Discussion #8

10/28/2026

Today

Rust fundamentals

Reminder: Project 2 is due this week, and we will have code reviews again next week!

Today we will:

- Review how strings are stored in memory
- Identify string-related errors in code
- Practice drawing stack-heap diagrams

Strings Review

The different types of strings

- **String:** Can be owned; initialized using **String::from()** or **.to_string()**, stored as a pointer on the stack + bytes allocated on the heap
- **&str:** A borrowed string/string literal. Stored as a pointer with a length
- By default, simply declaring a variable with the string in quotes makes a &str type object

Strings Review

Strings are UTF-8

- A String is a sequence of bytes, and each char can take up between 1 and 4 bytes
- len() counts bytes, not characters. A char with an accent, like "é", takes up 2 bytes, and emojis can take up 4
- Slicing &s[a..b] uses byte offsets and panics if it cuts a character in half
- Because of this, s[i] isn’t allowed. Which byte or char would it return?
- .chars() lets you access individual characters without byte worries

What's wrong with this code?

```
let fruits: [String] = ["apples", "oranges", "lemon"];
println!("{:?}", fruits);
```

Type mismatch; by default, “x” creates &str, not String

```
let fruits: [String] = ["apples", "oranges", "lemon"];
println!("{:?}", fruits);
```

```
  |
3 |     let fruits: [String] = ["apples", "oranges", "lemon"];
  |                            ^^^^^^^^ expected `String`, found `&str`
  |
help: try using a conversion method
  |
3 |     let fruits: [String] = ["apples".to_string(), "oranges", "lemon"];
  |     Define as &str here ^ or do this ------> ++++++++++++
```

Why can’t we do this?

```
let mut fruit: String = "Bapple".to_string();
fruit[0] = "G";
```

Why can’t we do this?

```
let mut fruit: String = "Bapple".to_string()
let sliced = &fruit[0];
```

Can't directly index into a String object
Follow up: why?

```
let mut fruit: String = "Bapple".to_string()
let sliced = &fruit[0];
```

error[E0277]: the type `str` cannot be indexed by `{integer}`

```
let mut fruit: String = "Bapple".to_string()
let sliced: &str = &fruit[0..1];
// let sliced: Option<char> = fruit.chars().next(); works too
```

What's wrong with this code?

```rust
// Return the first character in a string, if there is one

pub fn get_first(message: &str) -> Option<char>{
    let first: char = message.chars().next().expect("Empty String!");
    Some(first)
}

fn main() {
    let fruit: String = "apple".to_string();
    println!("{}", get_first(fruit));
}
```

Passed a String when expecting a &str
Need to unwrap first!

```
// Return the first character in a string, if there is one

pub fn get_first(message: &str) -> Option<char>{
    let first: char = message.chars().next().expect("Empty String!");
    Some(first)
}

fn main() {
    let fruit: String = "apple".to_string();
    println!("{}", get_first(fruit));
}
```

Expects &str!

.unwrap()

This compiles correctly but fails at runtime. Why?

```rust
pub fn compare_encodings(bytes: [u8; 2]) {
    let utf8: &str = std::str::from_utf8(&bytes).unwrap();
    let mut latin1: String = "".to_string();

    for b: u8 in bytes {
        latin1 += &(b as char).to_string();
    }

    assert_eq!(utf8, latin1);
}

▶ Run | ⚙ Debug
fn main() {
    let bytes: [u8; 2] = [0xC3, 0xA9];
    compare_encodings(bytes);
}
```

Output to terminal:

```
   Compiling crew_manifest v0.1.0 (C:\Users\kr
    Finished `dev` profile [unoptimized + debu
     Running `target\debug\crew_manifest.exe`

thread 'main' (51480) panicked at src\main.rs:
assertion `left == right` failed
  left: "é"
 right: "Ã©"
```

This compiles correctly but fails at runtime. Why?

```rust
pub fn compare_encodings(bytes: [u8; 2]) {
    let utf8: &str = std::str::from_utf8(&bytes).unwrap();
    let mut latin1: String = "".to_string();

    for b: u8 in bytes {
        latin1 += &(b as char).to_string();
    }

    assert_eq!(utf8, latin1);
}

▶ Run | ⚙ Debug
fn main() {
    let bytes: [u8; 2] = [0xC3, 0xA9];
    compare_encodings(bytes);
}
```

