Collections

Rust, in Practice and in Theory
Lecture 6

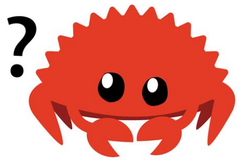
CAS CS 392 (M1)

Outline

Discuss **collections** (very briefly, I imagine these should be pretty familiar)

Workshop: A couple options

Errata: Slices and Borrowing

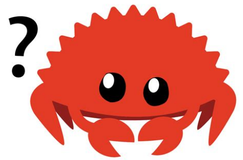



```
fn main() {
    let mut s = String::from("long string");
    let a : &mut str = &mut s[1..4];
    let b : &mut str = &mut s[3..8];
    a.make_ascii_uppercase();
    b.make_ascii_lowercase();
    println!("{}", &s) // prints: LONg string
}
```

You **cannot** have multiple overlapping slices...

Aside: Flow Sensitivity

```
fn f() -> i32 {
    let mut x = 1;
    let y = &x;
    x = x + 1;
    x + *y
}
```



```
fn f() -> i32 {
    let mut x = 1;
    let y = &x;
    x = x + 1;
    x
}
```

Borrow Checking is *flow sensitive*. The type of a variable *changes* according to its position in control flow.

Type checking is usually *flow insensitive*. The position of a term in an expression does not affect it's type, only the type of the super-expression.

my apologies...

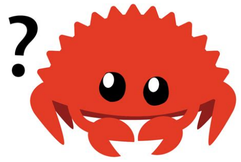
Vectors

```
let v: Vec<i32> = Vec::new();    // creating a new vector
let  mut v = vec![1, 2, 3];      // from array shorthand
v.push(5);                       // append to end
let x: Option<i32> = v.pop();    // removing from end
let x: &i32 = &v[2];             // unsafe indexing
let x: Option<&i32> = v.get(2); // safe indexing
```

A **vector** is a contiguous collection of data in memory

They have the usual methods (check the docs)

Vectors and Borrowing




```
let first = &v[0];
v.push(6);
let x = first;
```

A reference to an element in a vector counts as a borrow of the *entire* vector

(Apologies again for mixing this up in the case of slices)

Iteration

We can iterate over vectors in the usual way

(note the dereference operator *)

Why don't we iterate by index?

```
let mut x = 0;
for i in &v {
    x += i
}
for i in &mut v {
    *i += 10
}
```

Question

Can we iterate over a vector that might be updated intermittently?

Strings

```
let hello = String::from("السلام عليكم");
let hello = String::from("Dobrý den");
let hello = String::from("Hello");
let hello = String::from("שלום");
let hello = String::from("नमस्ते");
let hello = String::from("こんにちは");
let hello = String::from("안녕하세요");
let hello = String::from("你好");
let hello = String::from("Olá");
let hello = String::from("Здравствуйте");
let hello = String::from("Hola");
```

Strings are complicated...

We're not going to worry about it too much...

Hash Maps

```
use std::collections::HashMap;
let mut h : HashMap<String,i32> = HashMap::new(); // create
h.insert(String::from("ten"), 10);                // insert (moves values into h)
let x : Option<&i32> = h.get("ten");              // access (does not consume key)
```

The standard library also has hash maps with the usual interface

Note that insertion moves values whereas accessing does not

(See the docs for more examples)

Workshop

Pair programming: Word counter (where a word is a contiguous sequence of non-whitespace characters)

Complete assignment 2

Crash course on forth (https://skilldrick.github.io/easyforth/)

gforth manual (https://gforth.org/manual/)

Read about linked lists (https://rust-unofficial.github.io/too-many-lists/index.html)