Physics 750: Exercise 11

Tuesday, October 24, 2017

Data transferred over a UNIX stream is encoded as a sequence of characters (bytes) terminated by the end-of-file (EOF) marker. The C++ objects `cin` and `cout` provide a convenient interface to the terminal's standard input and standard output streams. The operators `>>` and `<<` are used to chain variables, literal expressions, and manipulators to `cin` and `cout`. The resulting behaviour is context-sensitive. The statement `cin >> x` causes characters (up to the next block of white space or the last meaningful character) to be read from the stream and interpreted according to whichever type the variable `x` was declared as. If the input stream has been exhausted (i.e., the EOF has been reached) or if the text read in is incompatible with the presumed type (e.g., we can understand `"123"` as an `int`, but not `"abc"`), then the method `cin.good()` will evaluate to `false`. For example, from a stream containing the data "`Only $5.99?[EOF]`," we can read five characters, a floating point number, and another character before reaching the end:

```
char c1, c2;
double price;
cin >> c1 >> c1 >> c1 >> c1 >> c1 >> price >> c2;
assert( cin.good() );
assert( c1 == '$' );
assert( c2 == '?' );
assert( price > 5.98 and price < 6.00 );
cin >> c2;
assert( !cin.good() );
```

Conversely, the statement `cout << x` causes the variable `x` to be translated into text (again, according to the type of `x`) and dumped to the standard output stream. The form of the output can be controlled using either `cout`'s methods or its stream manipulators.

1. Use the `curl` command to download from the class website everything you'll need for this exercise.

   ```
   $ WEBPATH=http://www.phy.olemiss.edu/~kbeach/
   $ curl $WEBPATH/courses/fall2017/phys750/src/exercise11.tgz -O
   $ tar xzf exercise11.tgz
   $ cd exercise11
   ```

2. A program file `table.cpp` and a data file `elements.dat` are included in the `Lab3` directory. The data file contains a list of the elements from H to Xe in order of increasing atomic weight. The program `table` reads each word it encounters and simply dumps it to `cout` on a line of its own. It keeps doing this until it encounters an EOF.

   ```
   $ make table
   g++ -o table table.cpp -O2 -ansi -pedantic -Wall
   $ cat elements.dat
   H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar K Ca Sc Ti V Cr Mn Fe Co Ni Cu
   Zn Ga Ge As Se Br Kr Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cd In Sn Sb Te I Xe
   $ ./table < elements.dat
   H
   He
   Li
   Be
   .
   .
   .
   Xe
   ```

Modify the program (i.e., edit the `table.cpp` code in `emacs` or `gedit` and then `make table` again) so that each element is immediately followed by its atomic number and is grouped in the same row with the other elements that share its closed-shell structure. Remember that the 1s orbital holds 2 electrons, the $2s2p$ orbitals hold 8, the $3s3p$ hold another 8, the $4s3d4p$ orbitals hold 18, and the $5s4d5p$ hold another 18.

```
$ ./table < elements.dat
H1 He2
Li3 Be4 B5 C6 N7 O8 F9 Ne10
Na11 Mg12 Al13 Si14 P15 S16 Cl17 Ar18
K19 Ca20 Sc21 Ti22 V23 Cr24 Mn25 Fe26 Co27 Ni28 Cu29 Zn30 Ga31 Ge32 As33 Se34 Br35 Kr36
Rb37 Sr38 Y39 Zr40 Nb41 Mo42 Tc43 Ru44 Rh45 Pd46 Ag47 Cd48 In49 Sn50 Sb51 Te52 I53 Xe54
```

3. Now consider the program file `read.cpp` and the data file `resistivity.dat`. The data file contains a record of (fictitious) electrical resistivity measurements performed on various elements. The convention adopted here is that the element name is enclosed in parentheses and the corresponding set of measurements is given as a comma-separated list enclosed in braces. If you compile and run the program `read`, you'll find that it extracts and writes to the terminal each of the element names.

```
$ make read
g++ -o read read.cpp -O2 -ansi -pedantic -Wall
$ more resistivity.dat
Experimental Data

Trial 1
Performed January 1, 2009
Sample A (Calcium)
Resistivity measuresments [nOhm.m]
{ 35.0955, 34.9091, 34.9141, 33.7726, 34.0036, 34.8873, 34.0472, 34.0721, 34.7659 }
.
.
.
$ ./read < resistivity.dat
Calcium:
Aluminum:
Copper:
Silver:
Done.
```

Modify the program so that it also reports the number of measurements and their average. Your output should look like this:

```
$ ./read < resistivity.dat
Calcium: average over 9 measurements is 34.4964
Aluminum: average over 3 measurements is 27.3406
Copper: average over 5 measurements is 17.1692
Silver: average over 6 measurements is 16.2828
Done.
```

4. Heron's iterative method for computing the square root of number ($a \mapsto a^{1/2}$) is one of the most ancient numerical algorithms—it's been known for nearly two thousands years. It is straightforward to derive a generalization of this method that computes $a^{p/q}$ with p and q integer:

$$x_0 := 1$$
$$x_{n+1} := \left(1 - \frac{1}{q}\right) x_n + \frac{a^p}{q} x_n^{1-q}$$

If the sequence (x_n) converges, it must converge to $a^{p/q}$.

