CSC 250 – Project #1
Fall 2014

Goals: Gain experience with loops (for, do, while), selection statements, and the static math methods available to your program(s).

Part 1: Project Summary (Due: 5pm Sept. 29th, 2014; Drop dead date: 5pm Sept. 30th, 2014)

Build a simplistic calculator that reads input from an input file and processes the "commands" provided.

Part 2: Processing

Open an input file named 'calculator.txt' (note the case and spelling!) which will contain a series of data for you to process with your simplistic calculator. Process the "*commands*" on each line and print their result to the output screen (standard output).

The input file's first line will contain an integer value (the "*command count*"), which indicates how many subsequent lines (or "*commands*") you should process. Note that the subsequent number of lines may be greater than the number provided, and if so, you should ignore any extra input lines in the input file. (E.g. if the first line contains the value 8, and there are 10 lines that follow it, only the first 8 lines should be processed and the additional two lines should be ignored and discarded without any messages, indications.)

Each line will start with a character or string token that is the name of the "*command*" to perform. Following the "*command*" is any relevant data needed to perform the requested "*operation*". Data following the command is separated by a single space character. A summary of every possible "*command*", and its operators (if applicable) is given below.

For each "*command*", output the result in the following form:

- first, echo the "*command*" and the data as it appears in the input file, and then
- follow the "*command*" echo with an equals sign, and then
- print the result and an new line character following the equals sign.

Erroneous "*commands*" may be placed in the input file. If you encounter any, ignore them and their data and continue processing the input file. Erroneous "*commands*" should be counted toward the "*command count*". No output should be produced for erroneous commands.

The default output precision for calculation results (except for the **round** command) will be shown by floating point values of 3 decimal places (following the decimal point). However, this value can be adjusted by using the **precision** "*command*". The precision command can have an integer parameter of 0 (zero) or greater. If the precision is set to 0 (zero), the result is shown as an integer value (and the resulting value should be truncated).

Example input file:

```
9
sin 15.0
precision 4
sqrt 46.665
min 3 89 680 45
n! 15
precision 2
+ 4.0 5.0
-  5.0 4.0
cube 3
* 1.0 -2.0
+ 4.0 5.0
```

Example output:

```
sin 15.0 = 0.258
sqrt 46.665 = 6.8312
min 3 89 680 45 = 45.0000
n! 15 = 1307674368000.0000
+ 4.0 5.0 = 9.00
-  5.0 4.0 = 1.00
cube 3 = 27.00
```

Commands:

Each of the following commands has the syntax that follows. Each command may be present one or more times in the input file. All commands produce output as described above, except for the **precision** command, which only has the effect of setting an internal value in the application.

Command	Description
+ a b	Adds together the values of 'a' and 'b' – treat this operation as a double calculation to ensure proper calculation of a double result (so that it is not truncated). The values of 'a' and 'b' will be provided as doubles.
- a b	Subtracts the value of 'b' from 'a' – treat this operation as a double calculation to ensure proper calculation of a double result (so that it is not truncated). The values of 'a' and 'b' will be provided as doubles.
* a b	Multiplies together the values of 'a' and 'b' – treat this operation as a double calculation to ensure proper calculation of a double result (so that it is not truncated). The values of 'a' and 'b' will be provided as doubles.
/ a b	Divides the value of 'b' into 'a' (e.g. a/b) – treat this operation as a double calculation to ensure proper calculation of a double result (so that it is not truncated). The values of 'a' and 'b' will be provided as doubles.
% a b	Calculates the value of 'a' modulo 'b'. The values of 'a' and 'b' will be provided as ints.
cube x	Calculates the cubed value of 'x', where 'x' is given as a double value. The value of 'x' will be provided as a double value, provides the result as a double value.
sin x	Calculates the sine of 'x', where 'x' is given as a double value (not radians), provides the result as a double value.

