Floating -Point Mini-lab

The exercises all have a **`makefile`**. gcc is the default compiler. If you define **`CXX=icc`** on the make command line, the icc compiler will be used to build the target. The default target is **`all`**, which builds all the files required in the particular exercise. There is also a **`realclean`** target to delete all the files created.

NOTE: you must build the realclean target before switching compilers. A typical sequence might be

```
make # build with gcc
# work on the exercise
# save any files which you way want later; e.g., executables
make realclean # clean before switching compilers
make CXX=icc # build using icc
# work on the exercise
# save any files which you way want later
make CXX=icc realclean # remove icc-created files
```

Exercise 01 – Muller

This is a perverse example constructed by Jean-Michel Muller. See section 1.3.2 of "Handbook of Floating-Point Arithmetic". The series should converge to 6 for the initial values $u_0 = 2$ and $u_1 = -4$. However, the computation converges to 100 in all cases.

For an analysis of what happens, see §5 in "How Futile are Mindless Assessments of Roundoff in Floating-Point Computation?" by W. Kahan at: http://www.eecs.berkeley.edu/~wkahan/Mindless.pdf.

The series is:

$$\begin{aligned} u_0 &= 2 \\ u_1 &= -4 \\ u_n &= 111 - \frac{1130}{u_{n-1}} + \frac{3000}{u_{n-1} * u_{n-2}} \end{aligned}$$

Note that all the constants are small integers which can be represented exactly as floating-point numbers. The general solution for the recurrence is

$$u_n = \frac{a * 100^{n-1} + b * 6^{n+1} + c * 5^{n+1}}{a * 100^n + b * 6^n + c * 5^n}$$

where a, b and c depend on the initial values u_0 and u_1. If $a \neq 0$, the limit for $n \to \infty$ is 100, but if $a = 0$, and $b \neq 0$, the limit is 6.

If $u_0 = 2$ and $u_1 = -4$, then $a = 0$, $b = -3$, $c = 4$ and the limit should be 6.

However the program doesn't compute 6. It computes 100!

Running instructions:

- enter 33 (or more) for n
- enter 2 for u_0
- enter -4 for u_1

You should see output such as the following:

```
n = 32
u0 = 2
u1 = -4
Computation from 3 to n:
u3=18.5
u4=9.378378378378379
u5=7.801152737752168
u6=7.154414480975334
u7=6.806784736924811
u8=6.592632768721792
u9=6.449465934053933
u10=6.348452060746624
u11=6.274438662728116
u12=6.218696768582163
u13=6.175853855815390
u14=6.142627170481006
u15=6.120248704570159
u16=6.166086559598099
u17=7.235021165534931
u18=22.062078463525793
u19=78.57557488787224
u20=98.34950312216536
u21=99.89856926618290
u22=99.99387098890279
u23=99.99963038728635
u24=99.99997773067949
u25=99.99999865921669
u26=99.99999991932181
u27=99.99999999514776
u28=99.99999999970828
u29=99.99999999998246
u30=99.99999999999893
u31=99.99999999999993
u32=99.99999999999999
```

The problem is caused by the computed value of u_4: it is slightly in error. The exact value of u_4 should be 9.3783783783783783784. Thus, each succeeding u_n is incorrect: the errors accumulate and grow as the recursion calculation proceeds. The expansion being calculated actually corresponds to a slightly different problem: one with a small non-zero value for a. Thus, the limit *should be* 100.

This is a typical example of the fact that often an algorithm when implemented in finite-precision floating-point arithmetic solves a slightly different problem than from which it was derived mathematically.

Exercise 02 – Rump

This is an example due to S. M. Rump. See section 1.3.2 of "Handbook of Floating-Point Arithmetic" for references and more details.
Running instructions:

- build and run the executable for **`float`**, **`double`** and **`__float128`** data types. The shell script **`DoAll`** does this.
- Compare the results.
- Repeat using icc (there is a **`DoAll-icc`** script) and compare the results with those for gcc.

The expression should evaluate to -0.827396…. However, regardless of the precision used, this result is never obtained.

