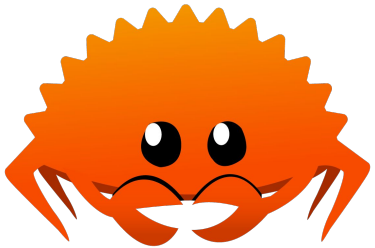
Rust

Informatik Workshop



Inhalt

- Motivation & Historie
- Basics
- Allgemeine Programmierkonzepte
- Fortgeschrittene Programmierkonzepte
- Diskussionsrunde

Makros

Makros

Was sind Makros?

- Verschiedene Arten von Makros
  - Prozedurale Makros
    - Benutzerdefinierte Makros mit #[derive]
    - Attributähnliche Makros
    - Funktionsähnliche Makros
  - Deklarative Makros
- Metaprogrammierung: Wir schreiben Code, der wiederum Code schreibt
  - nützlich, um die Menge an Code zu reduzieren, die geschrieben werden muss
- Bekannte Makros: println! und vec!
- Unterschiede zwischen Makros und Funktionen
  - Makros können eine variable Anzahl Parameter annehmen
  - Makros expandieren bevor, der Compiler interpretiert
  - Makrodefinitionen sind komplexer als Funktionsdefinitionen, da man Code schreibt, der Code schreibt

Makros

Vektoren

- Deklaratives Makro
- #[macro_export] wird benötigt, um das Makro in den Gültigkeitsbereichs des Crates mitzubringen
- Makro zum erstellen eines Vektors mit beliebig vielen Parametern eines beliebigen Typs

```
#[macro_export]
macro_rules! vec {
    ( $( $x:expr ),* ) => {
        {
            let mut temp_vec = Vec::new();
            $(
                temp_vec.push($x);
            )*
            temp_vec
        }
    };
}
```

Vereinfachte Version des Vektor Makros

Makros

Vektoren

Initialisierung eines Vektors bestehend aus Integer Werten

Mit Makro:

```
let v: Vec<u32> = vec![1, 2, 3];
```

Aufruf des Vektor Makros

statt...

```
let mut vec = Vec::new();
vec.push(1);
vec.push(2);
vec.push(3);
```

Erstellung des Vektors ohne Makro

Makros

Vektoren

Ersetzung von Code durch das Vektor Makro

```
let v: Vec<u32> = vec![1, 2, 3];
```

Aufruf des Vektor Makros

```
let mut temp_vec = Vec::new();
temp_vec.push(1);
temp_vec.push(2);
temp_vec.push(3);
```

Aussehen des Codes nach der Makro Expandierung

Makros

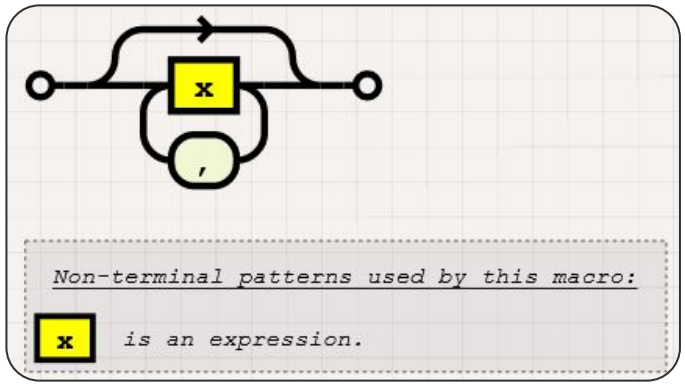
Vektoren

Aufbau des Vektor Makros

Wenn das Muster übereinstimmt (match), wird der zugehörige Codeblock ausgegeben

```
#[macro_export]
macro_rules! vec {
    ( $( $x:expr ),* ) => {
        {
            let mut temp_vec = Vec::new();
            $(
                temp_vec.push($x);
            )*
            temp_vec
        }
    };
}
```

Vereinfachte Version des Vektor Makros



Syntax Diagramm zum Vektor Makro

Makros

Beispiele für komplexere Makros

Das retry! Makro

Umhüllt eine gegebene Funktion mit einer retry Logik (die Anzahl der Wiederholungen kann optional angegeben werden)

```
fn main() {
    let mut eventually_succeed = succeed_after!(2);
    let res = retry!(eventually_succeed);
}
```

Beispielanwendung des retry! Makros

Use cases:

- der Versuch, ungültige Dateipfade zu öffnen
- Unlogische Werte parsen
- Daten von einem Server anfragen/erhalten

Makros

Beispiele für komplexere Makros

Das timeit! Makro

Kann eine Funktion umhüllen, um eine Meldung auszugeben, wie lange die Ausführung gedauert hat

Use cases:

- Benchmarking
- Testing

```rust
fn wait_for_it() -> String {
    std::thread::sleep(std::time::Duration::from_secs(2));
    return String::from("warten.");
}

fn main() {
    eprintln!("Auf die Ausführung von...");
    let res = timeit!(wait_for_it());
    eprintln!("{}", res);
}
```

Beispielanwendung des timeit! Makros

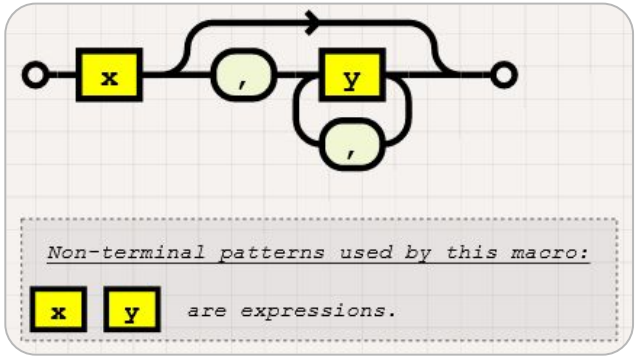
```
Auf die Ausführung von...
'wait_for_it' took 2012 ms
warten.
```

Ausgabe der Beispielanwendung

Makros

Erstellung eines eigenen Makros

Das find_min! Makro



Syntax Diagramm zum find_min! Makro

```
macro_rules! find_max {
    ($x:expr) => ($x);
    ($x:expr, $($y:expr),+) => (
        std::cmp::max($x, find_max!($($y),+))
    )
}
```

Erstellen eines eigenen Makros

Übung

Die Übungen befinden sich im Ordner “Uebung_Makros”