Fixed Point - Task

Phase 1: Price

The Huge Company

In our dreams, we are the CEO of a huge company, and we are planning the budget for the coming years. In the current year, the income was exactly $100,000,000.00, and according to our analysis, the increase in population and the popularity of our products will increase the income by $1.00 per year. When the income is greater than $100,000,010.00, it will be the right time for a nice exit, and we are anxious to know how many years we will have to wait for that to happen.

We are great in business, but bad in math, so we must calculate the expected time to get there. Luckily, we know programming, so the solution is very easy -

```
float incomeCurrent = 100000000.00f;
float incomeForExit = 100000010.00f;
unsigned int years = 0;
for (float income = incomeCurrent; income <= incomeForExit; ++income)
    ++years;
std::cout << "Wait " << years << "for exit!" <<  std::endl;
```

That's great!
We compile and run that code and wait to see how much time we should wait.
Can you tell the number of years you will have to wait?
Try it yourself!

You will wait a **long** time...

A business like ours cannot use floats, you think. A double will do. Yes, it will take twice the memory, but at least it will allow us retiring as multi-millionaires in a few years.

Will it? Give it a test.

The Grocery

Then, you wake up. No huge company. Not even a medium one. All you have is a small neighborhood grocery, and your monthly income doubtedly requires eight bytes of RAM...

So you get back to the floats. Any bun in your grocery costs one dollar and ten cents, so it makes sense to set their price to the float value:

```
float bunPrice = 1.1f;
```

Today is your lucky day, and a customer from next door bootcamp comes to buy some buns to their happy hour. The customer says that he would like to buy buns in a budget of $22.00. You quickly do the math:

```
float price = 1.1f;
unsigned int bunsNum = 0;
for (float budget = 22.0f; budget != 0; budget -= price)
    ++bunsNum;
```

So, how many buns can you sell for 22 dollars? You should probably calculate it yourself with that code.

Damn floats. Not only your imaginary exit did they ruin - now you cannot even sell a few bunch with float price. This is an issue for doubles, you think. Those fat guys that take no less than 64 bits from our RAM. So you change the floats to doubles, but…
It just doesn't work.

It seems that money and floating points do not work together. What can you do?

One solution might be to use integers, but our buns' price is smaller than a whole dollar. How can we deal with it? What about holding the price as integer - but instead of holding the number of dollars, we can hold the number of *cents*.
Now, the price of a bun is 110, the budget of our customer is 2200, and all the calculations can easily success.

Fixed Point

This approach is called **fixed point**. It is very common under some circumstances, for example, the one we have just described, or on processors that do not have any floating-point units.

So let's go to work: Create your *Price* class. The class will represent a price, and should:

- Hold a member of type int.
- Its default value is 0.00.
- If initialized by an integer, this is the "dollar" part.
- If initialized by two integers, the first is the "dollar" and the second is the "cent".
- Support all the arithmetic operators (+, -, *, /, %, +=, -=, *=, /=, %=, unary -, postfix and prefix ++, --).
- Support all comparison operators (==, !=, <, >, <=, >=) .
- Support assignment from int.
- Support casting to double.
- Support streaming out to std::cout.

Do we want "explicit" ctor(s)?
Should we implement the binary operators as members or non-members?
Why don't we want assignment from double?
What will happen if we try to assign 3.14 into a price object?

Do not forget that *implement it* also means *test it*!