PROBLEM for CS 179 : by Prof. W. Kahan

Exhibit a program that starts from any three given floating-point numbers x, y and z, and computes $p := x \cdot y \cdot z$ in some order that avoids undeserved over/underflow. Do likewise for $q := x \cdot y/z$.

SOLUTIONS: The proofs that these programs work correctly depend upon the properties of three *Environmental Constants* associated with the floating-point formats in which x, y, z, p and q are represented, regardless of whether those constants appear in the programs. The *Overflow threshold* Ω is the biggest finite number in that format; the *Underflow threshold* η is the smallest *normalized* positive number. The magnitudes of x, y and z are presumed to lie between Ω and $\varepsilon\eta$ inclusive where $\varepsilon\eta$ is the smallest *nonzero* magnitude and may be far tinier than η if underflow is *gradual*; on machines that underflow abruptly to zero $\varepsilon\eta = \eta$ except for CDC Cyber 17x's. $\varepsilon\eta = 2\eta$ for these Cybers to cope with "partially underflowed" numbers between η and $\varepsilon\eta$ that behave normally in *add*, *subtract* and *compare* but behave like zero in *multiply* and *divide*. Little is presumed about the product $\eta\Omega$, which lies very far from 1 on some machines.

An obvious program to compute p and q would first obtain their magnitudes using logarithms; $| p | = \exp(\ln | x | + \ln | y | + \ln | z |)$ and $| q | = \exp(\ln | x | + \ln | y | - \ln | z |)$. But these formulas lose accuracy badly when the data are very big or very small; the loss is caused by rounding each logarithm to working precision, and can be observed by comparing the computed values of $\exp(\ln | x |)$ and $| x |$ when it lies near Ω or η. And computing logarithms and exponentials wastes time. Our programs waste neither accuracy nor time.

Both programs start by Sorting $| x |, | y |$ and $| z |$ and continue thus:

Program for p :
Assume now that sorted $| x | \leq | y | \leq | z |$. Compute $x \cdot z$ first and then $p := (x \cdot z) \cdot y$ except on a machine with gradual underflow; on such a machine if $(x \cdot z)$ underflows recompute $p := (z \cdot y) \cdot x$.

Proof that p is correct.
If $x \cdot z$ overflowed, then $1 < | x | \leq | y | \leq \Omega < | x \cdot z | < | (x \cdot z) \cdot y |$ so p deserves to overflow too (except perhaps on a CRAY, which can overflow in certain cases when a product lies between $\Omega/2$ and Ω; but that is too perverse to consider here). Similarly if $x \cdot z$ underflowed on a machine that underflows abruptly to zero, then

$$1 > | z | \geq | y | \geq | x | \geq \eta > | x \cdot z | > | (x \cdot z) \cdot y |$$

so p must underflow too. On a machine that underflows gradually conformity with IEEE standards 754/854 requires also the ability to detect underflow, and this should be exploited if any of the data can be subnormal (i.e., between $\varepsilon\eta$ and η in magnitude). Then $x \cdot z$ underflows only when $1/\varepsilon \geq | z | \geq | y | \geq | x | \geq \varepsilon\eta$ and $\eta > | x \cdot z |$; since $\Omega > 1/\varepsilon^2$ on those machines, $\Omega > z \cdot y$ so $z \cdot y$ cannot overflow and if it underflows too then either $| z | > 1$ and then $| x \cdot y \cdot z | = | (x \cdot z)(z \cdot y)/z | < \eta^2 / | z | < \eta$, or else $| z | \leq 1$ and then $| x \cdot y \cdot z | < | x | \eta \leq \eta$, and p deserves to underflow either way.

Programs for q :
If we could treat q as a product $x \cdot y \cdot (1/z)$, we could compute it safely using the program for p; but the risk that $1/z$ may over/underflow precludes that option. A safe and simple program works on machines that allow programs to branch on over/underflow:

First swap x and y if necessary to establish $| x | \leq | y |$;
next compute $p := x \cdot y$; subsequently
if (p overflowed and $| z | > 1$) then $q := (y/z) \cdot x$
else if (p underflowed and $| z | < 1$) then $q := (((x/\varepsilon)/z) \cdot y) \cdot \varepsilon$
else $q := p/z$. (For Cybers use $\varepsilon = 1$ here, not 2.)

The validity of this program is easy to establish provided we may presume that $\sqrt{(\eta)}/\varepsilon^2 < \eta\Omega < \sqrt{\Omega}$, as appears to be true for all machines I know. But the ability to test for over/underflow and continue is not so common; what if over/underflow is silent? In the absence of a (portable) way to branch on over/underflow, we must produce a spaghetti-like code with branches that preclude spurious over/underflows. Such a program follows.

Two constants are needed. One is λ, the smallest power of the machine's radix no smaller than $\max\{1, 1/(\varepsilon\eta\Omega)\}$. The other is μ, the biggest power of the radix not exceeding $\min\{1, 1/(\eta\Omega)\}$. Multiplication by λ or μ is exact, so it cannot cause underflow on a machine that conforms to IEEE 754/854.

First sort $| x |, | y |$ and $| z |$, keeping track of z. This reduces the situation to one of three cases, depending upon whether $| z |$ is minimal, maximal, or neither:

In case $| z |$ is $>$ minimal, say $| z | \leq | x | \leq | y |$, test $| y |$;
if $| y | > 1$ then $q := (x/z) \cdot y$
else $q := (x/(\lambda z)) \cdot (\lambda y)$.
In case $| z |$ is maximal, say $| z | \leq | y | \leq | x |$, test $| x |$;
if $| x | < 1$ then $q := (y/z) \cdot x$
else $q := (y/(\mu z)) \cdot (\mu x)$.
In case $| z |$ is neither, say $| x | \leq | z | \leq | y |$, test both;
if $| x | > 1$ then $q := (y/z) \cdot x$
else if $| y | < 1$ then $q := (x/z) \cdot y$
else $q := (x \cdot y)/z$.

The proof that this program is correct is a tedious exercise in elementary inequalities, and is left to the reader.