A More Complex ADT

> This is a **warmup** exercise. It is **not compulsory**, and may be completed **individually or with your lab partner**.

A curious and useful feature of ADTs is that, because there is no direct dependence on any one implementation of the ADT by its consumers, you can write *multiple* implementations that all conform to the same interface.

Let's play with that idea a bit, and go back to our trusty `Complex` ADT.

Adding complex numbers is fairly straight-forward in the Cartesian form: we just add the real parts and add the imaginary parts. With multiplication, there's a bit more complexity but it's still fairly tractable.

Taking an exponent becomes a proper mess, though: we can't really express it cleanly by just thinking about real and imaginary parts... so let's think about something different.

On the Argand diagram, where the horizontal axis is the real part, and the vertical axis is the imaginary part, if we plot any complex number, yes, it has an (x, y) pair, but it's also a certain distance away from the origin, and the line forms a certain angle from the real axis.

These two values are the *magnitude*, usually written r, and the *phase*, usually written φ. The *polar form* of a complex number is made up of the magnitude and phase, not the real and imaginary parts.

It turns out that we can do simple trig to convert back and forth from Re/Im to r/φ:
$\Re(z) = r\cos\varphi$ and $\Im(z) = r\sin\varphi$.

So, with that in mind, how might we implement a Complex ADT storing their polar form?

Copy your `Complex.c` to `ComplexPolar.c`, and start tweaking it.

```
typedef struct _complex {
    double mod;
    double arg;
} complex;
```

Fairly obviously, `complexMod` and `complexArg` become trivial getter functions. `complexRe` and `complexIm` now need to do some work to get their results.

If you've implemented your `complexAdd` by simply `return`ing a call to `newComplex`, you can just use that.

A useful function might be `newComplexModArg`, a helper which makes other functions much easier to write. Unlike `newComplex`, it takes `mod` and `arg`:

```
static Complex newComplexModArg (double mod, double arg) {
    complex *c = calloc (1, sizeof (complex));
    if (c == NULL) {
        printf ("Complex: couldn't allocate memory\n");
        exit (1);
    }
    c->mod = mod;
    c->arg = arg;
    return c;
}
```

You could replace the guts of `newComplex` and of `complexMultiply` with a call straight to this helper.

And now, a function like `complexExp` could be written very simply indeed:

```
// takes z^x.
Complex complexExp (Complex z, double x) {
    return newComplexModArg (
        pow (z->mod, x), z->arg * x);
}
```

When you've finished your implementation, we have the same test suite as before hooked up as an Autotest.

To run some simple automated tests:

```
$ 1511 autotest complexADT2
```

To run Styl-o-matic:

```
$ 1511 stylomatic ComplexPolar.c
Looks good!
```

You'll get advice if you need to make changes to your code.

Submit your work with the *give* command, like so:

```
$ give cs1511 wk08_complexADT2
```

Or, if you are working from home, upload the relevant file(s) to the `wk08_complexADT2` activity on Give Online.