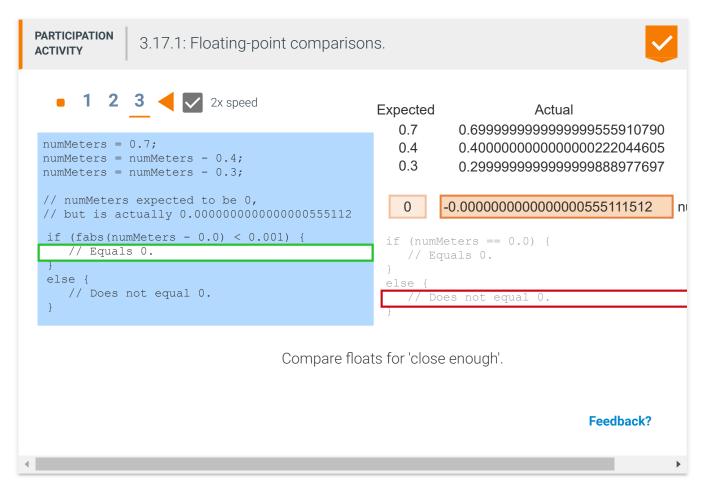
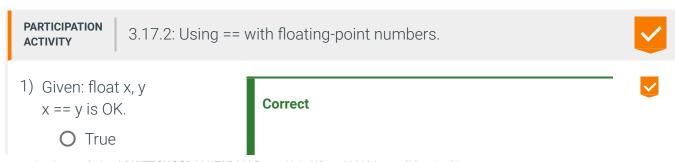
# 3.17 Floating-point comparison

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.



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.



### 3.17. Floating-point comparison

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

False

- O True
- False
- 3) Given: double x x == 32.0 is OK.
  - O True
  - False
- 4) Given: int x, y x == y is OK.
  - True
  - O False
- 5) Given: double x x == 32 is OK.
  - O True
  - False

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

### Correct

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

# Correct

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

## Correct

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

### 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 ==.

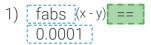
Feedback?

# PARTICIPATION ACTIVITY

3.17.3: Floating-point comparisons.

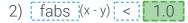


Each comparison has a problem. Click on the problem.



# Correct

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



### Correct

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

Feedback?

