Technische Hochschule Ingolstadt

Fakultät Informatik

Basic introduction to Rust

Principles of Modern Software Development WS 22/23

07.12.2022

Basic Introduction to Rust

General Information about Rust

Source: https://upload.wikimedia.org/wikipedia/commons/d/d5/Rust_programming_language_black_logo.svg

- History:
  - Initially introduced in 2010, first stable version in 2015
- Most recent version:
  - 1.6.5.0

Basic Introduction to Rust

General Information about Rust

- General purpose programming language
  - Can be used for multiple uses
- Static typed
  - Type-checking/derivation during compilation (e.g. compiled programming language)
- Memory Management
  - Memory management done mostly by the compiler programmer when not using specific types
- General Concept
  - Ownership: where data belongs to exactly one owner
  - Lifetime: Data has a defined lifetime and can only be accessed as long as the lifetime is valid

Basic Introduction to Rust

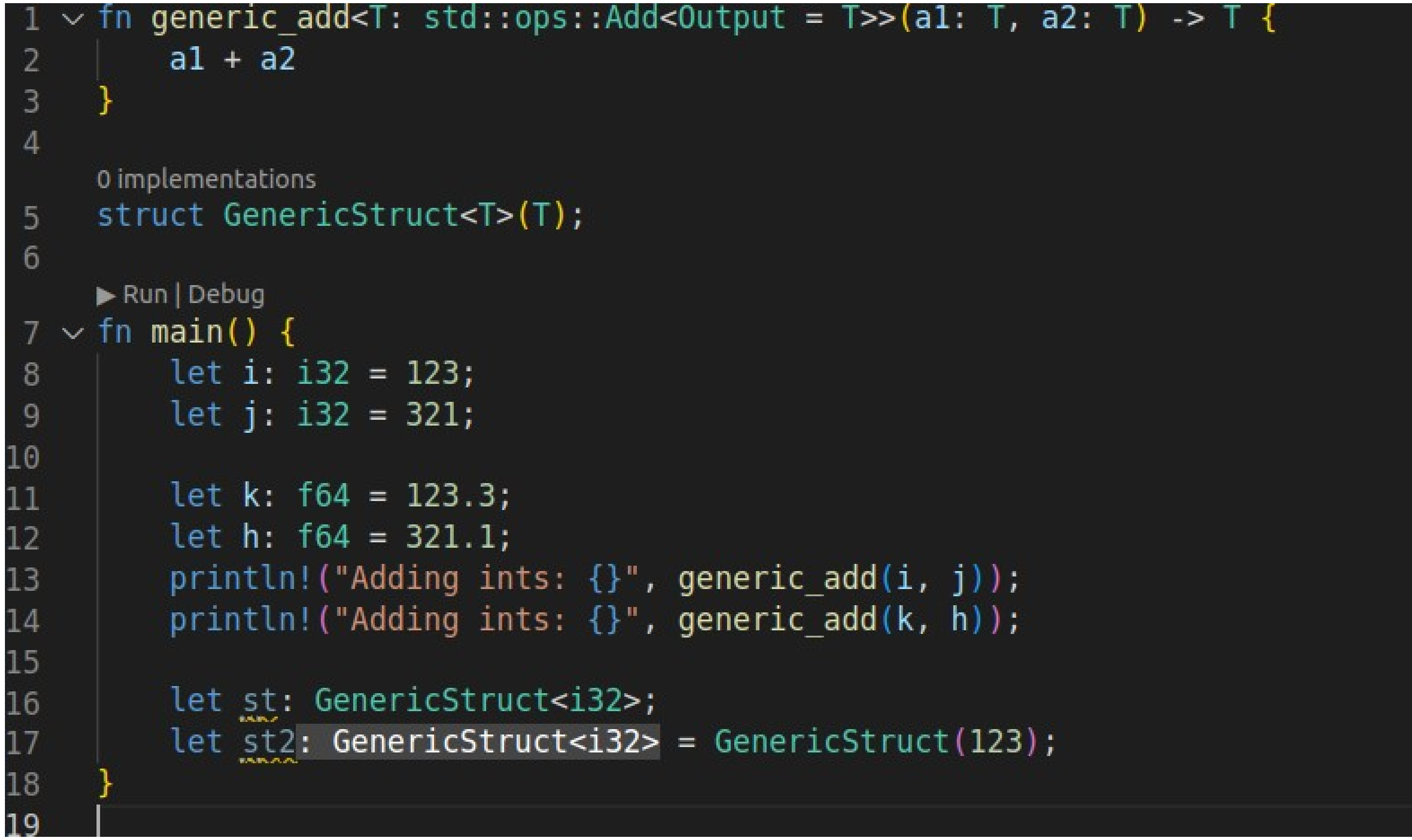
General Information about Rust

- One official compiler
  - rustc
- Almost everything is done via cargo
  - Setup projects
  - Manage dependencies
  - Compile and run
  - Execute tests, etc.