Comparing 2 utf-8 characters 0xC3, 0xA9 individually vs [0xC3 0xA9] = é

Output to terminal:

```
   Compiling crew_manifest v0.1.0 (C:\Users\kr
    Finished `dev` profile [unoptimized + debu
     Running `target\debug\crew_manifest.exe`

thread 'main' (51480) panicked at src\main.rs:
assertion `left == right` failed
  left: "é"
 right: "Ã©"
```

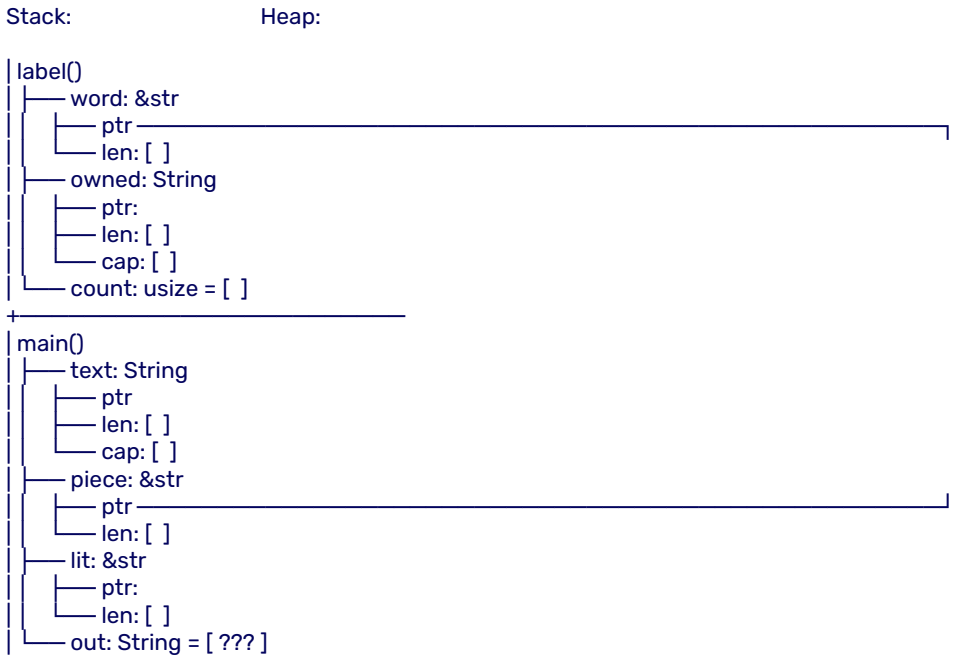
Stack & Heap Diagram Practice

Get out your notebooks & talk to your neighbors!

There will be two problems. Things to remember:

- Draw the stack on the left and the heap on the right as they exist during DRAW HERE
- Build the stack from the bottom up, one solid box per function frame, labeled with the function's name.
- A frame holds **every** variable its function declares, from the moment it is called. Write ??? for any that has no value yet (it should still have a name and type), and (moved) (or x out the pointer/value) for any whose value has moved
- Label every variable with its **type** and any other metadata. A String or Vec has len and cap and a slice has len in addition to pointers.
- Draw an arrow from each pointer to the thing it points at.

