CSE 410 – Advanced Data Structures
Topic 02: Rust Basics

Oliver Kennedy

Rust

Book: https://doc.rust-lang.org/book/

Standard Library: https://doc.rust-lang.org/std/index.html

Brown U's Interactive Tutorial: https://rust-book.cs.brown.edu/

Tools for Rust Development: https://www.rust-lang.org/tools

Setup

```
$> cargo new [projectname]

$> cd [projectname]

$> cargo run
```

Cargo

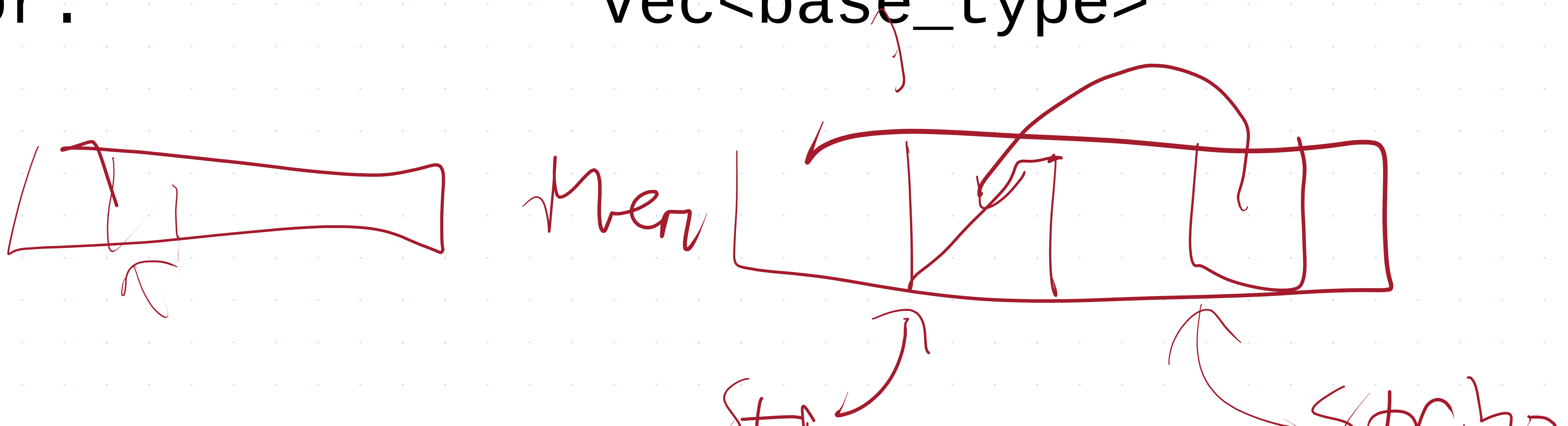
```
# just build
$> cargo build
```

Language Basics

```
// Define an immutable variable
let my_var = "the thing";

// Define a mutable variable
let mut my_var = "the thing";

// Define a function
fn my_fun(arg1: type1, arg2: type_2) -> ret_type
  { /* the function */ }

// Print text
println!("{}: {}", "thing1", my_var);

// Loop
for x in iterable_var { /* loop body */ }
```

<u>Types</u>

Characters:	char
Integers:	i8, i16, i32, i64
Unsigned Integers:	u8, u16, u32, u64
Floating Point:	f8, f16, f32, f64
System-specific:	usize
String (array):	str (.to_string())
String (vector):	String (.as_str())
Array:	[base_type], [base_type; N]
Vector:	Vec<base_type>



Struct

```
struct MyStruct
{
  field_1: type_1,
  field_2: type_2,
  ...
}

// Instantiating:
let foo = MyStruct {
  field_1: "val1",
  field_2: 42,
  ...
}
```

Impl

```
trait MyTrait
{
  fn do_the_thing(&self) -> String
}

impl MyTrait for MyStruct
{
  fn do_the_thing(&self) -> String
  {
    return format!("{}: {}", self.field_1,
                             self.field_2);
  }
}
```