For analyses of what happens, see the following references (Google is your friend):

- E. Low and W. Walster. Rump's example revisited. Reliable Computing 8(3):245-248, 2002.
- Cuyt, B. Verdonk, S. Becuwe and P. Kuterna. A remarkable example of catastrophic cancellation unraveled. Computing, 66:309-320, 2001.
- Also see §5 in "How Futile are Mindless Assessments of Roundoff in Floating-Point Computation ?" by W. Kahan at http://www.eecs.berkeley.edu/~wkahan/Mindless.pdf

Exercise 03 – Quadratic

This directory contains a simple program which solves the equation

$$f(x) = ax^2 + bx + c$$

for its real roots. It does so in a very numerically naïve way.

Compile the program and fill in this table (where x_+ and x_- are the roots found) using results from the naïve version:

a	b	c	x_+	x_-	$f(x_+)$	$f(x_-)$
+1	+2000	-3				
+2	-4000	-1				
+5	+8000	+2				

You should calculate the values of $f(x_+)$ and $f(x_-)$ by hand or with a calculator. Verify your results.

Study the values of the roots as displayed by the program. You should notice that the two roots are very different in magnitude; the ratio of their magnitudes is $\sim 10^6$. This is usually an indication of an ill-conditioned problem. Notice also that the low-order hex digits of the smaller root are usually repeated digits, often 0. This is caused by catastrophic cancellation in the calculation.

Now try the "improved" version of the program. (It is created by compiling the same source file with **`-D_IMPROVED`**.) Notice that the smaller root is now calculated more accurately and that the value of $f(x_{smaller})$ is closer to 0.

Solving the quadratic equation more accurately provides a simple example of how catastrophic cancellation can be removed from a problem algebraically.

The roots are given by

$$\begin{aligned} x_\pm &= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \\ &= -\frac{b}{2a}\left(1 \mp \sqrt{1 - \frac{4ac}{b^2}}\right) \end{aligned}$$

Let $\delta = 4ac/b^2$. Then

$$x_+ = -\frac{b}{2a}\left(1 - \sqrt{1-\delta}\right)$$

When $b^2 \gg 4ac$ (i.e., $\delta \ll 1$), the calculation of x_+ involves the taking the difference of two nearly equal computed quantities, resulting in catastrophic cancellation.

We can rationalize the expression for x_+ by multiplying numerator and denominator by $1 + \sqrt{1-\delta}$ giving

$$\begin{aligned} x_+ &= -\frac{b}{2a}\left(\frac{1-(1-\delta)}{1+\sqrt{1-\delta}}\right) \\ &= -\frac{2c}{b}\left(\frac{1}{1+\sqrt{1-\delta}}\right) \end{aligned}$$

and

$$x_- = -\frac{b}{2a}\left(1 + \sqrt{1-\delta}\right)$$

Now there is no catastrophic cancellation when x_+ is computed if $b^2 \gg 4ac$.

Exercise 04 – Summation

This directory contains the files to build a program which sums a collection of double precision values in different ways. The program also measures the execution time of each method. The summation techniques used include:

- simple summation, with and without sorting

- summation using hyper-precision arithmetic
- summation using quad-precision arithmetic
- an error free transformation (EFT) which implements a highly accurate summation scheme without use of increased precision
- vectorization
- unrolled loops
- OpenMP
- reduction methods from Intel® Threading Building Blocks (TBB)

First make some observations:

- Build the program with both gcc and icc and run each version at least 3 times.
- Are all the results always the same? Are they the same with both compilers?
- Which techniques give the same result each time? Which give results which vary? Can you explain why?
- Look at the source files containing the various functions. Understand how the various techniques are implemented.
- The main program demonstrates a method to measure elapsed time using OpenMP. Note that OpenMP is only used "computationally" in one of the summing routines.

Which of the techniques do you think provides the "correct" value for the sum? What do you think "correct" means in this case?

Examine the way in which the pseudo-random numbers are generated. Can you correlate that information with the different results? E.g., compare the "unroll by 4" results with the "unroll by 5" results.

Identify the techniques which use multiple threads of execution or multiple partial sums. Why does this affect the results?