Basic Introduction to Rust

General Information about Rust

- Generic
  - Design of functions/methods that can be used with multiple types that share behavior.
- Example in Rust:
  - Using Traits and generic parameters

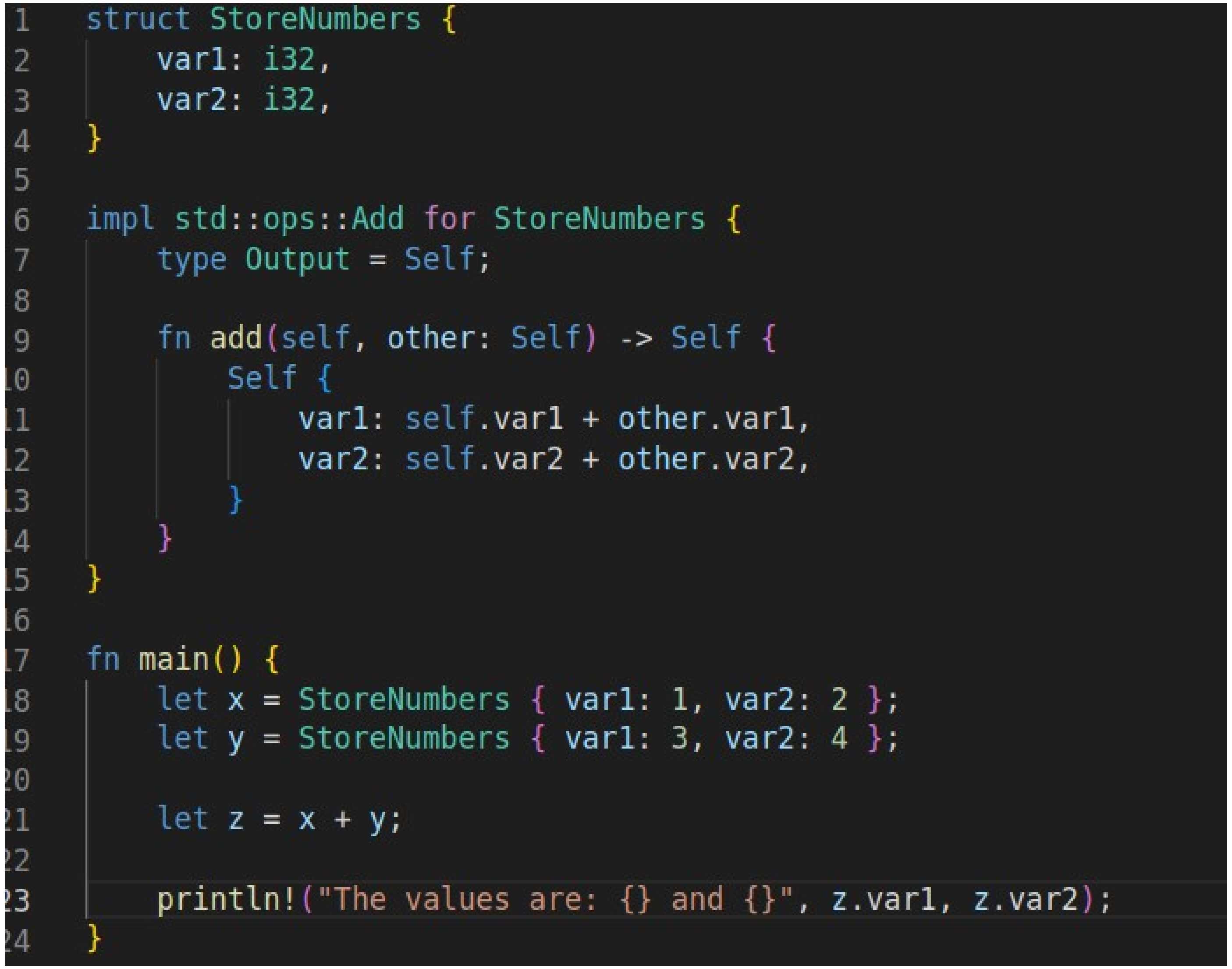


```rust
fn generic_add<T: std::ops::Add<Output = T>>(a1: T, a2: T) -> T {
    a1 + a2
}

0 implementations
struct GenericStruct<T>(T);

▶ Run | Debug
fn main() {
    let i: i32 = 123;
    let j: i32 = 321;

    let k: f64 = 123.3;
    let h: f64 = 321.1;
    println!("Adding ints: {}", generic_add(i, j));
    println!("Adding ints: {}", generic_add(k, h));

    let st: GenericStruct<i32>;
    let st2: GenericStruct<i32> = GenericStruct(123);
}
```