Enum

```
enum MyOptions
{
  Thing1,
  Thing2,
  Thing3WithArgs( arg_1: type_1, ... )
  ...
}
// Branching
match my_options {
  Thing1 => { /* if it's Thing1, do this... */ }
  Thing2 => { /* if it's Thing2, do this... */ }
  Thing3WithArgs(arg_1, ...) =>
  { /* if it's Thing3, do this, using arg_1 ... */
  }
... }
```

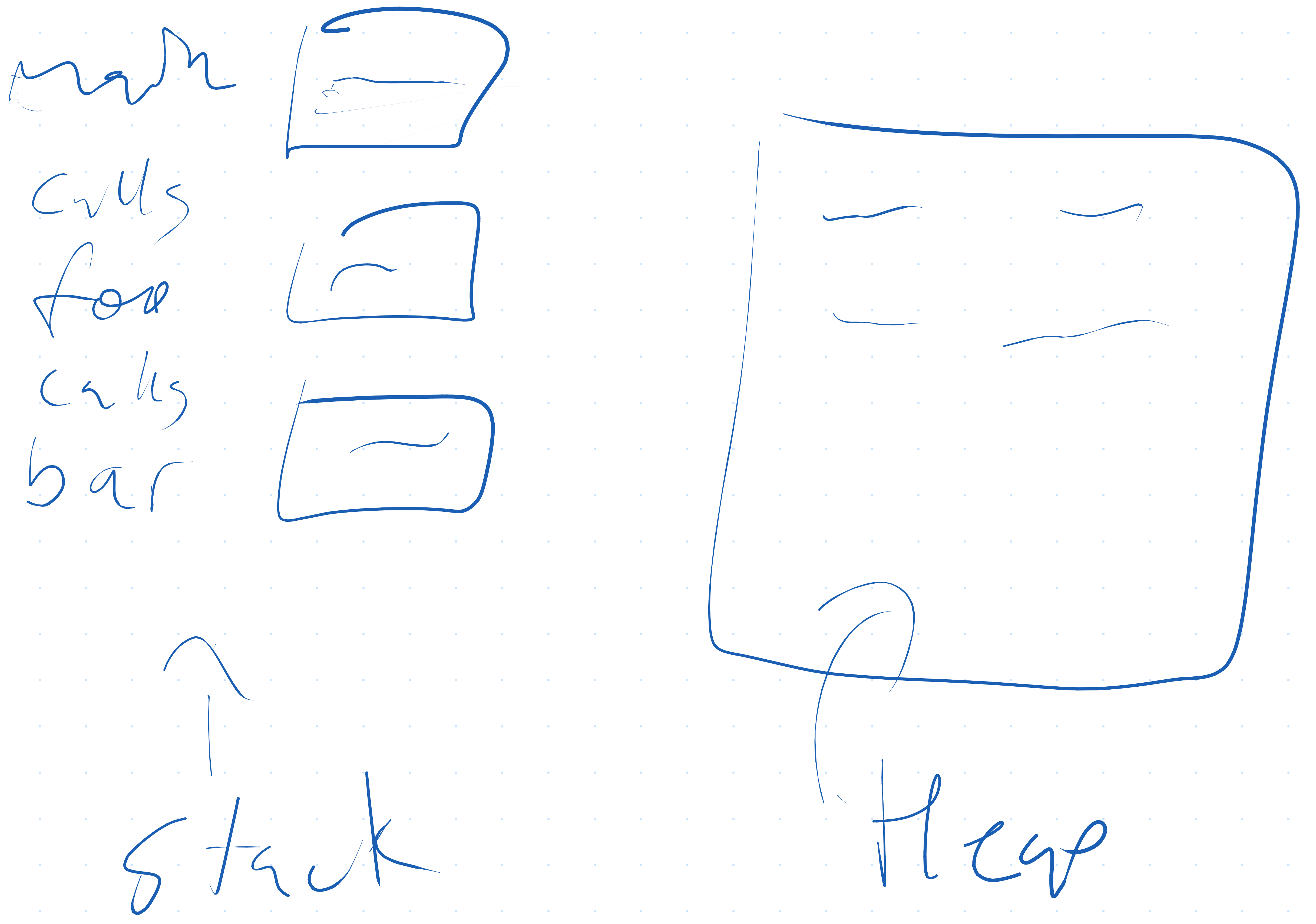
Some/None

```
Option<base_type>

let a_thing = Some("this thing")
let not_a_thing = None

a_thing.unwrap()     // -> "this thing"
not_a_thing.unwrap() // -> runtime error

// Better
match a_thing {
  Some(a_thing) => { /* do the thing */ }
  None          => { /* don't do the thing */ }
}
// or...
a_thing.unwrap_or(...)
```

