CS128 Honors

Lecture 7

Borrowing and Slices in Rust

Slides: Neil Kaushikkar

Goals For Today

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

Reminders

- HW5 releasing tonight due 9/27 at 11:59 pm CT
- HW4 due 9/22 at 11:59 pm CT
- MP1 due 9/28 at 11:59 pm CT

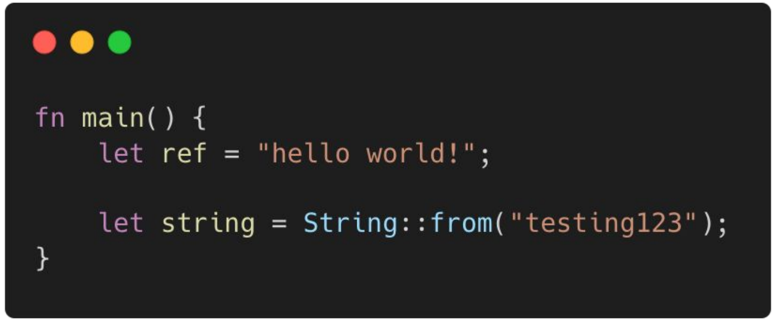
- From now on, you can work on homeworks in groups of **up to 3**
- If you work with a group, put a comment at the top of your file with the NetIDs of your partners so we know similar solutions are from the same group
- All partners **must** submit the assignment on PrairieLearn
- Feel free to use the **#team-building** channel in Discord to form groups

Answering Your Questions!

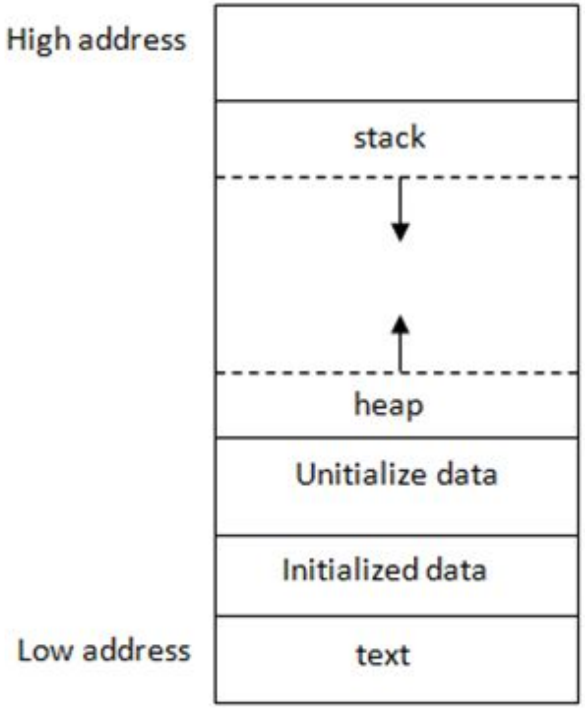
- "Please, please, PLEASE go over **&str** and **String** since I had to search up the methods to convert between them since the main errors I kept having in this and the past homework were just concerning those."
  - This one's on us
  - Difficult to fully grasp the nuances without knowing ownership, but we should have introduced the API sooner…
  - This entire lecture will be about **&str** and **String**

Answering Your Questions!

- "Please, please, PLEASE go over **&str** and **String** since I had to search up the methods to convert between them since the main errors I kept having in this and the past homework were just concerning those."
  - **&str**:
    - Reference to a string literal
    - Slice of a **String**
  - **String**



