Lecture 7

Borrowing and Slices in Rust

Slides: Neil Kaushikkar

Goals For Today

- Answering Your Questions
- Review Ownership & Borrowing
- Dereferencing Mutable References
- Slices of **String**s and **Vec**tors
- Review MP0

Reminders

- HW5 releasing tonight due 2/23 at 11:59 pm CT
- HW4 due 2/17 at 11:59 pm CT
- MP1 due 2/23 at 11:59 pm CT

Answering Your Questions!

- “The ownership stuff is pretty confusing, I know you went over it already but if you could continue incorporating this concept into the lessons it would be appreciated.”
  - Ownership is central to Rust
  - Everything we’ll be covering will be taught through the lens of ownership
  - **struct**s, multithreading, functional programming/iterators, etc…

Answering Your Questions!

- “When we use **enum**s in a **match** statement for a variable, does this variable have to be an **enum** type too?”

Brief Matching Example

Answering Your Questions!

- “Under what circumstances should we use **"{:?}"** in string formatting? (like what types need **"{:?}"** rather than **"{}"**”
  - General Rule of Thumb:
    - use **"{}"** for primitive types and **String**
    - Use **"{:?}"** for custom types: data structures (**Vec**), other **struct**s, **enum**s, etc...
  - In reality, you can use **"{}"** for some specific types depending on their implementation. More on this when we get to **trait**s.
    - **"{:?}"** used for debugging
    - **"{}"** used for pretty printing

Answering Your Questions!

- “Why do we have to dereference the variables in function 2? Is this like in regular CS 128”
- “It's not obvious from the slides that it's necessary to dereference mutable references to modify them”
    - Similar to pointers: you have to dereference the reference to change the data it refers to
    - Syntactically similar to dereferencing pointers in C++

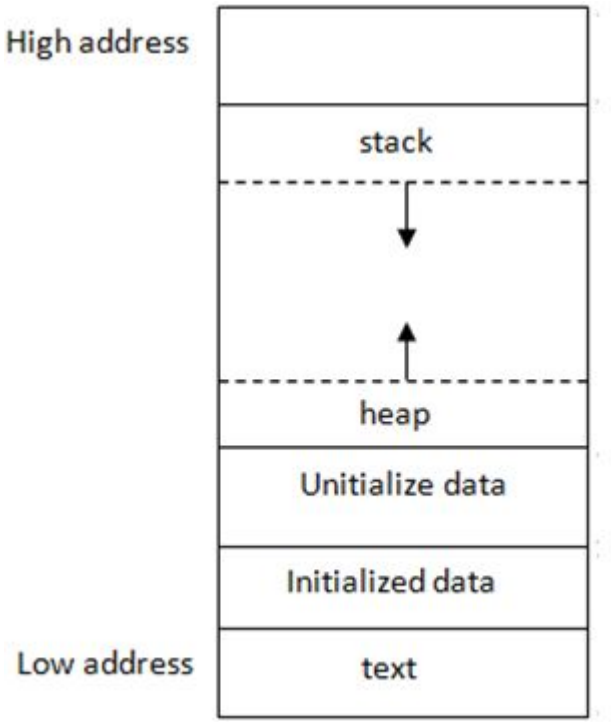
Dereferencing Example

Answering Your Questions!

- "I'm still a little confused on the difference between **String** and **&str**?"
  - **&str**:
    - Reference to a string literal
    - Slice of a **String**
  - **String**
    - Custom type
    - Has ownership over its characters

```
fn main() {
    let ref = "hello world!";

    let string = String::from("testing123");
}
```



Reference:
- https://courses.engr.illinois.edu/cs225/sp2020/resources/stack-heap/

Ownership Review

- Each value in Rust has a variable that's called its *owner*
- There can only be one owner at a time
- When the owner goes out of scope, the value will be dropped

```rust
fn main() {
    let s = String::from("hello");
    // ...
    {
        let w = String::from("world");
        // do something with w...
    }   // w is dropped here
    // ...
}   // s is dropped here
```

```rust
fn main() {
    let x = String::from("hello");

    let y = x; // y now OWNS the String "hello"

    // println!("{}", x); // THIS LINE WON'T COMPILE
    println!("{}", y);
}
```

Reference:

- https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html

References Review

- An ampersand (&) represents a *reference*
- Allows you to refer to some value without taking *ownership* of it
- We call the action of creating a reference *borrowing*

Reference:

