3.17 Floating-point comparison

Floating-point numbers should not be compared using ==. Ex: Avoid float1 == float2. Reason: Some floating-point numbers cannot be exactly represented in the limited available memory bits like 64 bits. Floating-point numbers expected to be equal may be close but not exactly equal.

PARTICIPATION ACTIVITY 3.17.1: Floating-point comparisons.

1 2 3 2x speed

```
numMeters = 0.7;
numMeters = numMeters - 0.4;
numMeters = numMeters - 0.3;

// numMeters expected to be 0,
// but is actually 0.00000000000000000555112

if (fabs(numMeters - 0.0) < 0.001) {
   // Equals 0.
}
else {
   // Does not equal 0.
}
```

Expected	Actual
0.7	0.69999999999999999555910790
0.4	0.40000000000000000222044605
0.3	0.29999999999999999888977697
0	-0.00000000000000000555111512

```
if (numMeters == 0.0) {
   // Equals 0.
}
else {
   // Does not equal 0.
}
```

Compare floats for 'close enough'.

Feedback?

Floating-point numbers should be compared for "close enough" rather than exact equality. Ex: If (x - y) < 0.0001, x and y are deemed equal. Because the difference may be negative, the absolute value is used: fabs(x - y) < 0.0001. fabs() is a function in the math library. The difference threshold indicating that floating-point numbers are equal is often called the ***epsilon***. Epsilon's value depends on the program's expected values, but 0.0001 is common.

The std::abs() function is overloaded to support floating-point and integer types. However, good practice is to use the fabs() function to make the operation clear.

PARTICIPATION ACTIVITY 3.17.2: Using == with floating-point numbers.

1) Given: float x, y
   x == y is OK.

   ○ True

Correct

- False

Floating-point numbers should not use == for comparison, due to inexact representation.

2) Given: double x, y
x == y is OK.
- True
- False

Correct

Variables of type double are floating-point numbers, which should not use ==.

3) Given: double x
x == 32.0 is OK.
- True
- False

Correct

x and 32.0 are floating-point numbers, which should not use ==.

4) Given: int x, y
x == y is OK.
- True
- False

Correct

Integer values are represented exactly, so == is OK to use.

5) Given: double x
x == 32 is OK.
- True
- False

Correct

Comparing a floating-point with an integer is very bad practice. The comparison should be with 32.0. And floating-point comparisons should not use ==.

Feedback?

PARTICIPATION ACTIVITY 3.17.3: Floating-point comparisons.

Each comparison has a problem. Click on the problem.

1) fabs (x - y) == 0.0001

Correct

Comparison should be <, not ==. If difference is less than 0.0001, the values are deemed "close enough."

2) fabs (x - y) < 1.0

Correct

While difference thresholds may vary per program, 1.0 is far too large for floating-point inexactness.

Feedback?

PARTICIPATION ACTIVITY 3.17.4: Floating point statements.

Complete the comparison for floating-point numbers.

1) Determine if double variable x is 98.6.

```
fabs (x - 98.6) < 0.0001
```

Check Show answer

Correct

fabs

fabs() computes floating-point absolute value.

2) Determine if double variables x and y are equal. Threshold is 0.0001.

```
fabs(x - y) <0.0001
```

Check Show answer

Correct

< 0.0001

If difference is less than 0.0001, values are close enough.

3) Determine if double variable x is 1.0

```
fabs( x-1.0 ) < 0.0001
```

Check Show answer

Correct

x - 1.0

The subtraction order doesn't matter, because the absolute value is taken.

Feedback?

Figure 3.17.1: Example of comparing floating-point numbers for equality: Body temperature.

```
Enter body temperature in Fahrenheit:
98.6
Temperature is exactly normal.

...

Enter body temperature in Fahrenheit: 90
Temperature is below normal.

...

Enter body temperature in Fahrenheit: 99
Temperature is above normal.
```