Fill in the body of the program `ratpow.cpp` so that the recurrence relation shown above is computed starting from `x = 1`. Arrange for values of `a`, `p`, and `q` to be read in from the command line. The formally correct stopping condition is that x_n^q becomes sufficiently close to a^p. Instead, let's just have the loop terminate when the next value in the sequence is sufficiently close to the last one. That comparison should be carried out not with the `==` operator but with a function `nearly_equal` that checks whether two numbers x and y satisfy $|x - y| < \epsilon|x|$ and $|x - y| < \epsilon|y|$.

```
$ make ratpow
g++ -o ratpow ratpow.cpp -O2 -ansi -pedantic -Wall -lm
$ ./ratpow
Returns the rational power a^(p/q) and compares to the
value computed with std::pow in the cmath library
Usage: ratpow a p q
        a=nonnegative fp number
      p,q=nonzero integers
$ ./ratpow 1.0 1 1
Newton's method value: 1
 C Math library value: 1
$ ./ratpow 2.0 1 2
Newton's method value: 1.414213562373095
 C Math library value: 1.414213562373095
$ ./ratpow 3.5 4 5
Newton's method value: 2.724296895429098
 C Math library value: 2.724296895429098
```

5. The file `series.cpp` defines a function `atan_series` that computes the first `N` terms of the series expansion

$$\begin{aligned}\arctan x &= x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \cdots \\ &= x\left[\left(\left(\left(\left(\cdots + \frac{1}{9}\right)x^2 - \frac{1}{7}\right)x^2 + \frac{1}{5}\right)x^2 - \frac{1}{3}\right)x^2 + 1\right].\end{aligned}$$

The function makes use of Horner's scheme (we encountered this in Lab 2) to minimize the number of arithmetic operations required. Change the code in `main` so that the output is in scientific notation and uses 10 decimal digits of precision. You should be able to reproduce exactly the session shown below.

```
$ make series
g++ -o series series.cpp -O2 -ansi -pedantic -Wall -lm
$ ./series
Calculates the sequence of partial series expansions

            x
            x - x^3/3
            x - x^3/3 + x^5/5
            x - x^3/3 + x^5/5 - x^7/7
            etc.

approximating atan(x) = x - x^3/3 + x^5/5 - x^7/7 + ...

Usage: series x-value
$ ./series 0.5
```

```
For x = 0.5, the successive partial expansions have the following values:
     1    4.5833333333e-01
     2    4.6458333333e-01
     3    4.6346726190e-01
     4    4.6368427579e-01
     5    4.6363988659e-01
     .
     .
     .
    28    4.6364760900e-01
    29    4.6364760900e-01
    30    4.6364760900e-01
-------------------------
   inf    4.6364760900e-01
$ gnuplot -persist view5.gp
```

The `view5.gp` script plots the results for $x = 0.5, 0.75, 1.0, 1.05$ as a function of the number of terms included. The radius of convergence for this expansion is 1.

6. Alternate series for $\arctan x$ with better convergence properties can be constructed by expanding with respect to new variable $y = x^2/(1+x^2)$ that maps $x \in \mathbb{R}$ onto $y(x) \in [0, 1)$. Keep in mind that y itself has an expansion $y = x^2 - x^4 + x^6 - x^8 + \cdots$, so a series truncated at finite order in y actually includes terms at all orders in x.

   version 2:
   Since the inverse of $y(x)$ is

$$x(y) = \pm\frac{\sqrt{(1-y)y}}{1-y},$$

   we can expand $(1/x)\arctan(x)$ as

$$\frac{1-y}{\sqrt{(1-y)y}}\arctan(x(y)) = 1 - \frac{1}{3}y - \frac{2}{3\cdot 5}y^2 - \frac{2\cdot 4}{3\cdot 5\cdot 7}y^3 - \frac{2\cdot 4\cdot 6}{3\cdot 5\cdot 7\cdot 9}y^4 - \cdots,$$

   which leads to

$$\begin{aligned}\arctan(x) &= x\left(\cdots + \frac{6\cdot 4\cdot 2\cdot(-1)}{9\cdot 7\cdot 5\cdot 3}y^4 + \frac{4\cdot 2\cdot(-1)}{7\cdot 5\cdot 3}y^3 + \frac{2\cdot(-1)}{5\cdot 3}y^2 + \frac{(-1)}{3}y + 1\right)\\ &= x\left[\left(\left(\left(\left(\left(\cdots + 1\right)\frac{6}{9}y + 1\right)\frac{4}{7}y + 1\right)\frac{2}{5}y + 1\right)\frac{-1}{3}y + 1\right].\end{aligned}$$

   version 3:
   Similarly, $(1/x + x)\arctan(x)$ has an expansion

$$\frac{x(y)}{y}\arctan(x(y)) = 1 + \frac{2}{3}y + \frac{2\cdot 4}{3\cdot 5}y^2 + \frac{2\cdot 4\cdot 6}{3\cdot 5\cdot 7}y^3 + \frac{2\cdot 4\cdot 6\cdot 8}{3\cdot 5\cdot 7\cdot 9}y^4 + \cdots.$$