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

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



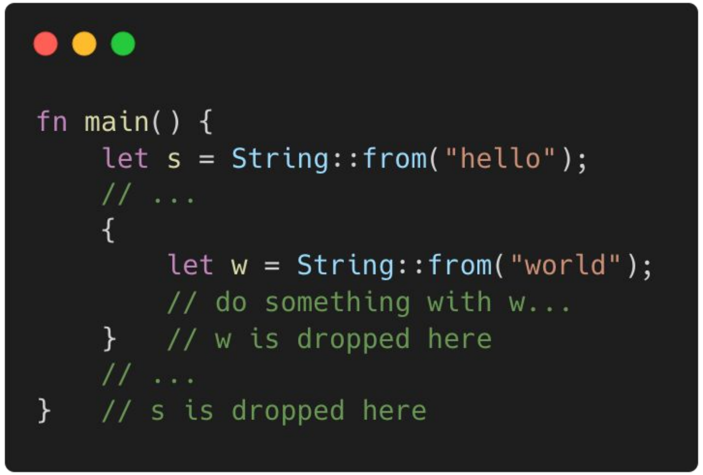
Reference:
- https://courses.engr.illinois.edu/cs225/sp2020/resources/stack-heap/

Answering Your Questions!

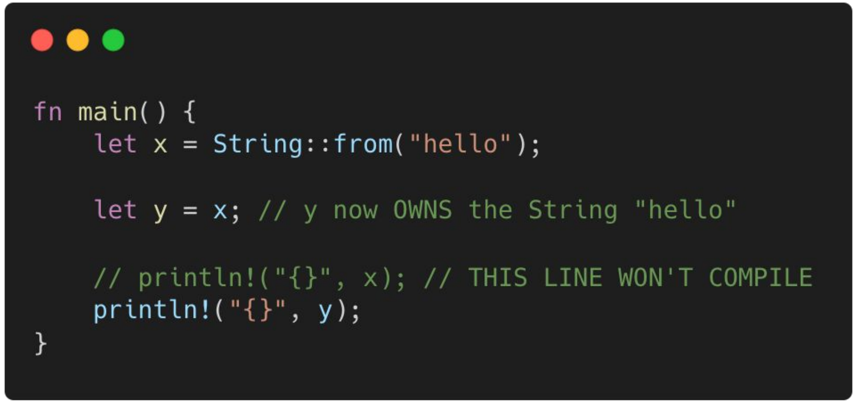
- “borrowing and owning is confusing lol”
    - Borrowing and ownership is VERY confusing (and annoying)
    - Everything we’ll be covering will be taught through the lens of ownership
    - **structs**, multithreading, functional programming/iterators, etc…
    - There will be plenty of examples of how ownership comes into play throughout the remaining lectures

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

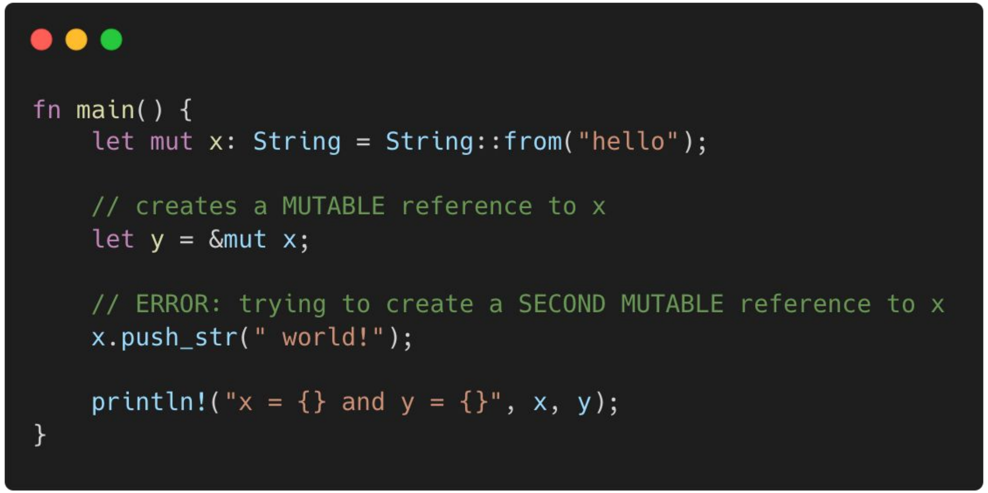
- An ampersand (&) represents a *reference*
- Allows you to refer to some value without taking *ownership* of it
- We call the action of creating a reference *borrowing*

