantity

# Comparison of Floating-Point Numbers: Tolerance

- It is common to set  $\varepsilon$  to  $10^{-14}$  when comparing **double** numbers:

```
const double EPSILON = 1E-14;
double r = sqrt(2.0);
if (fabs(r * r - 2) < EPSILON)
{
    cout << "sqrt(2) squared is approximately
";
}
```

- Include the `<cmath>` header to use `sqrt` and the `fabs` function which gives the absolute value.

# Lexicographical Ordering of Strings

- Comparing strings uses “lexicographical” order to decide which is larger or smaller or if two strings are equal.

“Dictionary order”

```
string name = "Tom";  
if (name < "Dick") ...
```

The test is false because “Dick”  
would come before “Tom”  
if they were words in a dictionary.

# Comparing Strings

---

- When comparing two strings, you compare the first letters of each word, then the second letters, and so on, until:
  - one of the strings ends
  - you find the first letter pair that doesn't match.
- If one of the strings ends, the longer string is considered the “larger” one.

# String Comparison Proceeds Letter by Letter

- We compare letter by letter, starting at the left.
- For example, compare "car" with "cart".

c a r

c a r t

- The first three letters match, and we reach the end of the first string – making it less than the second.
- Therefore "car" is before "cart" lexicographically.
- When you reach a mismatch, the string containing the “larger” character is considered “larger”.