Ownership | Lifetimes

Ownership Review

- Data in Rust programs have an owner
  - Owner is responsible for cleaning up data
    - Memory management
  - Only one owner (by default)
  - Functions, closures, structs, enums, and scopes are owners
- Data can be transferred (moved) from one owner to another
  - Function calls, variable reassignment, and closures
- Possible to “borrow” data from an owner
  - Owner still responsible for clean up

Ownership Review - Example

```rust
#[derive(Debug)]
enum FrozenItem {
    IceCube,
}

#[derive(Debug)]
struct Freezer {
    contents: Vec<FrozenItem>,
}
```

```rust
fn place_item(
    freezer: &mut Freezer,
    item: FrozenItem
) {
    freezer.contents.push(item);
}

fn main() {
    let mut freezer = Freezer { contents: vec![] };
    let cube = FrozenItem::IceCube;
    place_item(&mut freezer, cube);
    // cube no longer available
}
```

Lifetimes

- A way to inform the compiler that borrowed data will be valid at a specific point in time
- Needed for:
  - Storing borrowed data in structs or enums
  - Returning borrowed data from functions
- All data has a lifetime
  - Most cases are elided

Lifetime Syntax - struct

```rust
struct Name<'a> {
    field: &'a DataType,
}
```

- Convention uses 'a, 'b, 'c
- 'static is reserved
  - 'static data stays in memory until the program terminates

Lifetime Example - struct

```rust
enum Part {
    Bolt,
    Panel,
}

struct RobotArm<'a> {
    part: &'a Part,
}

struct AssemblyLine {
    parts: Vec<Part>,
}
```

```rust
fn main() {
    let line = AssemblyLine {
        parts: vec![Part::Bolt, Part::Panel],
    };
    {
        let arm = RobotArm {
            part: &line.parts[0],
        };
    }
    // arm no longer exists
}
```

Lifetime Syntax - function

```rust
fn name<'a>(arg: &'a DataType) -> &'a DataType {}
```

Solidifying understanding

- Lifetime annotations indicate that there exists some owned data that:
  - “Lives at least as long” as the borrowed data
  - “Outlives or outlasts” the scope of a borrow
  - “Exists longer than” the scope of a borrow
- Structures utilizing borrowed data must:
  - Always be created ***after*** the owner was created
  - Always be destroyed ***before*** the owner is destroyed

Recap

- Lifetimes allow:
  - Borrowed data in a structure
  - Returning references from functions
- Lifetimes are the mechanism that tracks how long a piece of data resides in memory
- Lifetimes are usually elided, but can be specified manually
- Lifetimes will be checked by the compiler