- https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html

Borrowing Review

- At any given time, you can have either:
  - one mutable reference using **&mut** or...
  - An infinite number of immutable references using **&**

```rust
fn main() {
    let mut x: String = String::from("hello");

    // creates a MUTABLE reference to x
    let y = &mut x;

    // ERROR: trying to create a SECOND MUTABLE reference to x
    x.push_str(" world!");

    println!("x = {} and y = {}", x, y);
}
```

Reference:

- https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html

String Slices

- The **String** type has ownership over its characters
- If we wanted to get a substring, we can take a slice:
  - A *string slice* is a reference to a portion of a **String**
  - The original string still has ownership of the **char**s

```
fn main() {
    let s1 = String::from("hello");
}
```

s1

name	value
ptr	
len	5
capacity	5

index	value
0	h
1	e
2	l
3	l
4	o

Reference:

- https://doc.rust-lang.org/book/ch04-03-slices.html

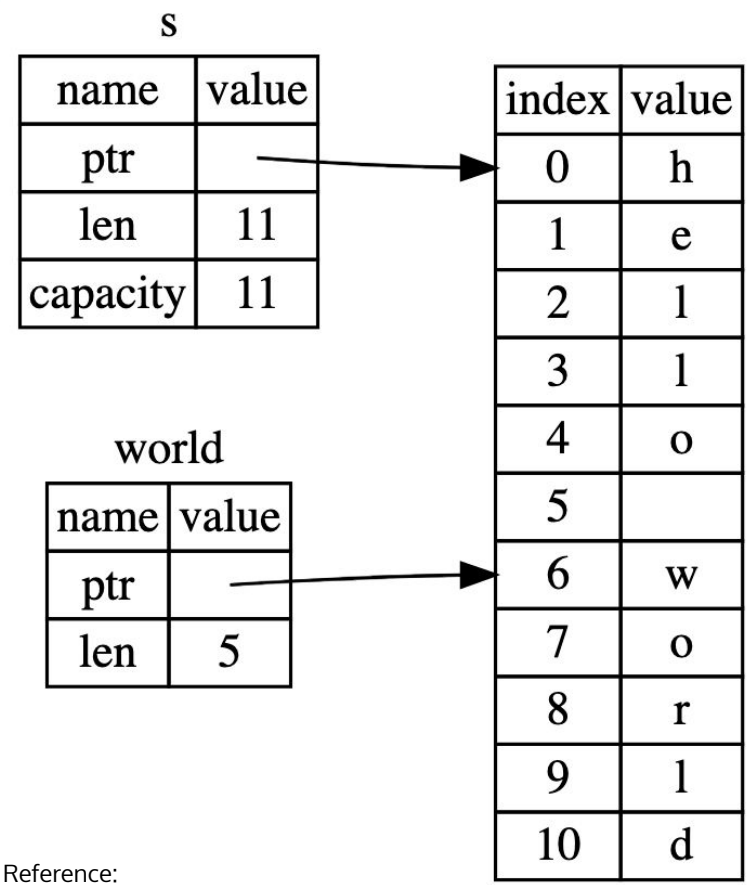
Creating String Slices

- Use & to create a reference and specify a range
  - [*start*..*stop*] - index *start* (inclusive) to *stop* (exclusive)
  - [..*stop*] - index 0 to *stop* (exclusive)
  - [*start*..] - index *start* (inclusive) to the end of the **String**
  - [..] - index 0 to the end of the **String**
- Slices are **READ-ONLY** (aka immutable)

```
let s = String::from("hello world");

let hello = &s[0..5];  // same as &s[..5]
let world = &s[6..11]; // same as &s[6..]
let hello_world = &s[..];
```

String Slices Under the Hood




```rust
let s = String::from("hello world");

let hello = &s[0..5];
let world = &s[6..11];
```

Reference:

