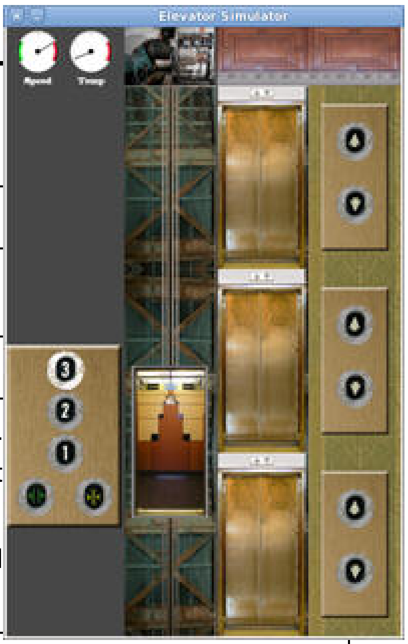
Project 2: Error History Help Page



So, how to you maintain the last 200 errors? You will need an array to do that, something like:

```
double errors[N];
```

where you have:

```
#define N 200
```

We use a #define so we don't hard code that number 200 at many places in your program. Doing so would make it hard to change the history length without introducing bugs.

If you initialize this array to all zeros, the total for the array will be zero as well. I would suggest also keeping that total around as a variable:

```
double errorTotal = 0;
```

Now, a naive approach might be to, for every new error you compute, move the contents of the array down, something like this:

```
    for(i=0;  i<N-1;  i++)
    {
        errors[i] = errors[i + 1];
    }

    errors[N-1] = e;
```

While this will work, it is very inefficient. A better approach is, for every new error, store the error in these locations:

0, 1, 2, ..., 198, 199, 0, 1, ...

Suppose I want to keep track of the last 5 numbers I heard someone say. I might have 5 cards and write each new number on a new card. When I hear the 6th number, I am out of cards, so I go back to the first card, cross out the old number and write a new one.

If you just keep track of the locations in a variable named errorLoc. Be sure errorLoc is an **int**, not a double and initialize it to zero, You just store the new error at location errorLoc and increment it like this:

```
    errorLoc++;
    if(errorLoc >= N)
    {
        errorLoc = 0;
    }
```

What this does is put the first error in the first location (location 0), the second in location 1, etc. until we get to the end of the array. Then we start over at the beginning again. Try some numbers by hand, assuming N = 5 for example and see what this does.

You can also this:

```
errorLoc = (errorLoc + 1) % N;
```

The % operator in C is called the modulus operator. a % b is the remainder after integer division of a by b. So, if you divide 5 by 5, the remainder is zero. This will work the same as the previous code example.

Finally, how do we sum the errors? Why sum them at all. We have a sum of the errors. If we replace something in the array with a new error, you can subtract out what you remove and add in what you just added. In our cards example, if I have a total of the last five numbers and I get a new number, I can subtract out the number I cross out and add in the new number I write down and the total will remain accurate.

So, how does Dt fit into this? I think it's easier to just multiply the sum by Dt instead of multiplying the error values you put in the array by Dt, but either way works.

I am getting repeated questions about *errorLoc*. Imagine you have 200 notecards in order. The notecards all have zeros on them. That's what your error history array is. They are in order, from card 0 to card 199. 1000 times per second you compute a new error. These cards will keep track of the last 200 errors. So, put a post-it note on the first card. That's what errorLoc is. When a new error comes in, cross out the zero on the card with the post-it note on it and write the error on it. Then move the post-it note to the next card.

When you write the error on the last card, card 199, you move the post-it note back to the first card. That's all that errorLoc is doing. Is that more clear?

Now, the summation says you need the sum of these cards. For every time you change the number on a card you could add up all 200 cards. I dare you to do that. You would get very tired of adding cards. If you keep track of the total for all cards, then each time you cross something off on a card, you subtract that from the total and each time you write a new number on a card you add it to the total. Then you have a total at all times. That's all there is to it.

Return to Project 2 Part A.

CSE 251