Result

```
Result<base_type, err_type>

Ok(the_result)
Err(the error)

//Error type can be anything.  Usually, e.g.:
[#derive(Debug, Clone)]
struct MyError
{
  message: str
}
```

The Rust Borrow Checker

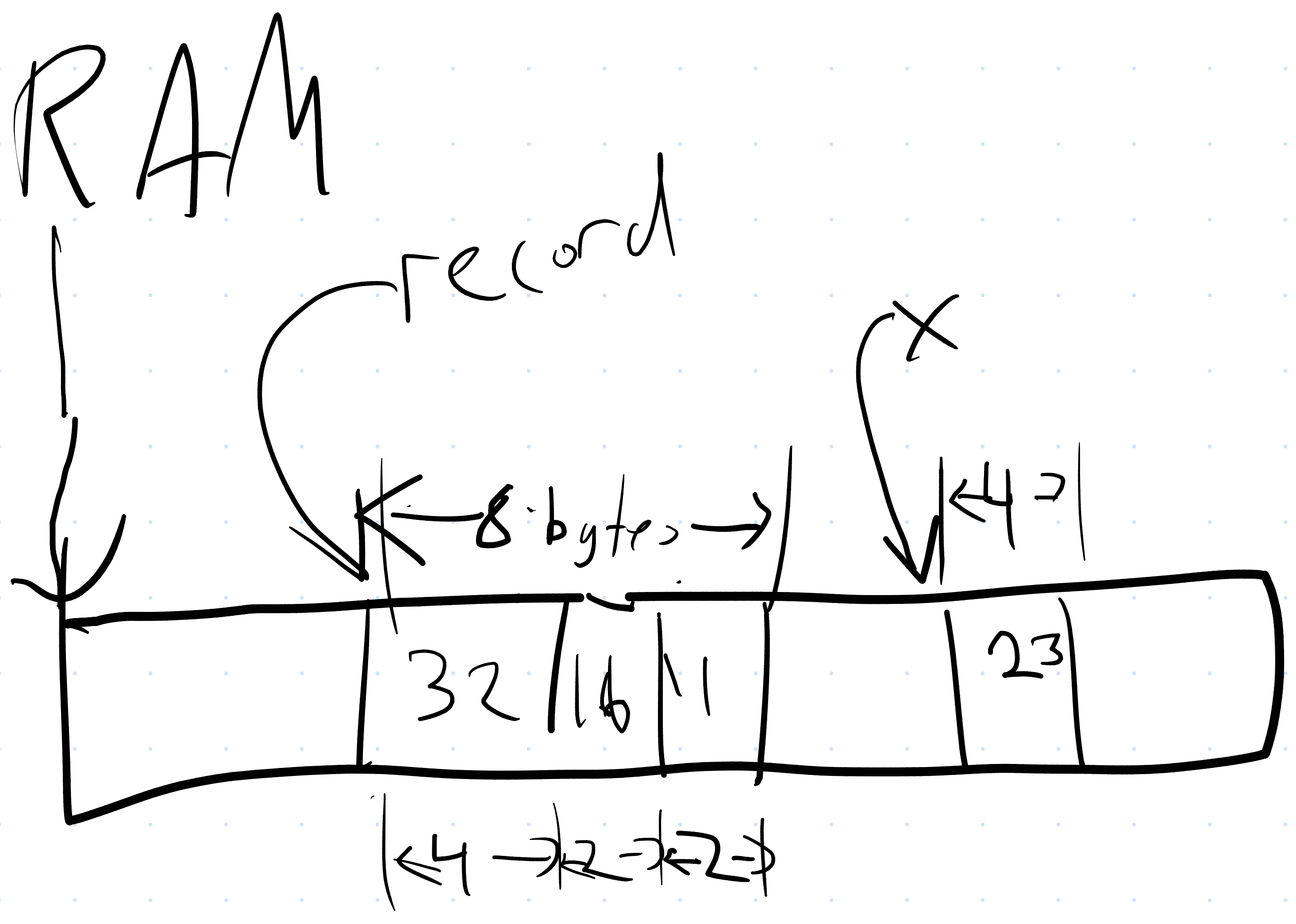


File Serialization

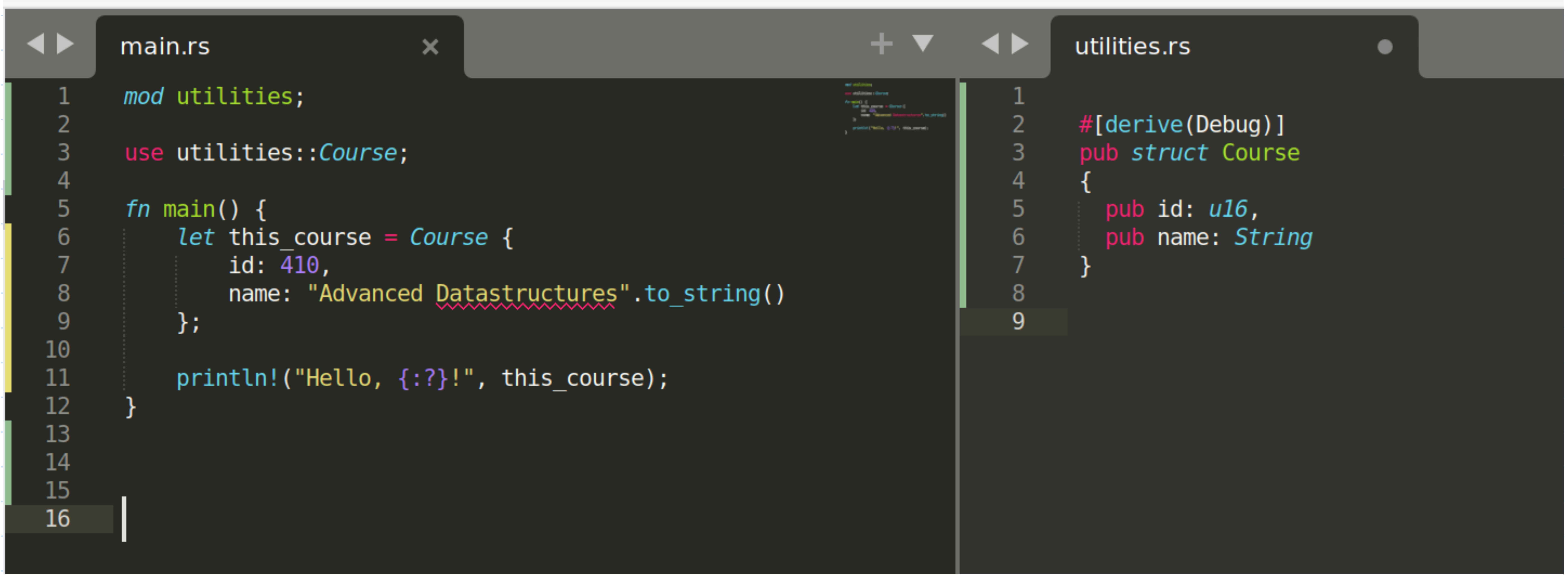
```
Record {
    a: u32,
    b: u16,
    c: u16
}
```