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 **&**

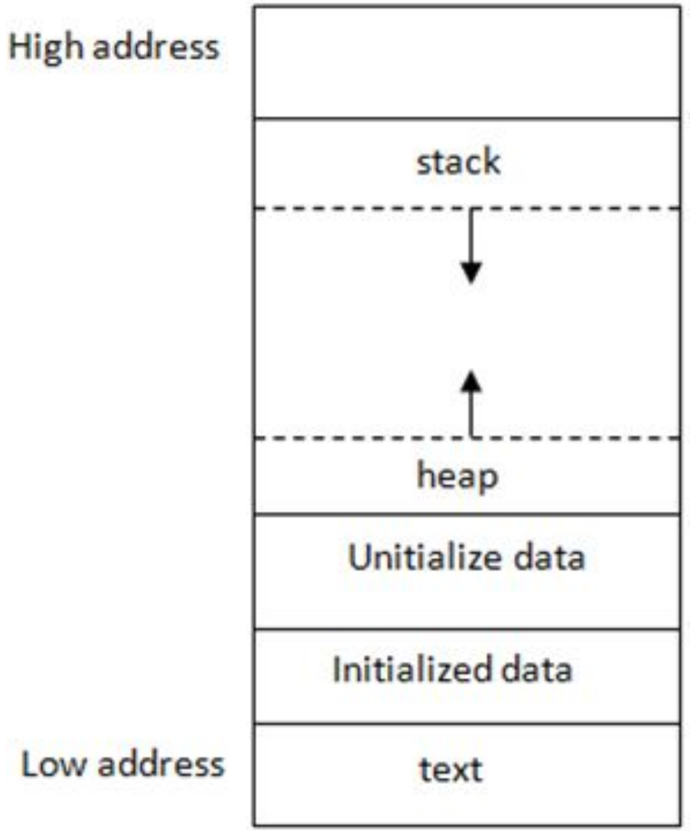


```
fn main() {
    let mut x: String = String::from("hello");

    // 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

Anatomy of a Program's Memory

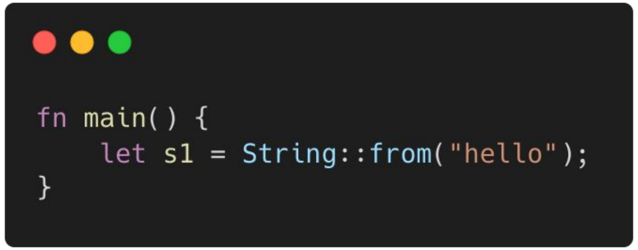


Reference:

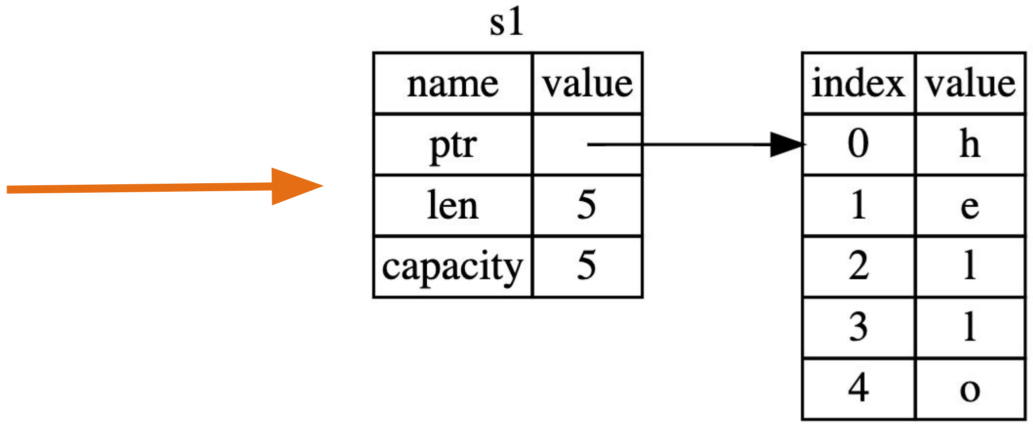
- https://courses.engr.illinois.edu/cs225/sp2020/resources/stack-heap/

Strings and Substrings

- The **String** type has ownership over its characters
- If we wanted to get a substring, we would like:
  - Some type of reference to a portion of the original **String** (to avoid duplicating out **String** data)
