- Start Date: 2014-07-24
- RFC PR #: https://github.com/rust-lang/rfcs/pull/184
- Rust Issue #: https://github.com/rust-lang/rust/issues/16950

Summary

Add simple syntax for accessing values within tuples and tuple structs behind a feature gate.

Motivation

Right now accessing fields of tuples and tuple structs is incredibly painful—one must rely on pattern-matching alone to extract values. This became such a problem that twelve traits were created in the standard library (`core::tuple::Tuple*`) to make tuple value accesses easier, adding `.valN()` , `.refN()` , and `.mutN()` methods to help this. But this is not a very nice solution—it requires the traits to be implemented in the standard library, not the language, and for those traits to be imported on use. On the whole this is not a problem, because most of the time `std::prelude::*` is imported, but this is still a hack which is not a real solution to the problem at hand. It also only supports tuples of length up to twelve, which is normally not a problem but emphasises how bad the current situation is.

Detailed design

Add syntax of the form `<expr>.<integer>` for accessing values within tuples and tuple structs. This (and the functionality it provides) would only be allowed when the feature gate `tuple_indexing` is enabled. This syntax is recognised wherever an unsuffixed integer literal is found in place of the normal field or method name expected when accessing fields with `.` . Because the parser would be expecting an integer, not a float, an expression like `expr.0.1` would be a syntax error (because `0.1` would be treated as a single token).

Tuple/tuple struct field access behaves the same way as accessing named fields on normal structs:

```
// With tuple struct
struct Foo(int, int);
let mut foo = Foo(3, -15);
foo.0 = 5;
assert_eq!(foo.0, 5);

// With normal struct
struct Foo2 { _0: int, _1: int }
let mut foo2 = Foo2 { _0: 3, _1: -15 };
foo2._0 = 5;
assert_eq!(foo2._0, 5);
```

Effectively, a tuple or tuple struct field is just a normal named field with an integer for a name.

Drawbacks

This adds more complexity that is not strictly necessary.

Alternatives

Stay with the status quo. Either recommend using a struct with named fields or suggest using pattern-matching to extract values. If extracting individual fields of tuples is really necessary, the `TupleN` traits could be used instead, and something like `#[deriving(Tuple3)]` could possibly be added for tuple structs.

Unresolved questions

None.