A. Simple Arithmetic

This problem was trickier than it looked. For the first two subgroups it was enough to use regular double arithmetic, as long as the output was printed with enough decimals (e.g. in C++ `cout << setprecision(10) << fixed` was helpful).

For the other test groups, however, this is not enough. Doubles only have 53 bits of precision (plus a 11-bit exponent), and we would need more than 60 bits just to be able to perfectly represent all integers up to 10^{18}.

80-bit doubles (i.e. `long long`s in C++), however, use 64 bits for the mantissa, which means that they can exactly represent any integer up to 2^{64}, which is sufficient to completely solve group 3. Group 3 could also be solved by using 64-bit integers when $c = 1$.

Unfortunately, long doubles are still not enough for group 4, because the answer is not necessarily an integer. One work-around would be to use Python's or Java's arbitrary precision arithmetic. However, that's a bit boring. A more hands-on solution which just uses plain integers/doubles (and thus works in C++/Pascal) is to compute the integer part and the fractional part separately. In other words, you can compute the integer part of $(a \cdot b)/c$ using 64-bit integers, and you can compute the fractional part as $((a \cdot b)\%c)/c$, which gives you an absolute precision of 10^{-15}, far better than needed. Example C++ solution:

```
int main() {
    long long a, b, c;
    cin >> a >> b >> c;
    long long ab = a * b;
    if (ab >= 10*c) {
        cout << ab / (10*c);
        ab %= 10*c;
    }
    cout << setprecision(18) << fixed << (long double)ab / c << endl;
}
```

B. Citations

We start by computing for each book `b` the time it takes to read it and all books in its subtree (`b.time`), and the number of books in that tree (`b.size`). Then, we will assign to each book a penalty `b.pen`, which is the contribution to the final answer of this book's subtree assuming we start reading it at time `T` $= 0$.

If we instead started reading it at another time `T`, the contribution would be `b.pen` $+$ `T` $\cdot$ `b.size`. The term `b.pen` is independent of `T`, and hence we can solve the problem for all books independently, ignoring their starting times.

To solve the problem for a given book, we need to find an order in which to read its children. There are several conceivable ways of doing this (trying all permutations, DP), but it turns out that it's possible to do it greed-

ily: we simply sort all children by increasing `b.time / b.size`. A swapping argument shows that this indeed minimizes the sum over all children of `(sum of b.time of previous books) * b.size`.

C. Ninety-nine

There is a simple winning strategy for the second player in this game: regardless of what your opponent says, you can always say the numbers 3, 6, 9, ..., 99, and win. The only problem is, you're not the second player.

What we'll need to do, then, is to exploit the fact that the first player is not playing optimally. One option is to play randomly, and if the opponent ever slips up and deviates from the optimal strategy, we can say a multiple of 3 and continue as if we were the second player. This succeeds with probability $1 - 2^{-33}$.