  - The original string to keep ownership of its **chars**



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



s1

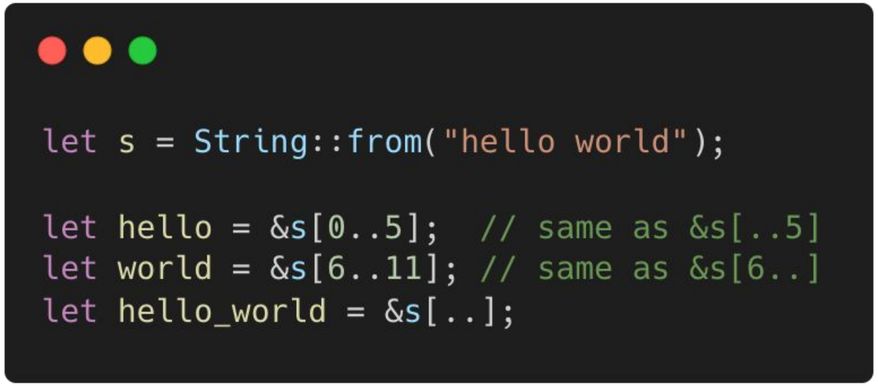
name	value
ptr	
len	5
capacity	5

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

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

Enter 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* (**&str**) is a reference to a portion of a **String**
  - This reference can be of substring or the ENTIRE string – it's a reference!
  - The original string still has ownership of the **char**s



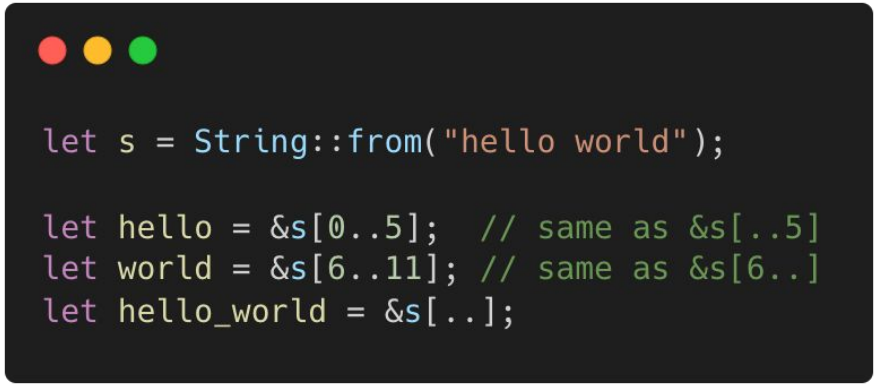
```
let s = String::from("hello world");

