creating a tiny big number library

Tiny Big Number Lib - Background

Tiny Big Number Lib (tbnlib) was written so that I could do physics projects and other large simulations in C++ without a huge amount of dependency overhead. It aims to implement the **bare minimum** necessary to do these calculations.

---

big(s) and signing

We can represent bigs as $\langle n_0, ..., n_i \rangle \to n_i ... n_0$, where $n \in [0, 9] \cup \mathbb{N}$ is a digit, and $i \in \mathbb{N}_{0\leq}$ is a index.

There are three possible states for a `big`; `unsigned`, `signed`, and `float`. Like standard modifiers, they can be applied in conjunction with each other; e.g. `signed float big`.

Fourier Division

> The following exposition assumes that the numbers are broken into two-digit pieces, separated by commas: e.g. 3456 becomes $34, 56$. In general x, y denotes $x \cdot 100 + y$ and x, y, z denotes $x \cdot 10000 + y \cdot 100 + z$, etc.

Procedure:

1. Shift to set Two-Digit Pieces
2. Apply General Form
3. Apply Power Modification & Place Decimal
4. Formatting the Remainder

General Term

We want to divide c by a, which gives b. First, we must expand to $\frac{c_0,...,c_i}{a_0,...,a_i} \implies b_0, ..., b_i \to b$

The general form follows:

$$b_i = \frac{r_{i-1}, c_{i+1} - \sum_{j=2}^{i} b_{i-j+1} \cdot a_j}{a_1}, \text{ with remainder } r_i$$

where any variables on negative indices, e.g. $r_{i-1},\ i = 0$, are nullified.

"Two-Digit Pieces"

Uh oh; what if $0 < [(a \vee c)\ (mod\ 2)]$?

We just append (in `big` form, prepend) 0 to a and c until $[d(a) = d(c)] \wedge [0 = c_i\ (mod\ a_i)]$, where $d(x) = \lfloor log_{10} x \rfloor + 1$.

The easiest way to do this is to take the higher digit count integer, ensure that the digit count is even (if it's not, make it even), and then bring the other integer up. *We must move the decimal place of b forward by however many* 0 *we add in this stage.*

```
big[2] ac{a, c};
unsigned int x = (d(a) < d(c)) ? 1 : 0; // if d(a) == d(c), it doesn't matter, a || c
while(d(ac[x])%2) {
    ac[x].push_front(0);
    j++; // where j is the decimal index
}
unsigned int y = 1 - x; // other big
while(d(ac[x]) < d(ac[y])) {
    ac[y].push_front(0);
}
Copy
```

Power Modifier

Storing the bigs in a flipped format means that we can use the vector index to easily determine the appropriate power modifier.

We can induce the Power Modifier by applying $x \cdot 10^{i+1} \mid 0 < i$, where i is the (base 0) index of x. When $i = 0$, we just apply x. This produces the form;

$$p(x) = \begin{cases} x \cdot 10^{i+1} & \mid 0 < i \\ x & \mid i = 0 \end{cases}$$

which we apply to our big;

$$n_i ... n_0 \rightarrow p(n_0) ... p(n_i)$$

With this modification, we're able to then flip the order and then apply the rest of the transformation.

Reference: Wikipedia