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
   double bodyTemp;

   cout << "Enter body temperature in Fahrenheit: ";
   cin >> bodyTemp;

   if (fabs(bodyTemp - 98.6) < 0.0001) {
      cout << "Temperature is exactly normal." <<
endl;
   }
   else if (bodyTemp > 98.6) {
      cout << "Temperature is above normal." << endl;
   }
   else {
      cout << "Temperature is below normal." << endl;
   }

   return 0;
}
```

Feedback?

PARTICIPATION ACTIVITY 3.17.5: Body temperature in Fahrenheit.

Refer to the body temperature code provided in the previous figure.

1) What is output if the user enters 98.6?

- (•) Exactly normal
- ( ) Above normal
- ( ) Below normal

Correct

98.6 - 98.6 will be less than 0.0001, despite any rounding.

2) What is output if the user enters 97.0?

- ( ) Exactly normal
- ( ) Above normal
- (•) Below normal

Correct

The equals branch won't execute because the difference is not less than 0.0001, and the second branch won't either, leaving the third branch to execute.

3) What is output if the user enters 98.6000001?

- (•) Exactly normal
- ( ) Above normal
- ( )

Correct

98.6000001 - 98.6 is 0.0000001, which is less than 0.0001. The comparison for "close enough" isn't a perfect approach. If the programmer really wants to detect such

Below normal	small differences, the programmer would use a smaller epsilon, like 0.000000001.

Feedback?

To see the inexact value stored in a floating-point variable, a manipulator can be used in an output statement. Such output formatting is discussed in another section.

Figure 3.17.2: Observing the inexact values stored in floating-point variables.

```
#include <iostream>
#include <ios>
#include <iomanip>
using namespace std;

int main() {
   double sampleValue1 = 0.2;
   double sampleValue2 = 0.3;
   double sampleValue3 = 0.7;
   double sampleValue4 = 0.0;
   double sampleValue5 = 0.25;


   cout << "sampleValue1 using just cout: "
        << sampleValue1 << endl;

   cout << setprecision(25)
        << "sampleValue1 is " << sampleValue1 <<
endl
        << "sampleValue2 is " << sampleValue2 <<
endl
        << "sampleValue3 is " << sampleValue3 <<
endl
        << "sampleValue4 is " << sampleValue4 <<
endl
        << "sampleValue5 is " << sampleValue5 <<
endl;

   return 0;
}
```

```
sampleValue1 using just cout: 0.2
sampleValue1 is
0.2000000000000000111022302
sampleValue2 is
0.2999999999999999888977698
sampleValue3 is
0.6999999999999999555591079
sampleValue4 is 0
sampleValue5 is 0.25
```

Feedback?

PARTICIPATION ACTIVITY 3.17.6: Inexact representation of floating-point values.

Enter a decimal value:

0.086543567

Convert

0.086543567 in a 32-bit floating-point representation (IEEE):

Sign	Exponent	Mantissa
0	0 1 1 1 1 0 1 1	1. 0 1 1 0 0 0 1 0 0 1 1 1 1
+1	$2^{(123 - 127)} = 2^{-4}$	1.384697079658

Representation's value: 0.08654356747865677

Feedback?

PARTICIPATION ACTIVITY 3.17.7: Representing floating-point numbers.

1) Floating-point values are always stored with some inaccuracy.
   - ○ True
   - ◉ False

Correct

Sometimes a floating-point number can be represented exactly in the finite number of bits, like 0.0, 2.0, or 0.25. Thus, sometimes == will work, but other times will fail, so still should not be used.

2) If a floating-point variable is assigned with 0.2, and prints as 0.2, the value must have been represented exactly.
   - ○ True
   - ◉ False

Correct

Although the value is printed as 0.2, 0.2 is not exactly represented when using a floating-point variable. When using type double, the nearest floating-point value is 0.200000000000000011102230246251565404236316680908203125. Rounding occurs when variables are printed due to the limited number of printed digits.

Feedback?

CHALLENGE ACTIVITY 3.17.1: Floating-point comparison: Print Equal or Not equal.

Write an expression that will cause the following code to print "Equal" if the value of sensorReading is "close enough" to targetValue. Otherwise, print "Not equal". Ex: If targetValue is 0.3333 and sensorReading is (1.0/3.0), output is:

```
Equal
```

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
   double targetValue;
   double sensorReading;

   cin >> targetValue;
   cin >> sensorReading;

   if (fabs(sensorReading-targetValue)<0.0001/* Your solution goes here  */) {
      cout << "Equal" << endl;
   }
   else {
      cout << "Not equal" << endl;
   }

   return 0;
}
```

Run ✓ All tests passed

✓ Testing with targetValue = 0.3333, sensorReading = 1.0/3.0

Your output `Equal`

✓ Testing with targetValue = 0.3333, sensorReading = 2.0/3.0

Your output `Not equal`

✓ Testing with targetValue = 0.6, sensorReading = 0.1 + 0.2 + 0.3

Your output `Equal`

Feedback?