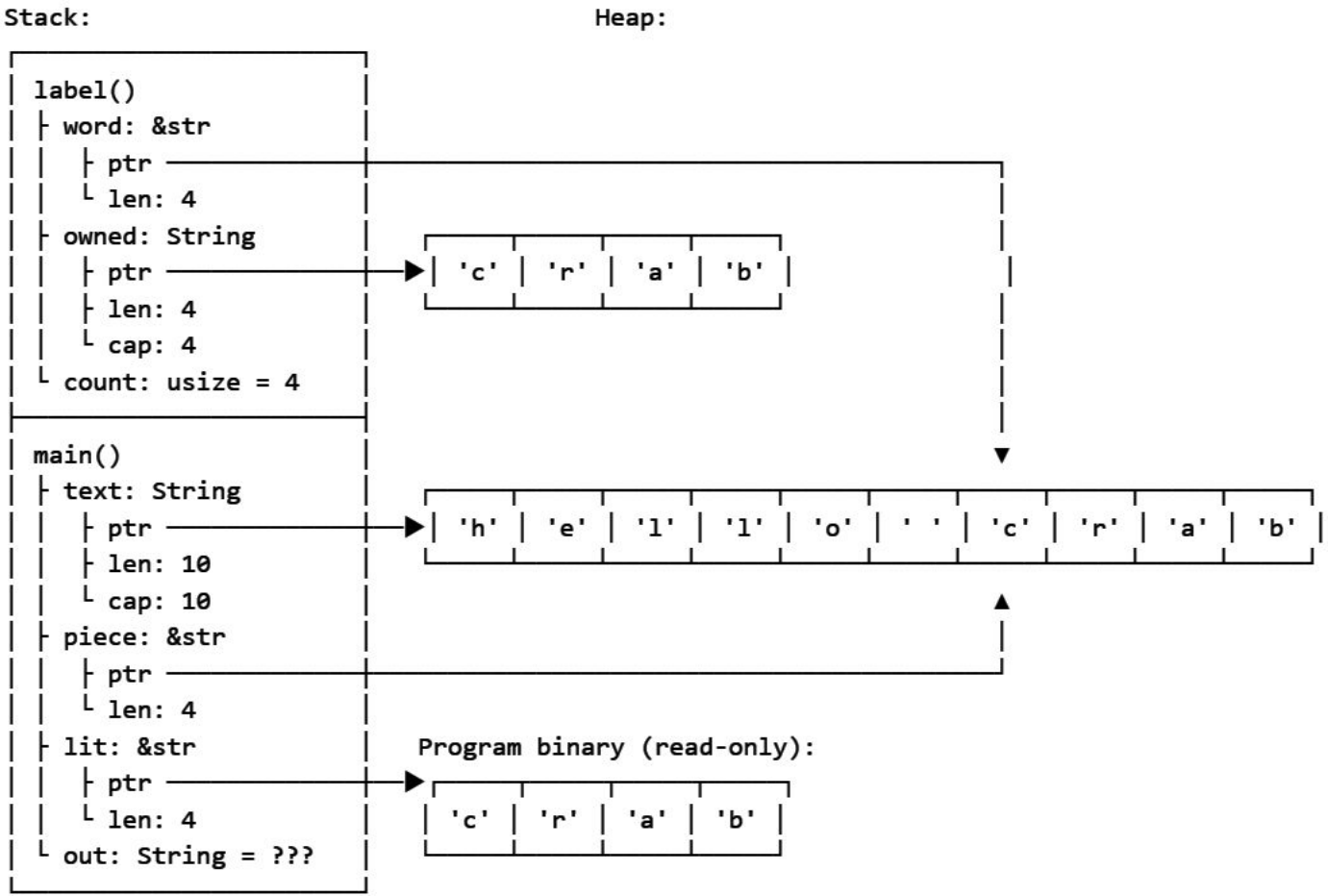
```
fn label(word: &str) -> String {
    let owned = word.to_string();
    let count = word.len();
    // DRAW HERE
    println!("{} has {} bytes", owned, count);
    owned
}

fn main() {
    let text = String::from("hello crab");
    let piece: &str = &text[6..10];
    let lit: &str = "crab";
    let out = label(piece);
    println!("{} | {} | {}", text, out, lit);
}
```