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

Reference:

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

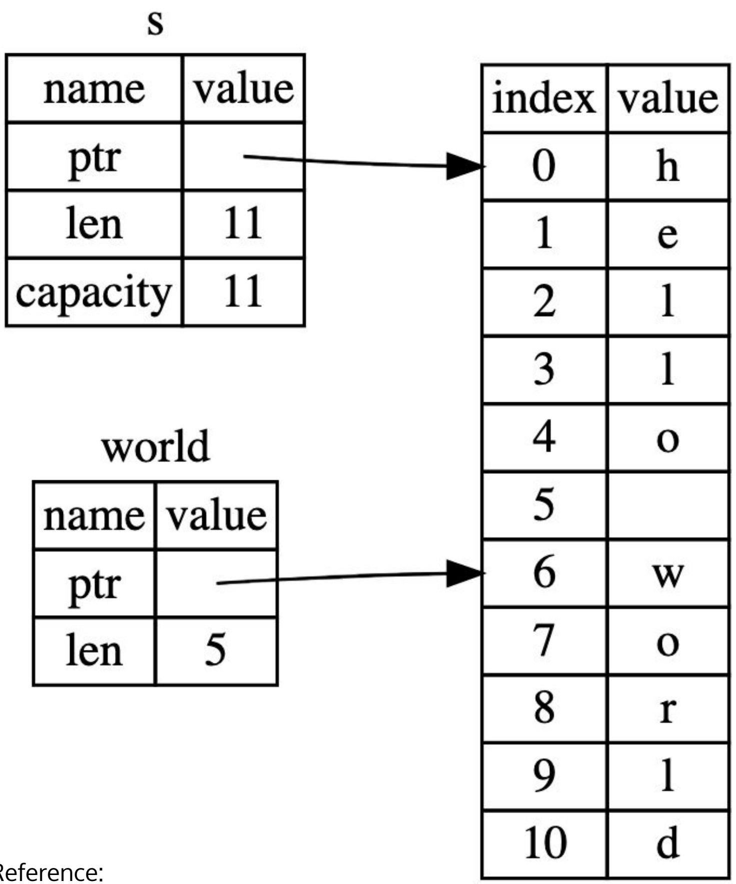
Creating String Slices

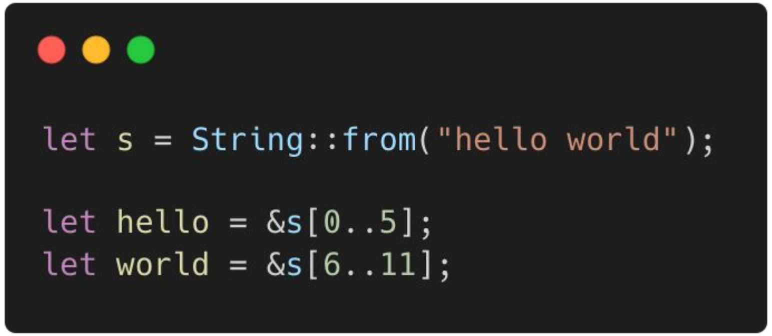
- Use & to create a reference and specify a range
    - [*start*..*stop*] - index *start* (inclusive) to *stop* (exclusive)
    - [..*stop*] - index 0 to *stop* (exclusive)
    - [*start*..] - index *start* (inclusive) to the end of the **String**
    - [..] - index 0 to the end of the **String**
- Slices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



```rust
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



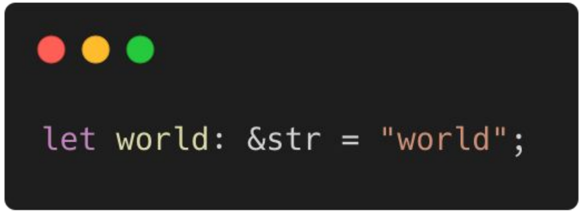


```rust
let s = String::from("hello world");