Basic Introduction to Rust

General Information about Rust

- “Object-oriented”
  - Supports structs and and respective methods
  - No overloading of functions
  - No base ↔ child connection allowed
- Definition of “interfaces” with traits

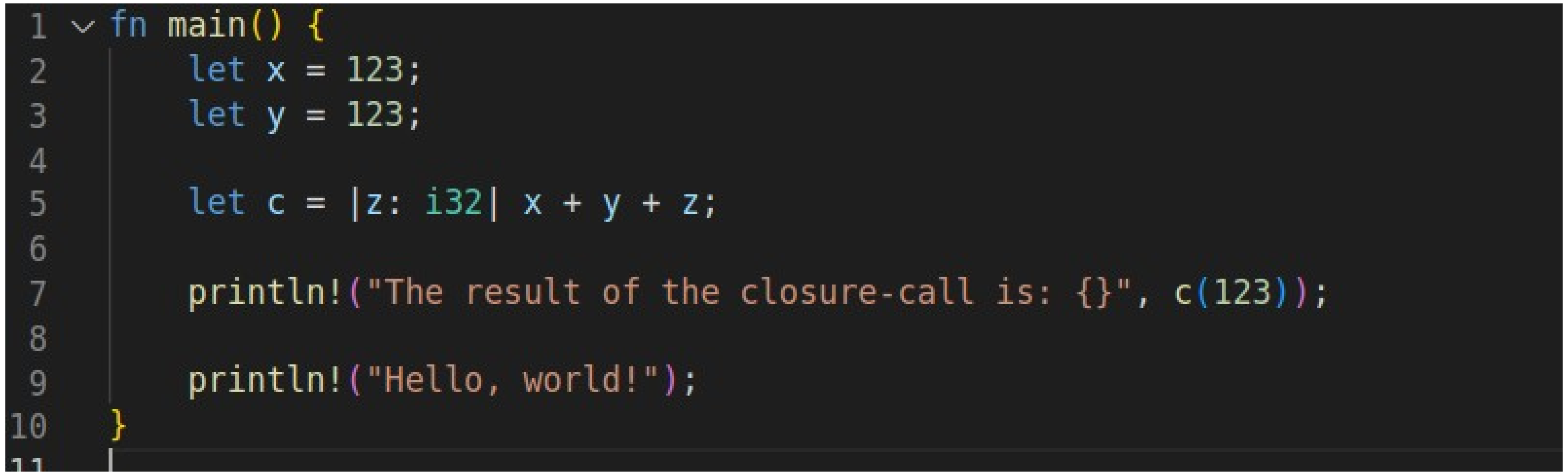


```rust
struct StoreNumbers {
    var1: i32,
    var2: i32,
}

impl std::ops::Add for StoreNumbers {
    type Output = Self;

    fn add(self, other: Self) -> Self {
        Self {
            var1: self.var1 + other.var1,
            var2: self.var2 + other.var2,
        }
    }
}

fn main() {
    let x = StoreNumbers { var1: 1, var2: 2 };
    let y = StoreNumbers { var1: 3, var2: 4 };

    let z = x + y;

    println!("The values are: {} and {}", z.var1, z.var2);
}
```

Basic Introduction to Rust

General Information about Rust

- Functional
  - Strong integration of closures
  - Strong integration of iterators



```
fn main() {
    let x = 123;
    let y = 123;

    let c = |z: i32| x + y + z;

    println!("The result of the closure-call is: {}", c(123));

    println!("Hello, world!");
}
```

Technische Hochschule Ingolstadt

Fakultät Informatik

Development Tools

Principles of Modern Software Development WS 22/23

10.11.2022

Development Setup

- Tools that can be used with rust-analyzer plugin:
  - Visual Studio Code
  - Eclipse IDE
  - Vim
  - etc.

