Assignment 1.

Given Sep 19 2000, due Oct 3 2000.

Objectives: To learn about floating point arithmetics, sources of error and start simple programming.

(1) Do exercises 1.2, 1.6, 1.7, 1.12, 1.13, 1.14, 1.15, 1.17 of the textbook.

(2) Supplementary to exercise 1.2, can you design an algorithm that can distinguish chopped decimal algorithm from rounded decimal algorithm?

(3) Based on Table 1.1 of the textbook, how many bytes does it take to store an IEEE single precision floating point number? and how many for a double precision one? (A byte is a set of eight digits of 0 and 1) When you run your program, the variables are usually stored in RAM. If the program calls input/output files, those files are stored in the hard disk after the the program terminates. This exercise should give you a rough idea of how much information you can store on your hard disk and/or RAM. (1 Megabytes $= 1024^2$ bytes, 1 Gigabytes $= 1024^3$ bytes).

(4) programming + paper-and-pen work Do either Exercise 1.11 or Example 1.13 (not both). To be more precise about Example 1.13:
(i) Write a program with infinite loop that terminates when the partial sum stops changing. Do this in SINGLE PRECISION and monitor the sum and number of terms, you should get $S = 15.\cdots$ in less then a few seconds. (Please do not try to run the infinite loop in double precision, you'll see why)
(ii) After (i) is done, try run it in double precision with a fixed number of terms (say 10^6 or the number of terms you got when (i) terminates) instead of a infinite loop. Record the CPU time. Estimate the total number of terms and the CPU time it takes to terminate if you ran the program with an infinite loop.
(iii) If you did Example 1.13, plot $\log(S_n)$ as a function of n. If you did exercise 1.11, repeat the procedure for $x = 0.1, 0.2, \cdots, 10.0$ (with a loop, of course) and plot $\log(your computed e^x)$ against x. The log here are meant to be the built in function that you call from your compiler, not a subroutine written by yourself.