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

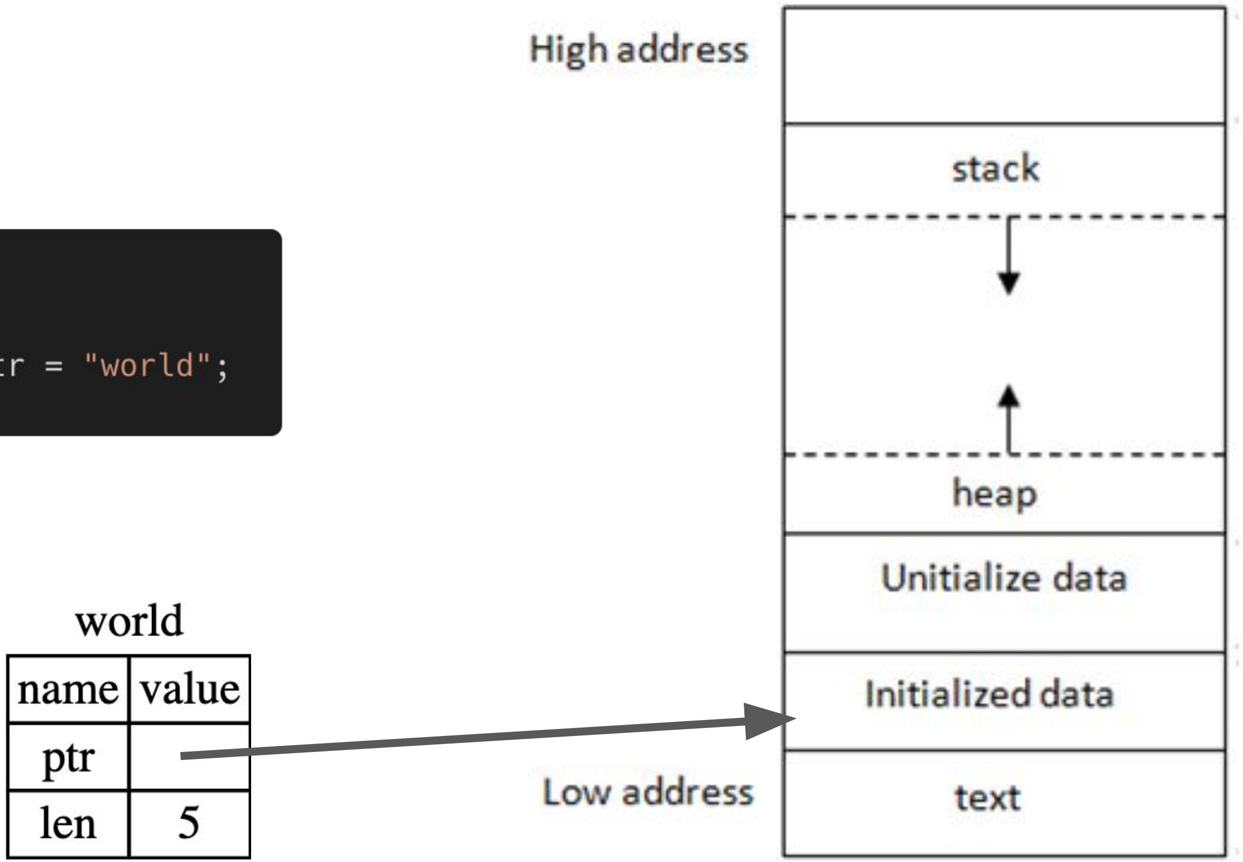
Reference:

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

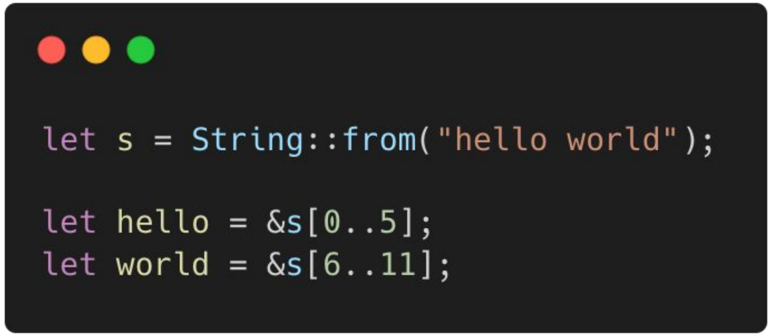
String Literals in Memory



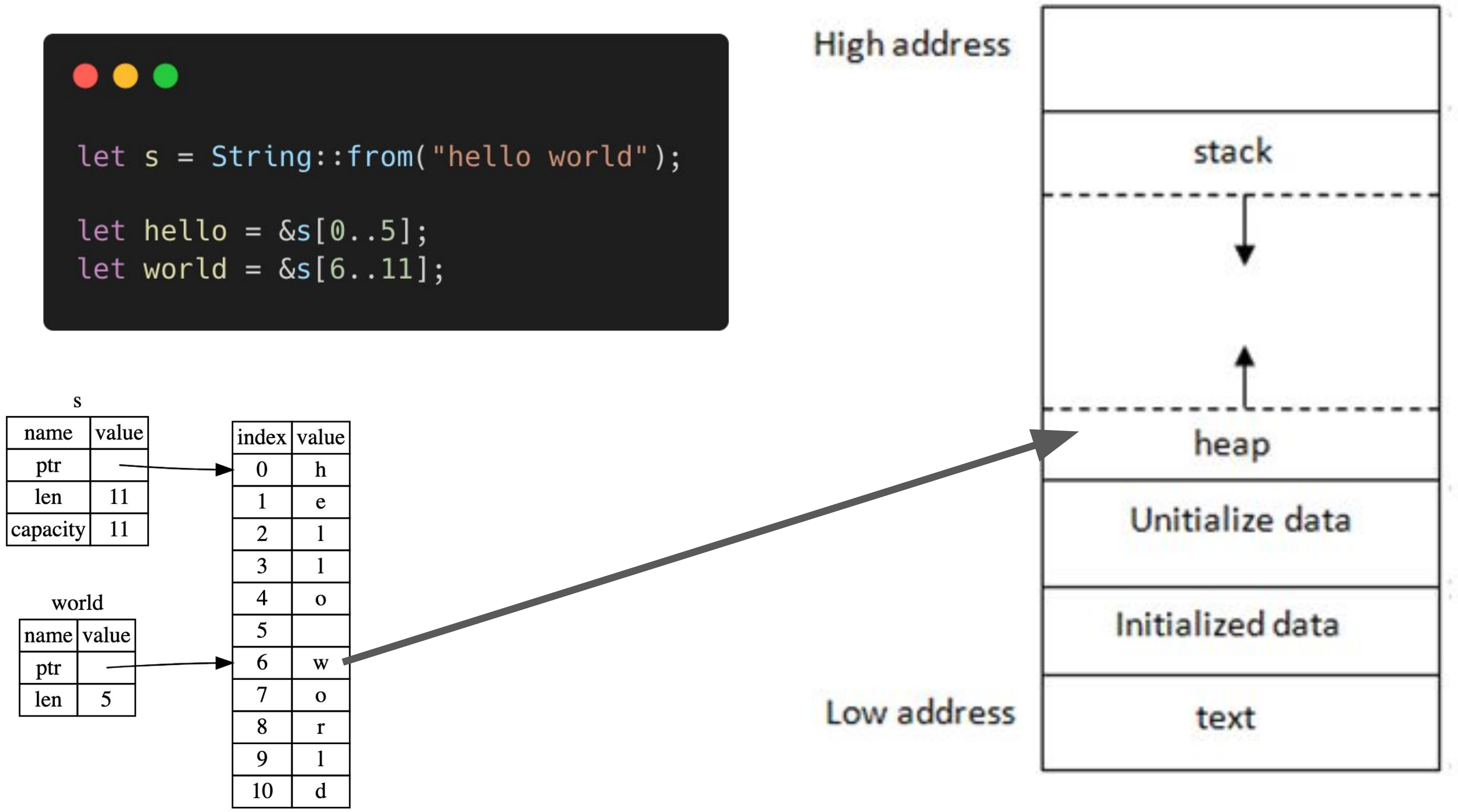
```
let world: &str = "world";
```



String Slices in Memory



```rust
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)
- Vector slices have type **&[T]**
  - The vector has elements of type **T** (any type)
  - A borrow to an array (vectors just have arrays under the hood!)

Vector Slices