Technische Hochschule Ingolstadt

Fakultät Informatik

Introduction to the Basics of the Rust Programming Language

Principles of Modern Software Development WS 22/23

12.10.2022

Introduction to the C++ Programming Language

General Introduction

- Interactive Rust course:
  - https://rust-book.cs.brown.edu

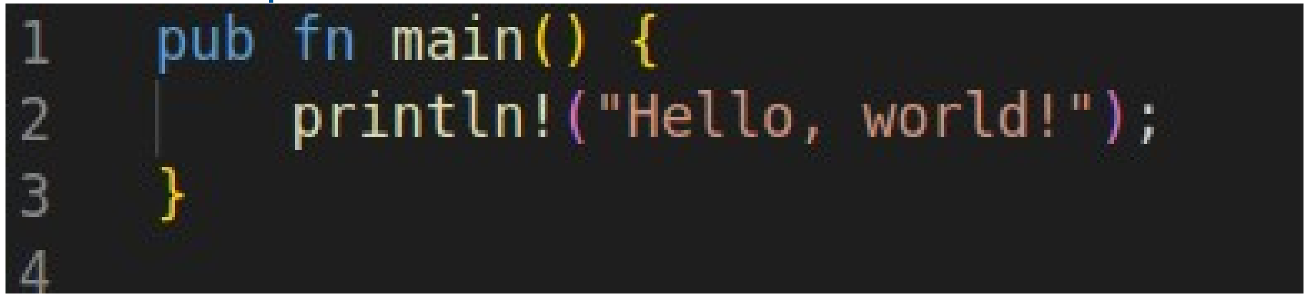
- Online Code-Execution:
  - https://godbolt.org/

COMPILER EXPLORER

Introduction to the C++ Programming Language

General Introduction

- Basic example



```
pub fn main() {
    println!("Hello, world!");
}
```

- Rust uses {} to indicate blocks
- Typical way of execution:
  - Compile source-files to executable
  - Directly run the executable

Technische Hochschule Ingolstadt
Fakultät Informatik

Introduction to the Basics of Rust's Ownership

Principles of Modern Software Development WS 22/23

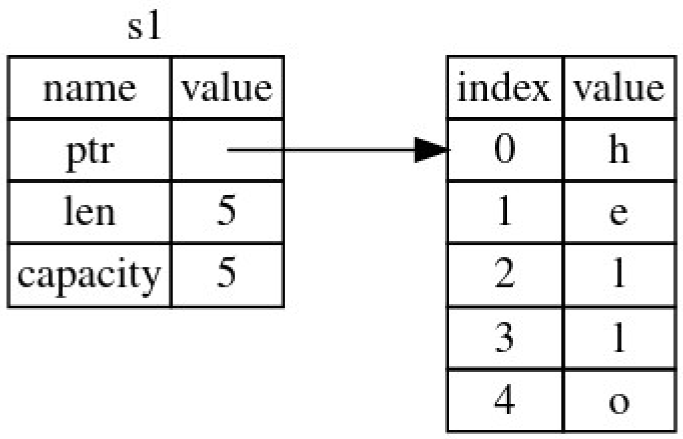
12.10.2022

Introduction to Ownership

General Introduction

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

  let s1 = String::from(“hello”);



