CS112 - Spring 2024

Lab09

Instructor: Paul Haskell

INTRODUCTION

In this lab, you will work with Java math, both the integer and floating point data types. There are a number of small programs, then a bigger one.

ShortToText

Your next program, **ShortToText.java**, will give you some experience manipulating the bits in the `short` data type. Your program should read a signed short value from the first command line argument and shall print out 16 "0"s and "1"s corresponding to the bit values in the short. The first "0" or "1" printed shall be the "most significant" bit (the one representing -32768) and the last "0" or "1" printed shall be the "least significant" bit (the one representing 1).

TextToShort

TextToShort.java should implement the reverse operation: read in a sequence of "0" and "1" characters from the first command line argument, convert them to a short value, and print out the short value.

- If the program reads fewer than 16 characters, assume the characters are the least significant bits and that any more significant bits are all 0.
- If the program reads more than 16 characters (or 0 characters, or there is no command line argument), print an ERROR message to `System.err`.
- Ignore any commas or spaces in the input. People might put these characters in the input to improve human readability.

It would be smart to use these two programs to test each other.

NumberTable

Now you can reuse your short-to-text code, along with the `printf()` capabilities we learned about, to print a table of numbers in different bases. In a program called **NumberTable.java** please print the numbers from 100 through 255 in: decimal, hexadecimal, and binary. Your output should look something like:

100 64 01100100

101 65 01100101

...

Note that for this program we only print 8 bits for the binary representation, not 16.

Poorly Conditioned Problem

There is no Java programming for this problem. Suppose you have:

1000x + y = 1001

998x + y = 999

What is the solution? Put your solution and all your intermediate work into **results.txt** .

Due to a measurement error, you accidentally have the very similar system

1000x + y = 1001

999x + y = 999

What is the solution now? Put your solution and intermediate work into **results.txt** . Scary, huh?

SmallSums

This problem is pretty simple. In a program called **SmallSums.java**, define a `float` constant and set it to the value 0.000001. Next, define a `float sum` variable, initialized to 0.0. Add the previous constant value to `sum` one million times. What do you get? Print out the result.

In the same program, do the same thing as above with `double` variables. Again, print out what you get.

Your program simply should print out two numbers, on two separate lines.

BigSums

Here is another pretty simple problem. In a program **BigSums.java**

- Initialize a `double` variable `d1` to 1024
- Set `d2` = `d1`;
- Set `d3` to `d2` + 1.0;
- Check (`d3 - d2`). Got the right answer?

Now do `d2 *= d1` and again set `d3` to `d2 + 1.0` . Still get the desired result?

Print out how many times you must multiply `d1` together when setting `d2` (and `d3`) until you get an incorrect result.

Now do the same thing using `float` instead of `double`. Now how many times must you multiply `f1` by itself to get an incorrect result?

Your program simply should print two numbers, on two separate lines.

SimpleFloatNum

Ok, now for something more difficult: you will create a class that models a floating point number, and a program called **SimpleFloatNum.java** that uses it. The bad news is that it personally took me 5 hours to complete this assignment. The good news is that I'm giving you most of the code!

SimpleFloatNum.java in the CourseInfo repository's **Lab09** directory already contains most of the code you need. The file contains two Java classes: `class SimpleFloatNum` is a "tester" class, with a `main()` method that you will fill in. `class SimpleFloat` will implement a simple floating point number and some basic math operations.

Let's start discussing `SimpleFloat`. This class represents a floating point number with:

- a flag '`isNegative`' that describes whether the `SimpleFloat` value is negative or not
- an `exponent`
- a `mantissa`, which always should be a positive number

The numerical value of `SimpleFloat` is:

mantissa x 2^{exponent}

and of course you must handle `isNegative` as part of the value calculation! `SimpleFloat` won't worry about Infinity or NaN. It will represent the value 0.0 with a mantissa of 0 and any value for the exponent.

The mantissa and exponents are represented with `short` values.

There are several constructors provided for `SimpleFloat` e.g:

```
SimpleFloat(short mantissa, short exponent)
SimpleFloat(short mantissa) // exponent = 0
SimpleFloat(double value)
```

There are two other helper functions provided also:

`ConstructorWorker()` // does some of the work for constructors

`normalize()` // sets values for mantissa and exponent to maximize numerical precision

Now let's discuss your work for the lab. You will add the bodies for three member functions:

```
String toString()
double value()
SimpleFloat addTo(SimpleFloat x)
```

The `toString()` method should output a `String` with the following format:

"<<mantissa>> x pow(2, <<exponent>>)"

For example, if the mantissa == 3 and the exponent == -1, then `toString()` should print

3 x pow(2, -1)

The `value()` method should return the value of this `SimpleFloat` as a double. How to calculate this was hinted at above.

Finally, the tricky part: you must implement the `addTo()` method. There are a handful of issues you will have to discover and solve. Recall that when we add numbers, we must convert their representations so that the exponents match—that might be your first task. Also, there are special cases to handle. The **SimpleFloatNum.java** file gives some hints for these.

Now how do you exercise all this code? Please have the **SimpleFloatNum** class's `main()` method read three double values from its first three command line arguments. The program should create three `SimpleFloat`s constructed from those doubles, use your `addTo()` method (twice) to add the three together, and then print out, on two successive lines,

```
finalResult.toString()
finalResult.value()
```

As usual, send an ERROR message to `System.err` if there are problems with the user input.

When you test your code, it should be pretty easy to find cases where the relatively low number of bits in your `SimpleFloat` gives results with less-than-perfect accuracy. For example

% java SimpleFloatNum 10000 0 0.6

% java SimpleFloatNum 1000000 1000000 1000000

Reminder

Put all your files in your **Lab09** directory and push to GitHub before the deadline. The first 6 parts of this assignment must be turned in before Tuesday Feb 27 at 11:59pm.

SimpleFloatNum.java must be turned in before Friday March 1 at 11:59pm

Conclusion

You should be proud to have gotten through one of the trickiest labs in the class, and to have experimented with some fairly advanced technology for mathematical computation.

Grading Rubric

ShortToText.java is worth 15 points total: 2 points each for 5 test cases, and 0-5 points based on code quality.

TextToShort.java is worth 10 points total: 2 points each for 5 test cases.

NumberTable.java is worth 10 points, if the output is correct.

results.txt is worth 0-10 points, depending on the accuracy of your solutions and the completeness of your work.

SmallSums.java is worth 10 points, 5 points for each of the values. The program must do the right thing also, not just print out numbers.

BigSums.java is worth 10 points, 5 points for each of the values. The program must do the right thing also, not just print out numbers.

SimpleFloatNum.java is worth 40 points: 4 points each for 10 different test cases. As usual, at least a few of the test cases will be of your error handling.