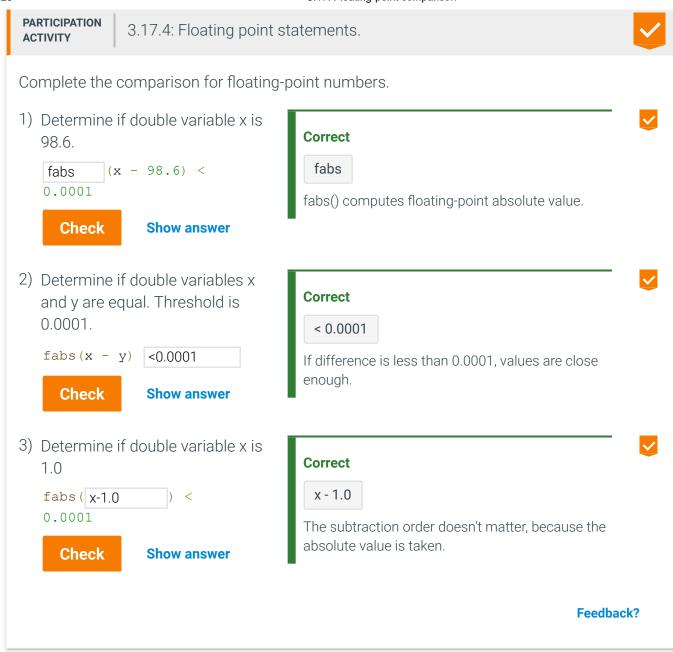


Figure 3.17.1: Example of comparing floating-point numbers for equality: Body temperature.

```
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: ";</pre>
   cin >> bodyTemp;
   if (fabs(bodyTemp - 98.6) < 0.0001) {</pre>
      cout << "Temperature is exactly normal." <<</pre>
end1;
   else if (bodyTemp > 98.6) {
      cout << "Temperature is above normal." << endl;</pre>
   else {
      cout << "Temperature is below normal." << endl;</pre>
   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
  - O Above normal
  - O Below normal
- 2) What is output if the user enters 97.0?
  - O Exactly normal
  - O Above normal
  - Below normal
- 3) What is output if the user enters 98.6000001?
  - Exactly normal
  - O Above normal

O

### Correct

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

### 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.

#### 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: "</pre>
        << sampleValue1 << endl;</pre>
   cout << setprecision(25)</pre>
        << "sampleValue1 is " << sampleValue1 <<</pre>
endl
        << "sampleValue2 is " << sampleValue2 <<</pre>
endl
        << "sampleValue3 is " << sampleValue3 <<</pre>
end1
        << "sampleValue4 is " << sampleValue4 <<</pre>
endl
        << "sampleValue5 is " << sampleValue5 <<</pre>
endl;
   return 0;
```

```
sampleValue1 using just cout: 0.2
sampleValue1 is
0.20000000000000000111022302
sampleValue2 is
0.29999999999999999888977698
sampleValue3 is
0.69999999999999955591079
sampleValue4 is 0
sampleValue5 is 0.25
```

Feedback?

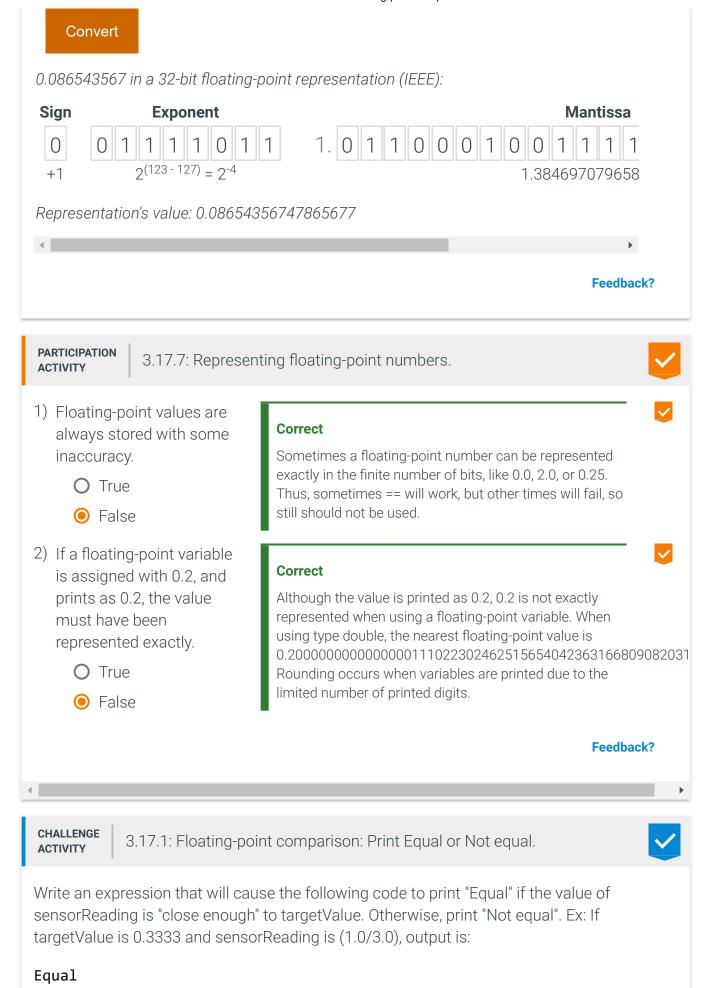
PARTICIPATION ACTIVITY

3.17.6: Inexact representation of floating-point values.



Enter a decimal value:

0.086543567



```
1 #include <iostream>
   2 #include <cmath>
   3 using namespace std;
   5 int main() {
         double targetValue;
         double sensorReading;
   7
   8
   9
         cin >> targetValue;
  10
         cin >> sensorReading;
  11
         if (fabs(sensorReading-targetValue)<0.0001/* Your solution goes here */) {</pre>
  12
            cout << "Equal" << endl;</pre>
  13
  14
  15
         else {
            cout << "Not equal" << endl;</pre>
  16
  17
  18
  19
         return 0;
  20 }
           ✓ All tests passed
  Run

✓ 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?
```