name	value
ptr	
len	5
capacity	5

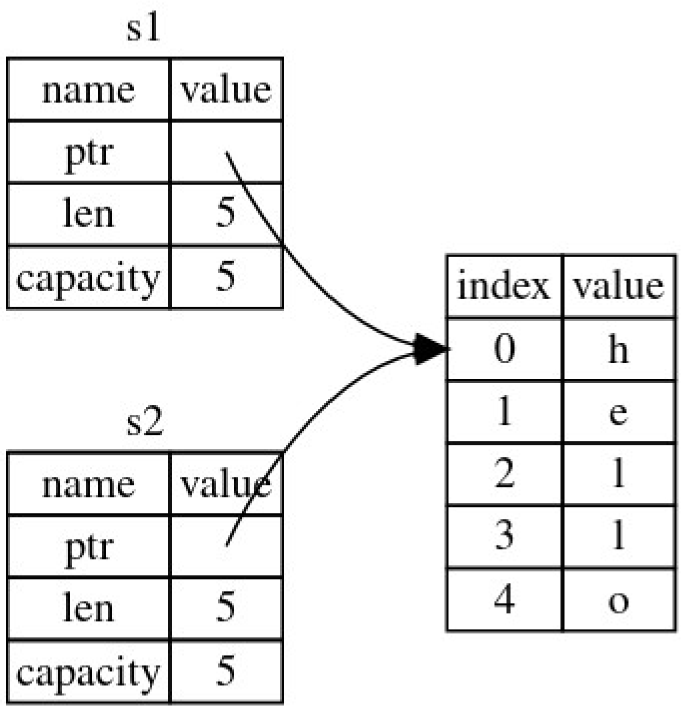
index	value
0	h
1	e
2	l
3	l
4	o

igure 4-1: Representation in memory of a `String` holding the value `"hello"` bound to `s1`

Introduction to Ownership

General Introduction

let s2 = s1;



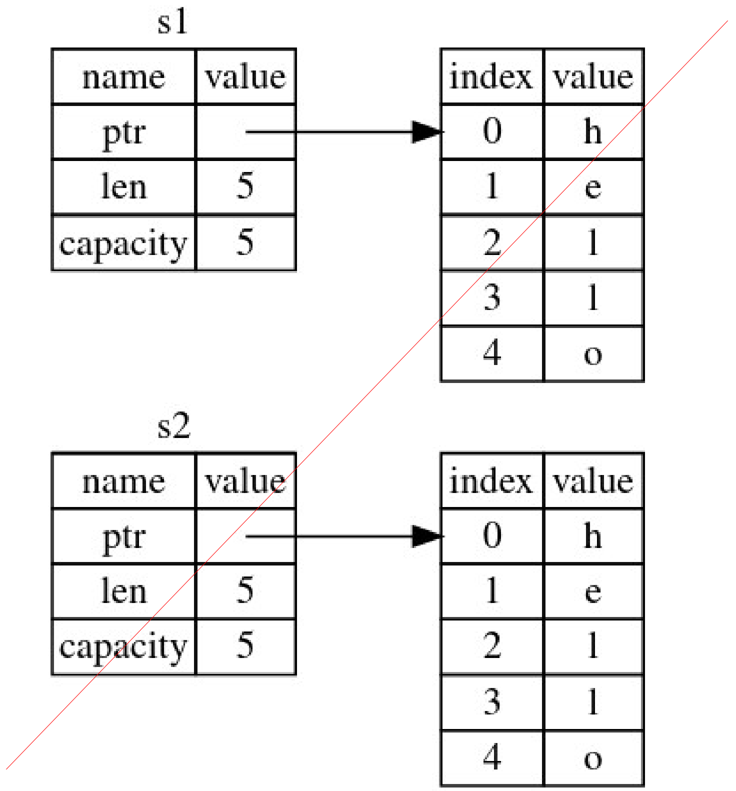


Figure 4-3: Another possibility for what `s2 = s1` might do if Rust copied the heap data as well

igure 4-2: Representation in memory of the variable `s2` that has a copy of the pointer, length, and capacity of `s1`

Introduction to Ownership

General Introduction

When s1 = s2,
s1 is now invalidated,
it can not be used anymore!
-> Move-thing by default!