   Rearranging the terms gives

$$\begin{aligned}\arctan(x) &= \frac{y}{x}\left(\cdots + \frac{2\cdot 4\cdot 6\cdot 8}{3\cdot 5\cdot 7\cdot 9}y^4 + \frac{2\cdot 4\cdot 6}{3\cdot 5\cdot 7}y^3 + \frac{2\cdot 4}{3\cdot 5}y^2 + \frac{2}{3}y + 1\right)\\ &= \frac{y}{x}\left[\left(\left(\left(\left(\left(\cdots + 1\right)\frac{8}{9}y + 1\right)\frac{6}{7}y + 1\right)\frac{4}{5}y + 1\right)\frac{2}{3}y + 1\right].\end{aligned}$$

   With these results in mind, make the following changes to `series.cpp`. Implement versions 2 and 3 of the expansion in the functions named `atan_series2` and `atan_series3`. Have the program output each version in its own column.

```
$ ./series 0.99
For x = 0.99, the successive partial expansions have the following values:
     1    6.6656700000e-01    9.9000000000e-01    4.9997474875e-01
     2    8.5676500998e-01    8.2665824958e-01    6.6495808335e-01
     3    7.2361281742e-01    7.9431821634e-01    7.3028818435e-01
     4    8.2511473381e-01    7.8517106891e-01    7.5800541305e-01
     5    7.4372034706e-01    7.8215266270e-01    7.7020037778e-01
     .
     .
     .
    28    7.8518321535e-01    7.8037308008e-01    7.8037307961e-01
    29    7.7581569879e-01    7.8037308007e-01    7.8037307985e-01
    30    7.8469578200e-01    7.8037308007e-01    7.8037307996e-01
--------------------------
   inf    7.8037308007e-01
$ gnuplot -persist view6.gp
```

The `view6.gp` script shows the rate of convergence graphically. Convince yourself that version 2 of the expansion converges the fastest.

7. (**Assignment 2:** to be submitted before the next lab period.) The *complete elliptic integral of the first kind*,

$$I(a,b) = \frac{1}{2}\int_{-\infty}^{\infty} \frac{dt}{\sqrt{(a^2+t^2)(b^2+t^2)}},$$

has the special property that

$$I(a,b) = I\Big(\frac{a+b}{2}, \sqrt{ab}\Big).$$

Amazingly, this simple result has rather deep implications (related to the so-called *arithmetic-geometric mean*) and can be used to derive [Borwein and Borwein, SIAM Review **26**, 351 (1984)] an iterative scheme for computing π that converges exponentially: viz., one starts with the initial conditions $\alpha_0 = \sqrt{2},\ \beta_0 = 0,\ \pi_0 = 2+\sqrt{2}$ and generates successive approximations recursively via

$$\begin{aligned}
\alpha_{n+1} &= \frac{1}{2}(\alpha_n^{1/2} + \alpha_n^{-1/2}),\\
\beta_{n+1} &= \alpha_n^{1/2}\Big(\frac{1+\beta_n}{\alpha_n+\beta_n}\Big),\\
\pi_{n+1} &= \pi_n\beta_{n+1}\Big(\frac{1+\alpha_{n+1}}{1+\beta_{n+1}}\Big).
\end{aligned}$$

There are also several exact identities relating linear combinations of arctan to π.

$$\begin{aligned}
\pi &= 4\arctan(1/2) + 4\arctan(1/5) + 4\arctan(1/8)\\
&= 16\arctan(1/5) - 4\arctan(1/239)\\
&= 24\arctan(1/8) + 8\arctan(1/57) + 4\arctan(1/239)
\end{aligned}$$

Modify the file `pi.cpp` so that the program outputs in the first column an integer count, in the second column the AGM expansion result, and in the three other columns the numerical estimates of π based on the arctan formulas shown above. The first column refers either to the number of iterations in the AGM scheme or to the number of terms included in the series expansion of arctan. Be sure to use the function `atan_series2` that you wrote for question 5. The final output should look like this:

```
$ make pi
g++ -o pi pi.cpp -O2 -ansi -pedantic -Wall -lm
$ ./pi
Usage: pi #iterations
$ ./pi 7
  1     3.1426067539416                3.3     3.1832635983264     3.1570872788666
  2      3.141592660966     3.1538461538462     3.1422380549654     3.1416881708556
  3     3.1415926535898     3.1430059171598      3.141606891258      3.141593494527
  4     3.1415926535898      3.141783262891     3.1415930195281     3.1415926622229
  5     3.1415926535898     3.1416206328643     3.1415926638427     3.1415926536865
  6     3.1415926535898     3.1415969882255     3.1415926538935     3.1415926535909
  7     3.1415926535898        3.1415933509     3.1415926535991     3.1415926535898
--------------------------------------------------------------------------------
inf     3.1415926535898
$ gnuplot -persist view7.gp
```

If you've done everything correctly, the script `view7.gp` should give you a plot of the discrepancy between the approximation and exact value in each of the four cases. Be sure you understand what is implied by the shapes of the various curves.