$\rightarrow$

```
let record = Record {
    a: 32
    b: 16
    c: 11
}

let x = 23 as u32
```



[illegible]

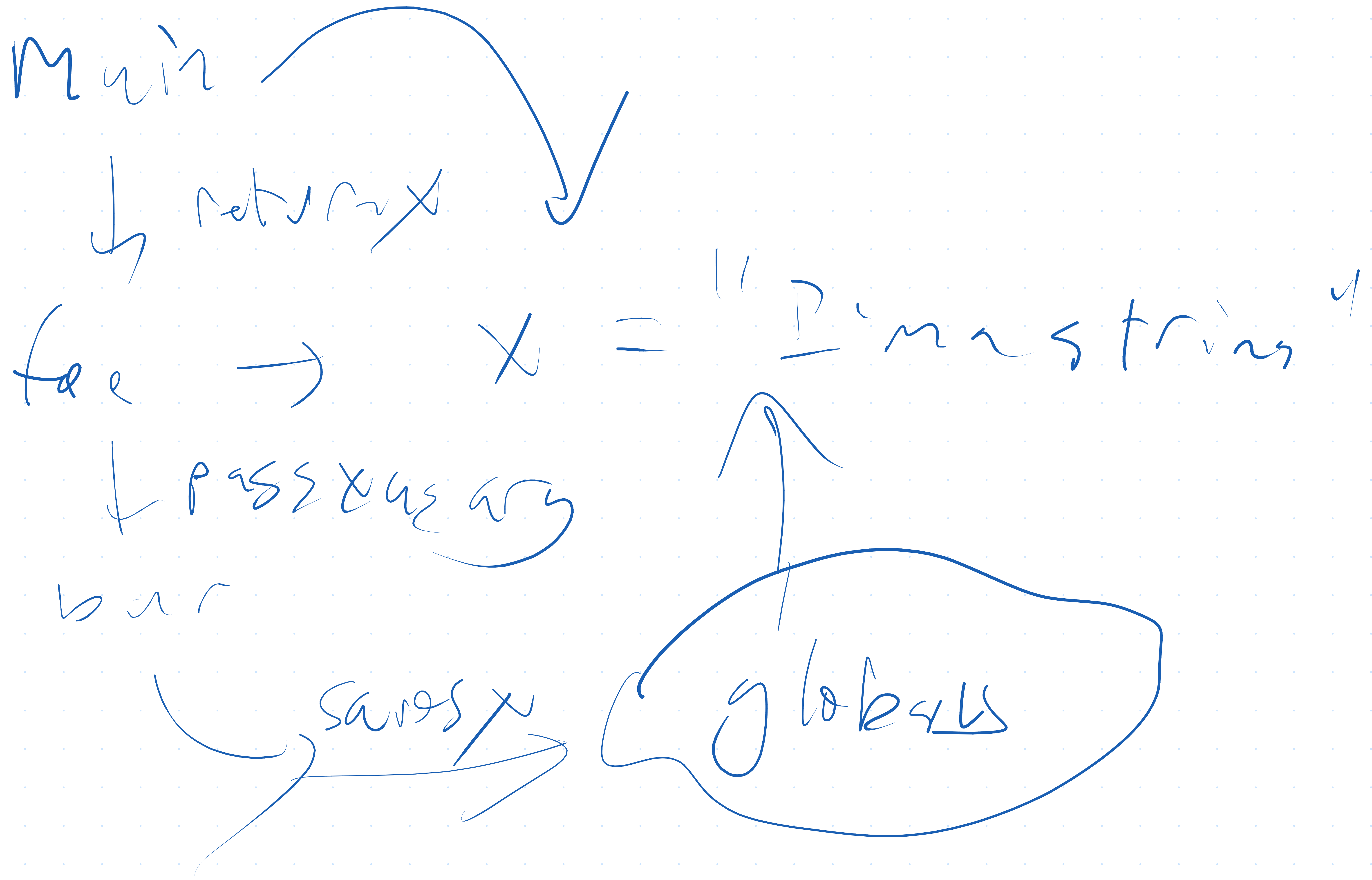


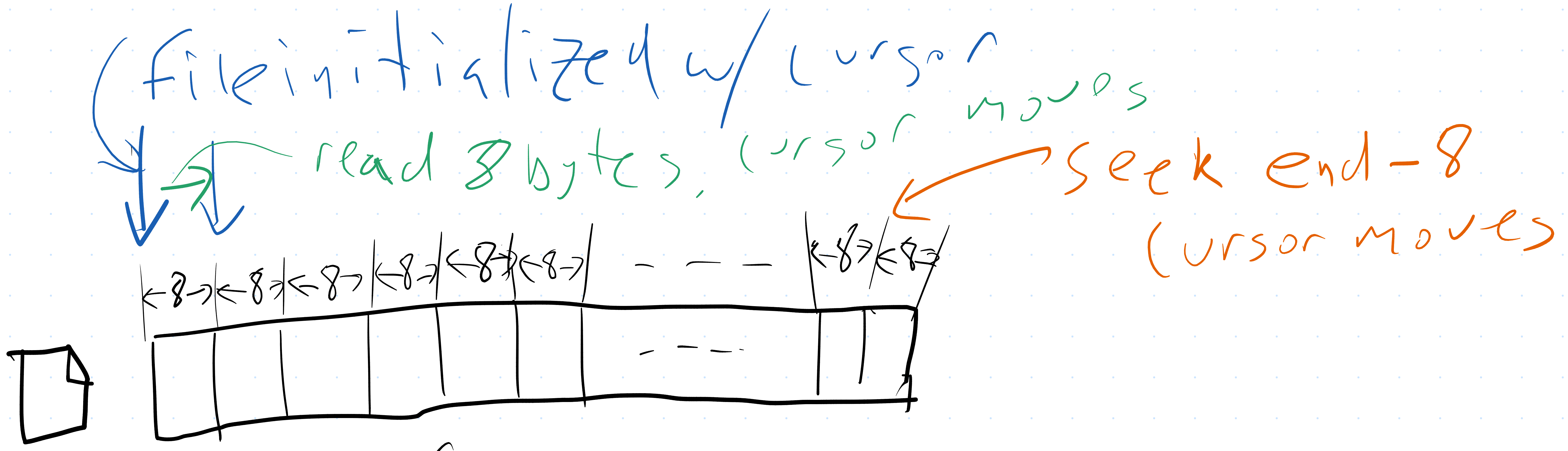
```rust
mod utilities;

use utilities::Course;

fn main() {
    let this_course = Course {
        id: 410,
        name: "Advanced Datastructures".to_string()
    };

    println!("Hello, {:?}!", this_course);
}



```

```rust

#[derive(Debug)]
pub struct Course
{
  pub id: u16,
  pub name: String
}


```

Shorthand

```
fn my_function(foo: str) -> Result<str, Error>
{ return Err(...) }

...

let ret = match my_function("hello world"){
            Ok(result) => result
            Err(err) => return Err(err)
          }

// ... instead write:
let ret = my_function("hello world")?
```


Main
return x
foo
x = "I'm a string"
pass x as arg
bar
saves x
globals



Useful functions for P1

File::open("path") → File

file.metadata() → Metadata

file.seek(to)

file.read_exact(buffer)

```
foo {
  x = "                    "
  bar(x)
  baz(x)
  return x
}
```

Question
Who is responsible for cleaning up x