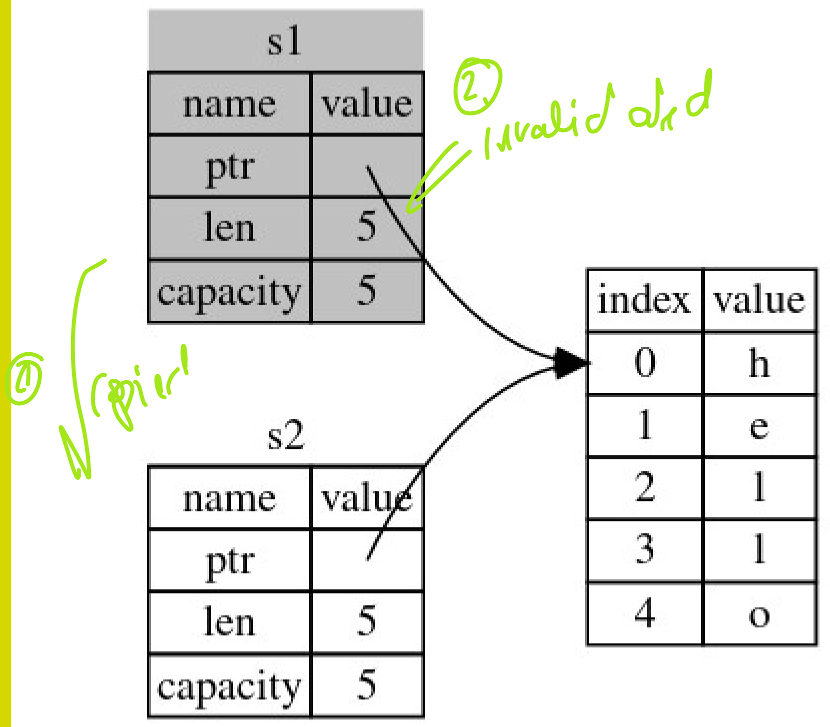


Figure 4-4: Representation in memory after `s1` has been invalidated

-> Basic Types make a copy!
Pointer can not be copied

Introduction to Ownership

General Introduction

This doesn't violate the ownership rule of one owner!

Because we still have only one owner (s1), and only a reference to it!

fn cal_fn(s : &String);

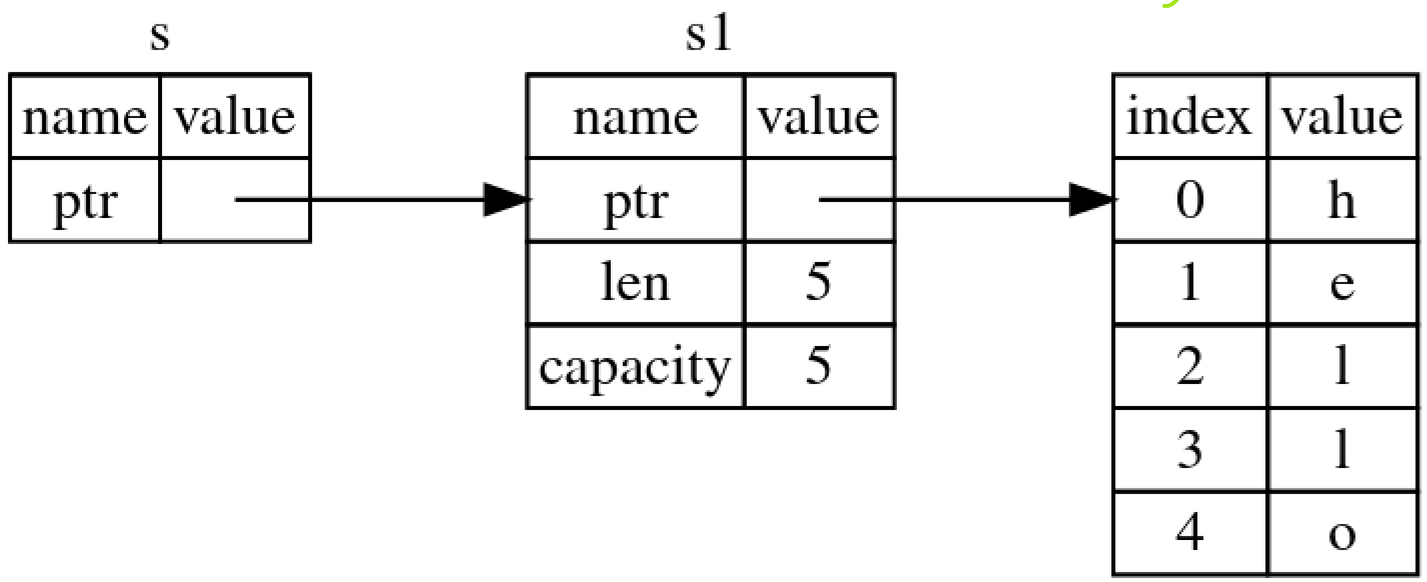


Figure 4-5: A diagram of `&String s` pointing at `String s1`