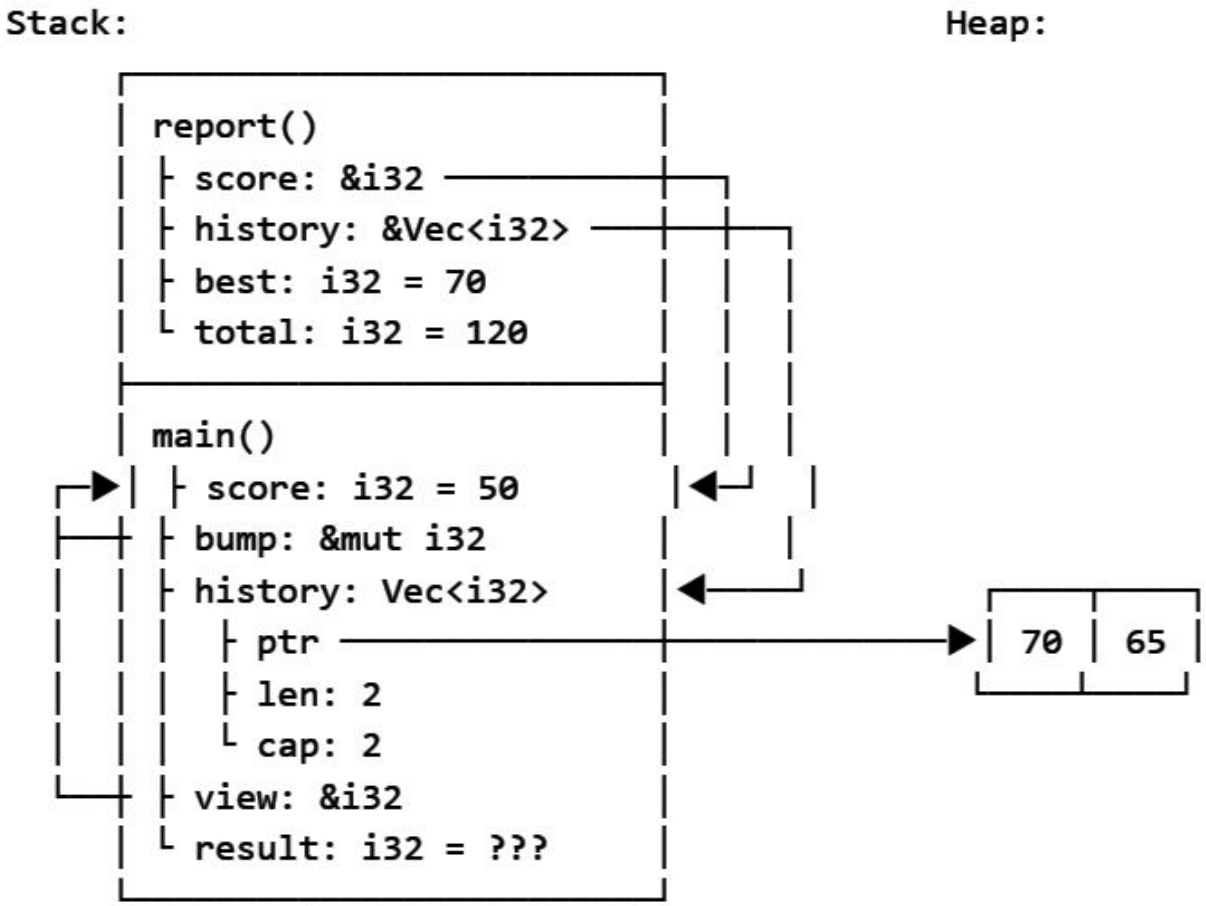
Heap Allocation 1 (`text`):
Buffer : ['h', 'e', 'l', 'l', 'o', ' ', 'c', 'r', 'a', 'b']
Bytes : 0 1 2 3 4 5 6 7 8 9

Heap Allocation 2 (`owned`):
Buffer : [, , ,]
Bytes : 0 1 2 3

Static Memory (`lit`):
Buffer : ['c', 'r', 'a', 'b']
Bytes : 0 1 2 3


Stack:
Heap:
label()
├ word: &str
│ ├ ptr
│ └ len: 4
├ owned: String
│ ├ ptr
│ ├ len: 4
│ └ cap: 4
└ count: usize = 4
main()
├ text: String
│ ├ ptr
│ ├ len: 10
│ └ cap: 10
├ piece: &str
│ ├ ptr
│ └ len: 4
├ lit: &str
│ ├ ptr
│ └ len: 4
└ out: String = ???
'c' 'r' 'a' 'b'
'h' 'e' 'l' 'l' 'o' ' ' 'c' 'r' 'a' 'b'
Program binary (read-only):
'c' 'r' 'a' 'b'

```
fn report(score: &i32, history: &Vec<i32>) ->
i32 {
    let best = history[0];
    let total = *score + best;
    // DRAW HERE
    total
}

fn main() {
    let mut score = 40;
    let bump = &mut score;
    *bump += 10;
    let history = vec![70, 65];
    let view = &score;
    let result = report(view, &history);
    println!("{}", result);
}
```


Stack:
Heap:
report()
├ score: &i32
├ history: &Vec<i32>
├ best: i32 = 70
└ total: i32 = 120
main()
├ score: i32 = 50
├ bump: &mut i32
├ history: Vec<i32>
├ ptr
├ len: 2
└ cap: 2
├ view: &i32
└ result: i32 = ???
70
65

Practice quiz – Gimkit!

https://www.gimkit.com/view/6ae027d24750d41039ebd153

Any Questions?