Conditional Compilation

Rust has a special attribute, `#[cfg]`, which allows you to compile code based on a flag passed to the compiler. It has two forms:

```
#[cfg(foo)]

#[cfg(bar = "baz")]
```

They also have some helpers:

```
#[cfg(any(unix, windows))]

#[cfg(all(unix, target_pointer_width = "32"))]

#[cfg(not(foo))]
```

These can nest arbitrarily:

```
#[cfg(any(not(unix), all(target_os="macos", target_arch = "powerpc")))]
```

As for how to enable or disable these switches, if you're using Cargo, they get set in the `[features]` section of your `Cargo.toml`:

```
[features]
# no features by default
default = []

# Add feature "foo" here, then you can use it.
# Our "foo" feature depends on nothing else.
foo = []
```

When you do this, Cargo passes along a flag to `rustc`:

```
--cfg feature="${feature_name}"
```

The sum of these `cfg` flags will determine which ones get activated, and therefore,

which code gets compiled. Let's take this code:

```
#[cfg(feature = "foo")]
mod foo {
}
```

If we compile it with `cargo build --features "foo"`, it will send the `--cfg feature="foo"` flag to `rustc`, and the output will have the `mod foo` in it. If we compile it with a regular `cargo build`, no extra flags get passed on, and so, no `foo` module will exist.

cfg_attr

You can also set another attribute based on a `cfg` variable with `cfg_attr`:

```
#[cfg_attr(a, b)]
```

Will be the same as `#[b]` if `a` is set by `cfg` attribute, and nothing otherwise.

cfg!

The `cfg!` macro lets you use these kinds of flags elsewhere in your code, too:

```
if cfg!(target_os = "macos") || cfg!(target_os = "ios") {
    println!("Think Different!");
}
```

These will be replaced by a `true` or `false` at compile-time, depending on the configuration settings.