CS 133 A

Group Project 4: "RPN Programmable Calculator"

Group name:

Names:_____

RPN stands for "Reverse Polish Notation." This type of calculator works very well for computer science and engineering problems. It uses post-fix notation instead of in-fix notation--that is, operations are written after the numbers operated on. For example, in in-fix notation, adding two numbers, then subtracting a third number might be written as "4 + 5 – 6". In post-fix notation, the same expression could be written as " 4 5 + 6 -" or " 4 5 6 - +".

Your assignment is to write an Object-oriented program in C++ to run an RPN calculator simulator. To do this, you should form a team of three-to-five members. You will need to turn in only one copy of the project for you group. Your group will need to define the following class with these minimum requirements:

class CRPNCalc *(some starting files are on Canvas)*

- should at least have private or protected properties:
  - double **m_registers[NUMREGS]** -- Numregs should be 10 -- the calculator's storage registers
  - string **m_buffer** -- a buffer for input
  - deque<double> **m_stack**-- the input stack. Represented as a deque to allow stack rotation.
  - list<string> **m_program** -- a list of program instructions for the one current program. Programs can be entered directly in program mode or loaded from file.
  - istringstream **m_instrStream** -- stream for inputting. Can be set from console input or from **m_program** elements.
  - bool **m_error** -- set when an error occurs. Cleared in print() after the "<<error>>" message is printed.
  - bool **m_helpOn** -- toggled on or off by the "H" key. When on, print displays a menu of key strokes.
  - bool **m_on** -- whether the calculator is turned on or not. run() exits when this is false.

- Member methods should include:
  - a *constructor* -- has one bool parameter for whether or not the calculator is turned on (*defaults to true*). Initializes properties that need initializing, sets all of the registers to 0.0, and if **m_on**, calls *run()*.
  - void *run()* -- sets the calculator running. Loops while **m_on**.
  - void *print(ostream& ostr)* -- prints identifying line, the help menu if **m_helpOn**, the element at the top of the stack if it is not empty, and "<<error>>" if **m_error** (*then sets **m_error** off*).
  - void *input(istream& istr)* -- copies input into **m_buffer**, sets up **m_instrStream** with it, and calls *parse()*
  - *void parse()* -- extracts and executes the commands that are in **m_instrStream**. ***note:*** *The sample executable allows more than one "token" (number or operator) in a line. While this is nice for the user, it makes coding the parse method much more difficult. You **may** code the parse method so that only one token per line is allowed (therefore, the user would need to type in "2<cr>3<cr>+<cr>" in order to add 2 and 3, rather than just "2 3+<cr>"). If you wish to try to allow for multiple tokens, five points of extra credit is available.*

  methods to perform the required operations for each of the following inputs:
  - **+** -- addition
  - **-** -- subtraction -- not to be confused with negation, as in "-3"
  - ***** -- multiplication
  - / -- division
  - ^ -- exponentiation -- *e.g.* 4 5^ would be 4 to the 5th power (4^5).
  - **%**-- mod -- since the data type is double, this is fmod()
  - **C** -- Clear -- removes all elements from the stack

- **CE** -- Clear Entry -- removes the top element from the stack
- **D** -- rotate the stack Down -- remove the top element and place it at the bottom.
- **F** -- save program to File -- asks the user for file name; saves the program list string-by-string to the file.
- **G0 – G9** -- Gets the value from **m_registers[n]** and places it at the top of the stack
- **H** -- toggles on/off **m_helpOn.**
- **L** -- Load program from file – asks the user for a file name and loads the program list string-by string from the file.
- **M** -- Multiplies the top element of the stack by -1.
- **P** -- starts recording a Program after clearing any existing program. A "P" typed in during this process stops the recording. Each line is inserted as an element into **m_program**.
- **R** -- Runs the program stored in **m_program**.
- **S0 – S9** -- Sets the value in **m_registers[n]** to the value at the top of the stack
- **U** -- rotate the stack Up -- remove the bottom element and place it at the top.
- **X** -- Exit run(), setting **m_on** to false.

Any method encountering an error (*such as trying to do a binary operation without having at least two elements in the stack*) should make sure the program state is the way it was before starting the attempt and should set **m_error** to true.

Also, since all binary operations will need to get two numbers from the stack, all unary operations will need to get 1 number off the stack, etc, it would be wise to have methods like *binary_prep(double& d1, double& d2)* and *unary_prep(double& d)* to avoid re-writing code over and over again.

You will also need to define:

- ostream &operator <<(ostream &ostr, const CRPNCalc &calc);
- istream &operator >>(istream &istr, CRPNCalc &calc);

Once your calculator is working, use it to write, run and save an RPN program which goes through the numbers in registers 0 - 6, puts the sum in register 7, the mean (*average*) in register 8, and the standard deviation in register 9.

> **Standard deviation** is defined as the **square root** of the **variance** of a set of data, where the variance is:
> (sum_of_the_squares - (sum2 / number_of_elements)) / (number_of_elements - 1)
> (*as a quick test, if the data are [5 | 7 | 3 | 6 | 6 | 2 | 5], the standard deviation should be 1.772810520855837*).
> Include this saved program file with your project.

The program should be fully planned in advance, using object-oriented design. It should be well documented, work correctly, and be fully tested.

Make sure to test your program using your saved "standard deviation program".
Possible: 100 points

Extra Credit: If you wish, you may expand the parse method to allow for multiple tokens within the same line.
Possible: 5 points

Deliverables: (one set per group)

Physical:

The Project should be turned in inside a clear plastic file folder. This folder should have a simple flap to hold paper in place--NO buttons, strings, Velcro, etc. Pages should be in order, not stapled.

- Assignment Sheet (printed pdf from the web), with your name written on it, as a cover sheet.
- Printed Source Code with Comments *(including heading blocks -- a file header for each file plus a function header for each function. Describe parameters, any input or output, etc., no line wrapping). Print in portrait mode, 10 - 12 point font.*

Electronic:

- All .h, .cpp, .exe(Release Version) zipped together. Do not use rar or any archive format other than zip. Rename the file: "<GroupName>_p4.zip".
- Sample Output (as .rtf -- run the program, copy the window using <*Alt*|*PrtScn*>, paste into Paint, invert colors *(<Ctrl*|*Shift*|*I*>), copy, open Wordpad, save.)
- A simple test plan including explanations of any discrepancies and reasons for each test. Show actual input and ALL values output as well as ALL expected output. Test each possible action. Save as .xls, xlsx, .doc or .docx file

Submit this single zip file by going to canvas, select this class, select the Assignment tab on the left, select Assignment 4, select the submission tab at the top right, find the file, and Submit.

Due: Monday, June 13, 2016 9:30 a.m.*(beginning of class)*
Groups will give a short presentation of their project at this time.