Assignment 01 (part 4): Data types and arithmetic

Complete each of the problems below. Any code you write should be added to the repository and checked in using the filename convention hw1pX.cpp where X corresponds to the number below.

Include in your README.md file any explanations, as necessary. You can add sections using '#' or subsections with '##' in the Markdown file.

Points for each problem will be awarded as outlined on Canvas. No credit is given if your code does not compile on the CCV environment.

1) The floating point precision of the **cout** stream can be adjusted by the method precision as follows,

```
double x =0.14580858;
cout.precision(5);
cout << x << endl;
```

Write a simple program with a **for** loop, starting with 1 significant digit, that shows the best precision you can achieve for a **float** and **double** real types. Explain why this precision drops off where it does for each type.

2) Given the C++ types of signed and unsigned integers determine by programming what happens when each combination of signed -> unsigned casts are performed, e.g. **int -> unsigned short**, **long -> unsigned short**. Interpret what is happening as a general rule. Do not bother with the **long long** type.

3) Determine the value of the 32 bit floating point number **0x3F400000**. Show the hand calculation steps for how you determined this value.

4) Experiment with the comparison operations, by considering cross-type comparisons, e.g. **float** with **int**. What happens during the comparison of mixed types?

5) In comparing two real **double** values for equality, due to round off and precision limits, two ideally equal quantities may not be exactly equal. Devise a test that will produce a correct decision when the values are essentially equal. What does essentially equal mean?

6) The following code is one possible implementation to test a variable **v**,

```
#include <math.h>
double v;  //Some value around which we wish to test for equality
// ilogb gives the unbiased exponent of a double value expressed
// as a power of 2.
int vpow = ilogb(v)-52;
double eps = ldexp(1.0,vpow);  //ldexp returns 1.0*2^vpow
```

eps provides a tolerance on the test for equality. Implement this test and experiment with equality of different ranges of double values and see if **eps** is a reasonable value. Is the computation of **eps** a reasonable approach to testing for equality. Why or why not?