CS 420 Advanced Programming Languages
Fall Semester, 2022
Doc 7 Rust Misc
Sep 20, 2022

```
fn increase(x: &mut i32) {
    *x += 1;
}
```

```
let mut b = Box::new(5);
increase(&mut b);
assert_eq!(*b,6);
```

Box is a Struct

Implementing Operators

```
use std::ops::Add;

#[derive(Debug, Copy, Clone, PartialEq)]
struct Point {
    x: i32,
    y: i32,
}

impl Add for Point {
    type Output = Point;

    fn add(self, other: Point) -> Point {
        Point {
            x: self.x + other.x,
            y: self.y + other.y,
        }
    }
}
```

```
fn main() {
    assert_eq!(
        Point { x: 1, y: 0 } + Point { x: 2, y: 3 },
        Point { x: 3, y: 3 }
    );
}
```

Operators

Add
AddAssign
BitAnd
BitAndAssign
BitOr
BitOrAssign
BitXor
BitXorAssign
Deref
DerefMut
Div
DivAssign
Drop
Fn
FnMut
FnOnce
Index
IndexMut
Mul
MulAssign
Neg
Not
RangeBounds
Rem
RemAssign
Shl
ShlAssign
Shr
ShrAssign
Sub
SubAssign

Can not create other operators

Attributes

```
#[test]
#[ignore = "not yet implemented"]
fn mytest() {
    // …
}
```

```
#[test]
#[should_panic(expected = "values don't match")]
fn mytest() {
    assert_eq!(1, 2, "values don't match");
}
```

```
#[derive(PartialEq, Clone)]
struct Foo<T> {
    a: i32,
    b: T,
}
```

```
impl<T: PartialEq> PartialEq for Foo<T> {
    fn eq(&self, other: &Foo<T>) -> bool {
        self.a == other.a && self.b == other.b
    }

    fn ne(&self, other: &Foo<T>) -> bool {
        self.a != other.a || self.b != other.b
    }
}
```

Printing

```
println!("{ }", 1234.56789);              // 1234.56789
println!("{:.2 }", 1234.56789);           // 1234.57
println!("{:?}", (3, 4));                 // (3, 4)  //:? debug

let a = 2;
let b = 3;
println!("{} + {} is {}", a, b,  a + b);     //2 + 3 is 5
```

Debug Printing

```
let mut map = HashMap::new();
map.insert("Portland", (45.5237606,-122.6819273));
map.insert("Taipei",   (25.0375167, 121.5637));

println!("{:?}", map);          // {"Portland": (45.5237, -122.6819), "Taipei": (25.0375, 121.5637)}
println!("{:#?}", map);
```

```
{
    "Portland": (
        45.5237,
        -122.6819,
    ),
    "Taipei": (
        25.0375,
        121.5637,
    ),
}
```

{:#?}
    Formatted debug

Debug Printing

```
#[derive(Debug)]
struct Complex { re: f64, im: f64 }


let a = Complex { re: -0.5, im: f64::sqrt(0.75) };
println!("{:?}", a);                              // Complex { re: -0.5, im: 0.8660254037844386 }
```

Without Debug the print statement will not compile

Life Times

```
fn main() {
    let r;                    // ---------+-- 'a
                              //          |
    {                         //          |
        let x = 5;            // -+-- 'b  |
        r = &x;               //  |       |
    }                         // -+       |
                              //          |
    println!("r: {}", r);     //          |
}                             // ---------+
```

Does not Compile

```
fn longest(x: &str, y: &str) -> &str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

Compiler can not tell the lifetime of the return value

Need to specify the lifetimes explicitly

