Mapping a numeric range onto another

[+55] [5] Joe
[2011-04-20 14:21:33]
[c math arduino]
[https://stackoverflow.com/questions/5731863/mapping-a-numeric-range-onto-another]

Math was never my strong suit in school :(

```
int input_start = 0; // The lowest number of the range input.
int input_end = 254; // The lowest number of the range input.
int output_start = 500; // The lowest number of the range output.
int output_end = 5500; // The largest number of the range ouput.

int input = 127; // Input value.
int output = 0;
```

How can I convert the input value to the corresponding output value of that range?

For example, an input value of "0" would equal an output value of "500", an input value of "254" would equal an output value of "5500". I can't figure out how to calculate an output value if an input value is say 50 or 101.

I'm sure it's simple, I can't think right now :)

Edit: I just need whole numbers, no fractions or anything.

[+98] [2011-04-20 14:57:20] Alok Singhal [✓ACCEPTED]

Let's forget the math and try to solve this intuitively.

First, if we want to map input numbers in the range [`0`, `x`] to output range [`0`, `y`], we just need to scale by an appropriate amount. 0 goes to 0, `x` goes to `y`, and a number `t` will go to `(y/x)*t`.

So, let's reduce your problem to the above simpler problem.

An input range of [`input_start`, `input_end`] has `input_end - input_start + 1` numbers. So it's equivalent to a range of [`0`, `r`], where `r = input_end - input_start`.

Similarly, the output range is equivalent to [`0`, `R`], where `R = output_end - output_start`.

An input of `input` is equivalent to `x = input - input_start`. This, from the first paragraph will translate to `y = (R/r)*x`. Then, we can translate the `y` value back to the original output range by adding `output_start`: `output = output_start + y`.

This gives us:

```
output = output_start + ((output_end - output_start) / (input_end - input_start)) * (input - input_start)
```

Or, another way:

```
/* Note, "slope" below is a constant for given numbers, so if you are calculating
   a lot of output values, it makes sense to calculate it once.  It also makes
   understanding the code easier */
slope = (output_end - output_start) / (input_end - input_start)
output = output_start + slope * (input - input_start)
```

Now, this being C, and division in C truncates, you should try to get a more accurate answer by calculating things in floating-point:

```
double slope = 1.0 * (output_end - output_start) / (input_end - input_start)
output = output_start + slope * (input - input_start)
```

If wanted to be even more correct, you would do a rounding instead of truncation in the final step. You can do this by writing a simple `round` function:

```
#include <math.h>
double round(double d)
{
    return floor(d + 0.5);
}
```

Then:

```
output = output_start + round(slope * (input - input_start))
```

(3) this worked perfectly, many thanks! - **Joe**
(2) I've written this function at least a few times over the decades, and yet this explanation is better than I could give myself. +1 for pulling out the divide - was useful for 6.5 million calculations per run. - **delrocco**

(2) This is one of the best answers I've seen on this site. Thank you! - **foobarbaz**

1

[+22] [2011-04-20 19:45:56] Dustin

Arduino has this built-in as map [1].

Example:

```
/* Map an analog value to 8 bits (0 to 255) */
void setup() {}

void loop()
{
  int val = analogRead(0);
  val = map(val, 0, 1023, 0, 255);
  analogWrite(9, val);
}
```

It also has the implementation on that page:

```
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

[1] http://www.arduino.cc/en/Reference/Map

(8) FYI, their `map` function is broken. It was converted from a floating point function by simply changing all the arguments to int, and it has a broken distribution. - **Mud**

2

[+11] [2011-04-20 14:37:35] Sven Marnach

The crucial point here is to do the integer division (which includes rounding) at the right place. None of the answers so far got the parentheses right. Here is the right way:

```
int input_range = input_end - input_start;
int output_range = output_end - output_start;

output = (input - input_start)*output_range / input_range + output_start;
```

3

[+10] [2017-03-27 11:20:45] Erti-Chris Eelmaa

the formula is

> f(x) = (x - input_start) / (input_end - input_start) * (output_end - output_start) + output_start

I'll hook up this post here: https://betterexplained.com/articles/rethinking-arithmetic-a-visual-guide/ as it helped me a lot when trying to come up with this intuitively. Once you understand what the post is saying, it's trivial to come up with these formulas on your own. Note that I used to struggle with such questions as well. (I have no affiliations - just found it very useful)

say you have range `[input_start..input_end]`, let's start by normalising it such that 0 is `input_start`, and 1 is `input_end`. this is simple technique to make the problem easier.

how do we do that? we'll, we'd have to shift everything left by input_start amount, such that if input x happens to be `input_start`, it should give zero.

so, let's say `f(x)` is the function that does the conversion.

```
f(x) = x - input_start
```

let's try it:

```
f(input_start) = input_start - input_start = 0
```

works for `input_start`.

at this point, it does not work for `input_end` yet, as we have not scaled it.

let's just scale it down by the length of the range, then we'll have the biggest value (input_end) mapped to one.

```
f(x) = (x - input_start) / (input_end - input_start)
```

ok, let's give it a try with `input_end`.

`f(input_end) = (input_end - input_start) / (input_end - input_start) = 1`

awesome, seems to work.

okay, next step, we'll actually scale it to output range. It is as trivial as just multiplying with the actual length of the output range, as such:

```
f(x) = (x - input_start) / (input_end - input_start) * (output_end - output_start)
```

now, actually, we're almost done, we just have to shift it to right so that 0 starts from output_start.

```
f(x) = (x - input_start) / (input_end - input_start) * (output_end - output_start) + output_start
```

let's give it a quick try.

```
f(input_start) = (input_start - input_start) / (input_end - input_start) * (output_end - output_start) + output_start
```

you see that the first part of equation is pretty much multiplied by zero, thus cancelling everything out, giving you

```
f(input_start) = output_start
```

let's try `input_end` as well.

```
f(input_end) = (input_end - input_start) / (input_end - input_start) * (output_end - output_start) + output_start
```

which in turn will end up as:

```
f(input_end) = output_end - output_start + output_start = output_end
```

as you can see, it now seems to be mapped correctly.

4

[+2] [2011-04-20 14:25:31] QuantumMechanic

```
output = ((input - input_start)/(input_end - input_start)) * (output_end - output_start) + output_start
```

What that does is find out proportionally "how far into" the input range the input is. It then applies that proportion to the size of the output range to find out in absolute terms how far into the output range the output should be. It then adds the start of the output range to get the actual output number.

This will always give `500`, except for the input `254`, which will yield `5500`. - **Sven Marnach**

5