COMP6991 23T1

Functions

Function pointers

```
fn(T,U,V,…) -> R
```

MyMap begins

Function pointer limitations

No envrionment!

Closures

```
|t,u,v,…| <body>
```

What is the type of a closure?

(uh oh)

```
let x: ? = |x,y,z| { … }
```

Closure traits

The good news: they do implement *these* funky traits!

`FnOnce`

`FnMut`

`Fn`

(we'll use this one for now)

Closure trait syntax

```
FnOnce(T,U,V,…) -> R

FnMut(T,U,V,…) -> R

Fn(T,U,V,…) -> R
```

MyMap returns

Ownership recap

Consider these functions:

```
struct Foo;
impl Foo {
    fn foo_owned(self) { … }
    fn foo_exclusive(&mut self) { … }
    fn foo_shared(&self) { … }
}
```

What are the restrictions on calling each of these functions?

If you have <trait>, you can call it...

FnOnce	FnMut	Fn
It can be called only once	It can be called many times, but never simultaneously	It can be called many times, simultaneously

<trait>'s implementation is determined by

FnOnce	FnMut	Fn
Closure captures value by move	Closure captures exclusive borrow	Closure captures only shared borrows

In order to call <trait>, you require...

FnOnce	FnMut	Fn
`self`	`&mut self`	`&self`

All together now…

	`self`	`&mut self`	`&self`	`<none>`
`T`	Owned, can only be used once	Can only hold one at a time	Can have many at a time	No semantics
`fn` type	**`FnOnce`** Can only be called once	**`FnMut`** Can only be called once at a time	**`Fn`** Can be called many times simultaneously	**`fn`** No extra semantics

Hot tip for good API design

`FnOnce > FnMut > Fn > fn`

Most flexible  Least flexible

What should MyMap take?

MyMap: Endgame

What about FnOnce?

How about...

`time_function`!