# Systemnahe Programmierung in Rust

## - “The Book” / Ownership / Kap. 4 -

Hubert Högl

Technische Hochschule Augsburg / Informatik
https://tha.de/~hhoegl

2024-10-05 13:23:09

# Ownership (4.1)

- In Rust wird der Heap-Speicher durch das Konzept der “Eigentümerschaft” verwaltet
- Grundeigenschaften von *Stack* und *Heap*
- Eigentümerregeln
  - Jeder Wert in Rust hat eine Variable, die als sein Eigentümer bezeichnet wird.
  - Es kann immer nur einen Eigentümer zur gleichen Zeit geben.
  - Wenn der Eigentümer den Gültigkeitsbereich verlässt, wird der Wert aufgeräumt.
- Gültigkeitsbereich einer Variablen (scope)

```
{
    let s = "Hallo";
}
```

# Der Typ String

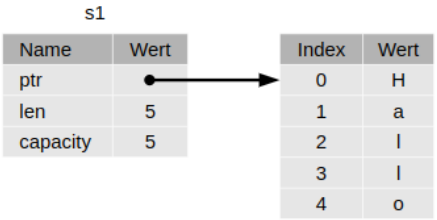


Figure 1: String `s1`. Stack (links) und Heap (rechts)

Was passiert bei `s2 = s1`?

```
let s1 = String::from("Hallo");
let s2 = s1;
println!("{} Welt!", s1);
```

Verschieben (move): flache Kopie und invalidieren von `s1`.

# Clone() und copy()

`clone()` kopiert auch die Daten auf dem Heap:

```
let s1 = String::from("Hallo");
let s2 = s1.clone();
```

`copy()` arbeitet nur auf dem Stack. Es wird bei Daten verwendet, die eine feste Grösse zur Kompilierzeit haben.

```
let x = 5;
let y = x;
```

# Eigentümerschaft und Funktionen

```rust
fn takes_ownership(some_string: String) { ... };

fn makes_copy(some_integer: i32) { ... };

fn takes_and_gives_back(a_string: String) -> String {
    a_string
};
```

# Referenzen und Ausleihen (4.2)

```rust
let s1 = String::from("Hallo");
let len = calculate_length(&s1);

fn calculate_length(s: &String) -> usize {
    s.len()
}
```

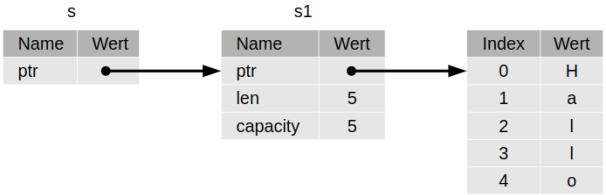


Figure 2: Referenz auf einen String

Gegenteil: *Dereferenzierung* mit * Operator (Kap. 8 und 15).

# Regeln für Referenzen

Referenzen besitzen nicht die Werte, auf die sie verweisen.

Erstellen einer Referenz ist *Ausleihen* (borrowing).

Veränderliche Referenzen mit `&mut`.

Nicht-lexikalische Lebensdauer: Compiler erkennt auch schon *vor* dem Gültigkeitsbereichsende, ob eine Referenz nicht mehr verwendet wird.

Ausleih-Regeln:

- Zu jedem beliebigen Zeitpunkt kannst du entweder eine veränderliche Referenz oder eine beliebige Anzahl unveränderlicher Referenzen haben.
- Referenzen müssen immer gültig sein.

# String Anteilstyp (string slice) (4.3)

Motivation: Wortende in einem String finden.

Zeichenkettenanteilstyp: `&str`

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

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

**Wichtig:** Bereichsindizes bei Zeichenkettenanteilstypen müssen sich nach gültigen UTF-8-Zeichengrenzen richten. Falls nicht, gibt es einen Laufzeitfehler.

```
fn first_word(s: &String) -> &str {
    let bytes = s.as_bytes();
    for (i, &item) in bytes.iter().enumerate() {
        if item == b' ' {
            return &s[0..i];
        }
    }
    &s[..]
}
```

# Anteilstyp (Slice) (4.3)

Literale sind Anteilstypen: `let s = "Hallo Welt!";`

Array Anteilstyp:

```
let a = [1, 2, 3, 4, 5];
let slice = &a[1..3];        // Typ &[i32]
assert_eq!(slice, &[2, 3]);
```