```
fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

Both arguments and the return value will live at least as lifetime 'a

The return value can not live longer than either of the arguments

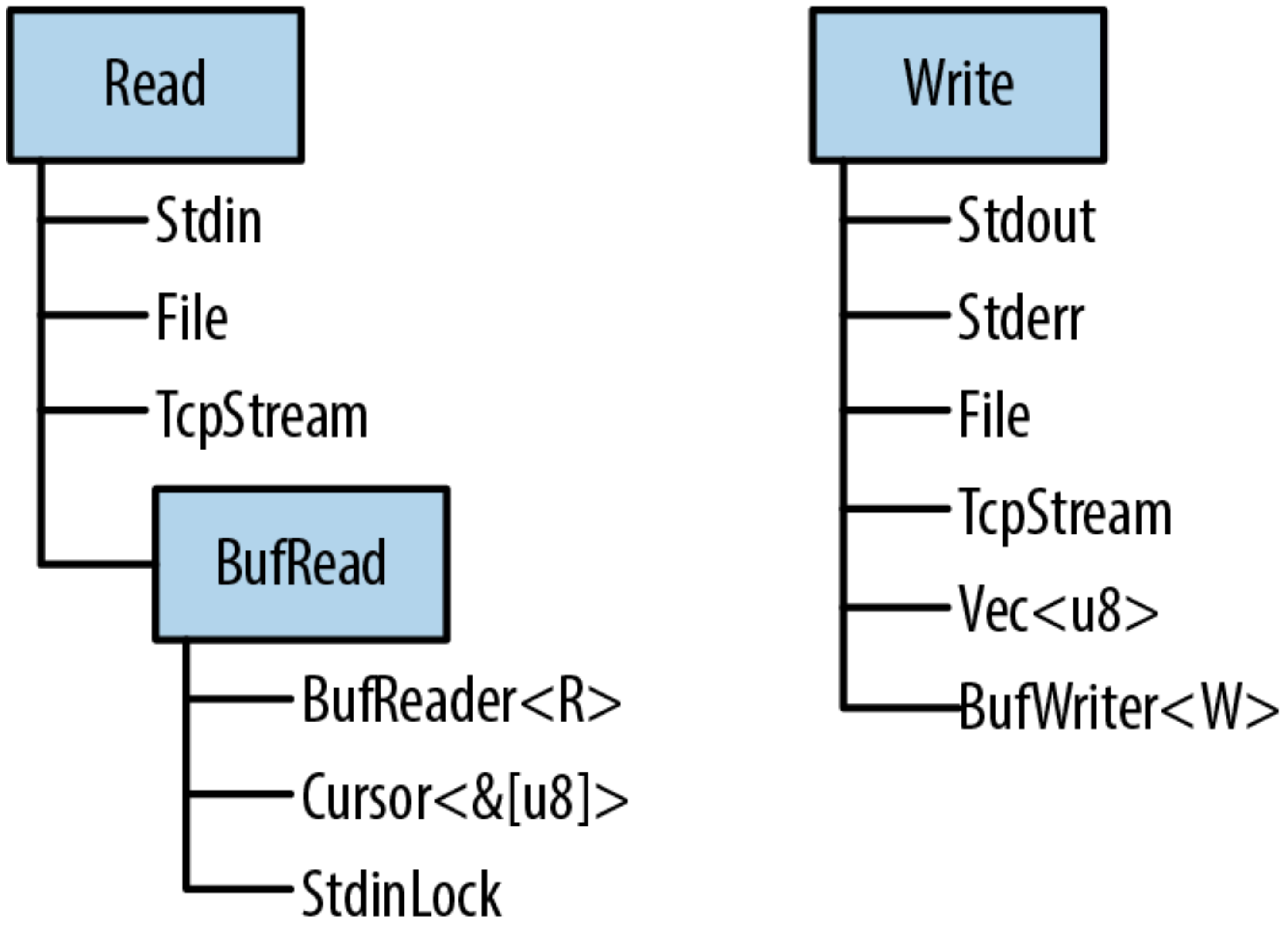
```
fn main() {
    let string1 = String::from("long string is long");

    {
        let string2 = String::from("xyz");
        let result = longest(string1.as_str(), string2.as_str());
        println!("The longest string is {}", result);
    }
}
```

This works as the return value does not outlast either of it arguments

```
fn main() {
    let string1 = String::from("long string is long");
    let result;
    {
        let string2 = String::from("xyz");
        result = longest(string1.as_str(), string2.as_str());
    }
    println!("The longest string is {}", result);
}
```

This will not compile as the return value lasts longer than one of the arguments

IO, Reader & Writers



Readers

Files opened using std::fs::File::open(filename)

std::net::TcpStreams, for receiving data over the network

std::io::stdin(), for reading from the process’s standard input stream

std::io::Cursor<&[u8]> and std::io::Cursor<Vec<u8>> values,

which are readers that “read” from a byte array or vector that’s already in memory

Reader Methods

reader.read(&mut buffer: &mut [u8])

Reads bytes into butter

Returns Result<u64, io::Error> - number of bytes read

reader.read_to_string(&mut string)

reader.bytes()

Returns an iterator over the bytes of the input stream

BufReader Methods

reader.read_line(&mut line)

reader.lines()

Returns an iterator over the lines of the input

Example Grep

```
use std::io;
use std::io::prelude::*;

fn grep(target: &str) -> io::Result<()> {
    let stdin = io::stdin();
    for line_result in stdin.lock().lines() {
        let line = line_result?;
        if line.contains(target) {
            println!("{}", line);
        }
    }
    Ok(())
}
```

Sample Reader

```
let a = "this is\nsample input\nin three lines";
let b    = std::io::Cursor::new(a)
    .lines()
    .map(|l| l.unwrap());

println!("{:?}", b);     // Map { iter: Lines { buf: Cursor { inner: "this is\nsample input\nin three lines
                                                                      pos: 0 } } }

for line in b {
    println!("{}", line);
}
```

```
this is
sample input
in three lines
```

```
let a = "12\n51\n4";
let lines = std::io::Cursor::new(a)
    .lines()
    .collect::<std::io::Result<Vec<String>>>()
    .unwrap();
println!("{:?}", lines);
```

```
["12", "51", "4"]
```

```
let a = "12\n51\n4";

let mut input = String::new();

std::io::Cursor::new(a)
    .read_line(&mut input)
    .unwrap();

let n = input
    .trim()
    .parse::<usize>()
    .unwrap();

println!("{}", n);
```

```
12
```

```
let a = "12\n51\n4";
let mut input = String::new();
let b: Vec<i32> = std::io::Cursor::new(a)
    .lines()
    .map(|l| l.unwrap().parse::<i32>().unwrap())
    .collect();
println!("{:?}", b);
```

```
                    ["12", "51", "4"]
```

Runs limited Times - Compiler Enforced

```
#![feature(generic_const_exprs)]
#![allow(incomplete_features, unused_variables)]

struct FnN<const N: usize, F>(F);

impl<const N: usize, F> FnN<N, F> {
    fn new(f: F) -> Self { Self(f) }

    fn call<A, R>(mut self, a: A) -> (FnN<{ N - 1 }, F>, R)
    where
        F: FnMut(A) -> R,
    {
        let res = (&mut self.0)(a);
        let new_fn_n = FnN::<{ N - 1 }, F>::new(self.0);
        (new_fn_n, res)
    }
}
```

```
fn main() {
    let f: FnN<3, _> = FnN::new(|_: ()| println!("hi!"));
    let (f, _) = f.call(());
    let (f, _) = f.call(());
    let (f, _) = f.call(());
    // let (f, _) = f.call(());
   // Uncommenting the line will produce
   //a compile error.
}
```

https://www.reddit.com/r/rust/comments/xelzrv/function_that_can_only_be_called_a_set_number_of/