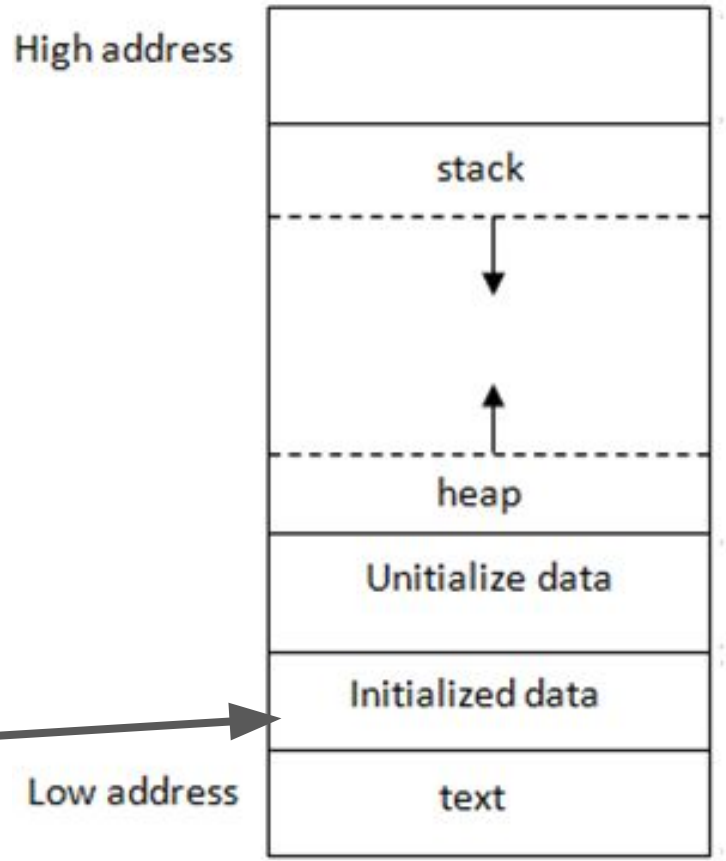
- https://doc.rust-lang.org/book/ch04-03-slices.html

String Literals in Memory

```
let world: &str = "world";
```

world

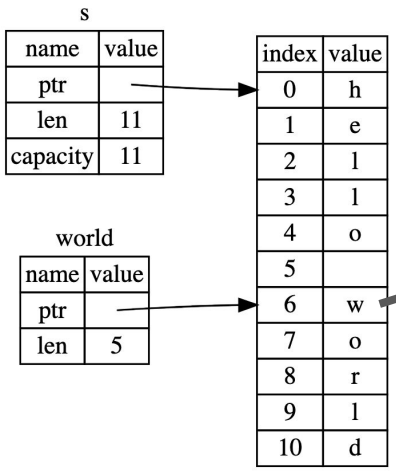
name	value
ptr	
len	5

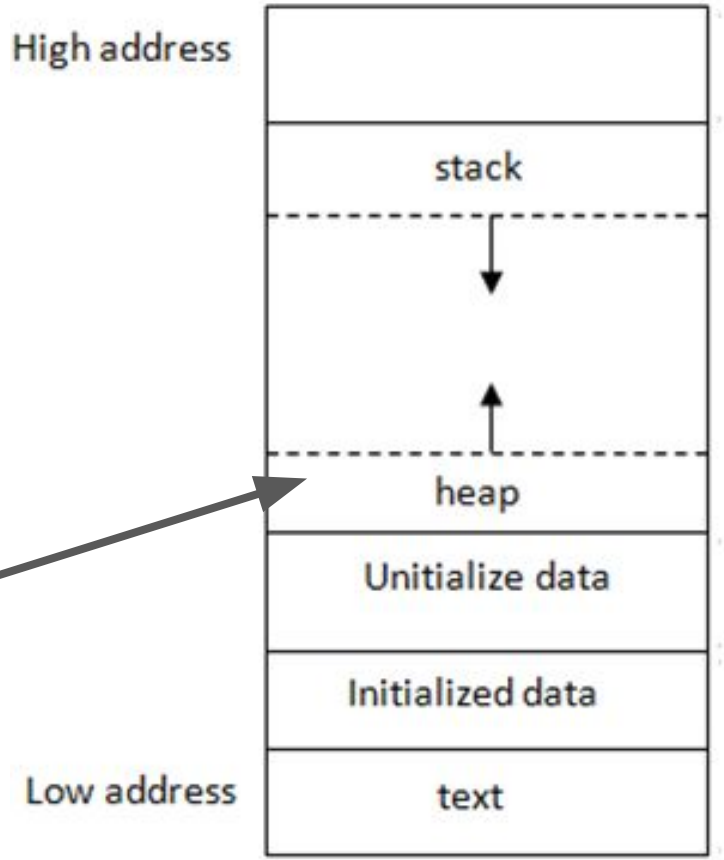


String Slices in Memory

```
let s = String::from("hello world");

let hello = &s[0..5];
let world = &s[6..11];
```





Slices Example

Vector Slices

- Constructed the same way as a **String** slice
  - Borrow the original vector
  - Specify a range with the [*start*..*stop*] notation
- Again, slices are **READ-ONLY** (aka immutable)

Review MP0