Command	Description
cos x	Calculates the cosine of ‘x’, where ‘x’ is given as a double value (not radians), provides the result as a double value.
tan x	Calculates the tangent of ‘x’, where ‘x’ is given as a double value (not radians), provides the result as a double value.
sqrt x	Calculates the square root of ‘x’, where ‘x’ is given as a double value, provides the result as a double value.
n! x	Calculates the factorial of ‘x’ (e.g. x!), a non-negative integer value. Output of this command should be an integer value, not subjected to precision formatting.
pow x y	Calculates the value of ‘x’ raised to the ‘y’ power. The values of ‘x’ and ‘y’ will be provided as doubles.
min n x1…xn	Calculates the minimum value of n values (x1, x2, x3, …, xn). The values of ‘n’, and all ‘x’ values will be provided as integers. Output of this command should be an integer value, not subjected to precision formatting.
max n x1…xn	Calculates the maximum value of n values (x1, x2, x3, …, xn). The values of ‘n’, and all ‘x’ values will be provided as integers. Output of this command should be an integer value, not subjected to precision formatting.
precision x	Sets the precision of the calculator to ‘x’ places of precision for all subsequent calculations. Results in no output produced. The value of ‘x’ will be provided as an integer.
round x	Returns the closest integer value to the value of ‘x’. The value of ‘x’ will be provided as a double. Output of this command should be an integer value, not subjected to precision formatting.
exp x	Returns Euler's number *e* raised to the power of 'x'. The value of 'x' will be provided as a double. Output of this command should be a double value.
cuberoot x	Returns the cube root of the value of 'x', provided as a double. Output of this command should be an double value.

Part 3: Specifics for the delivery of this project:

The task of organizing and creating the jar for each deliverable will be up to you and your partner. You should always check your work prior to delivering the jar files, as this file is what I will be grading. You want to ensure that only the files I request are contained in the jar file.

You may submit the project any number of times until the due date. Only the last submission will be downloaded, examined, and graded by me. Late points will be deducted for projects turned in starting 1 minute after the due date (that’s why they are called due dates!) Following the drop-dead date for a project, no solutions will be graded for the project. At such a point in time, you will receive a zero grade for the work if you have not uploaded a solution to Canvas.

Part 4: Advice and Disclaimer

Plan your schoolwork and life accordingly. Many students don’t adequately plan their work schedules and attempt to finish programs at the last minute. Doing so usually introduces bugs/problems into the solution. Consider how much time you will require to adequately test your solution for boundary cases and bugs. Failure to plan on your part does not constitute an emergency on my part!

I do not give individual extensions for projects, and rarely give class extensions for work. You have nearly a week and a half to complete this project. I suggest you start your design as soon as possible. Consider what you need to build to solve the problem at hand. Programming involves designing a solution, implementing your solution, and testing your solution.

Part 5: Do You Need Help?

Are you stuck? Ask yourself if you have planned the design of your project. Have you asked questions in class? Did you attend the lectures, and labs? Have you come to see me? Like a textbook or web site, I am a resource that I expect you to use throughout the course of the semester. If you are stuck with your project, I strongly suggest you make an appointment to come see me in person. Generally, I can offer some guidance or help to get you back on the path.

When you come to see me, plan on bringing your source code to me on a memory stick or a laptop. Do not email me your source code, I do not provide email-based help for coursework. I prefer that you make time to visit me in my office where we can sit down and discuss your source code together. I can ask you questions and you can tell me why you coded something the way you did. We can use my whiteboard to draw large diagrams and talk about good solutions and designs. This is not something that can be done easily through email.

Part 6: Code Commenting and Formatting (Indenting) Requirements

Please refer to the separate document provided in Canvas for the details of the commenting and formatting guidelines.

Creating a jar file:

The '`jar`' command is installed by default with the Java compiler (javac). Thus, if you can successfully compile your program, you should be able to jar your files together. A simplified syntax of the '`jar`' command is as follows:

```
Usage: jar {ctx}[vf] [jar-file] files ...
Options:
    -c  create new archive
    -t  list table of contents for archive
    -x  extract named (or all) files from archive
    -v  generate verbose output on standard output
    -f  specify archive file name
```

To create a jar file named '`depasquale.jar`' that contains all of the Java source code files in the current directory, I would perform the following command:

```
jar –cvf depasquale.jar *.java
```

(Note: this process works both in UNIX and Windows via the command line. Some program development environments support the creation of jar files, but until you understand how to create then by hand (and test their creation), I suggest you stick with the command line to create jar files.)

Viewing the contents of a jar file:

To view the contents of a jar file named '`depasquale.jar`', I would perform the following command:

```
jar –tvf depasquale.jar
```

Extracting the contents of a jar file:

To extract the contents of a jar file named '`depasquale.jar`' into the current working directory, I would perform the following command:

```
jar –xvf depasquale.jar
```