

Betriebssysteme 2

[Dashboard](#) ► [Meine Kurse](#) ► [Betriebssysteme 2](#) ► [Abschnitte](#) ► [Exercises](#) ► [Aufgabenblatt 2 \[4P\]](#)



Aufgabenblatt 2 [4P]

1 Mini-Quiz (0P)

Wir haben für Sie ein [Quiz](#) vorbereitet, das Sie vor der Bearbeitung der Programmieraufgaben lösen sollten. Ärgern Sie sich nicht, wenn Sie nicht alle Fragen richtig beantworten: die Codebeispiele sind bewusst Grenzfälle, die Ihr Verständnis testen sollen.

2 Stackbasierter Vektor (2P)

Laden Sie das vorbereitete Archiv herunter und importieren Sie das Cargo-Projekt `stack_vec`.

Implementieren Sie die Schnittstelle eines stackbasierten Kollektionstyps, mit dem sich ein statisches Array wie ein Vektor benutzen lässt. Die Datenstruktur geben wir Ihnen vor:

```
struct StackVec<'a, T:'a> {
»   storage: &'a mut [T],
»   len: usize
}
```

Hier ist ein Beispiel für die Benutzung der Klasse:

```
1 let mut storage = [0u8; 1024];
2 let mut vec = StackVec::new(&mut storage);
3
4 for i in 0..10 {
5     vec.push(i * i).unwrap();
6 }
7
8 for e in vec.as_slice() {
9     println!("{}", *e);
10 }
11
12 let last_element = vec.pop().unwrap();
13 assert_eq!(last_element, 9 * 9);
```

Die auszufüllende Schnittstelle benutzt die Strukturen `Result<T,E>` und `Option<T>` aus der Standardbibliothek, die im Vorlesungsteil erwähnt wurden. Beide Strukturen stehen in jedem Rust-Programm zur Verfügung und sind nachfolgend noch einmal wiedergegeben. Sogar die Enumerationsvarianten `Ok/Some` und `Err/None` lassen sich unqualifiziert verwenden. `Result` und `Option` implementieren unter anderem die Methode `unwrap()` (Zeilen 5 und 12 im Beispiel), die für `Ok` bzw. `Some` den Wert zurückgibt und die Ausführung für `Err` bzw. `None` mit einer Fehlermeldung abbricht.

```
1 enum Result<T,E> {
2     » Ok(T),
3     » Err(E)
4 }
5
6 enum Option<T> {
7     » Some(T),
8     » None
9 }
```


3 Endlicher Automat (2P)

Importieren Sie das Cargo-Projekt `state_machine`.

Das [Typestate-Pattern](#) ist ein API-Entwurfsmuster, das Informationen über den Laufzeitzustand eines Objektes in dessen Datentyp kodiert. Die Schnittstelle lässt sich dadurch zur Übersetzungszeit so einschränken, dass in bestimmten Zuständen undefinierte Operationen zu Übersetzungs- statt zu Laufzeitfehlern führen. Damit lässt sich also aus Sicht des API-Entwicklers eine bestimmte Reihenfolge in der Abarbeitung von Operationen erzwingen, was sowohl für den Nutzer hilfreich ist, als auch Laufzeitkosten spart.

Implementieren Sie einen endlichen Automaten mit den Zuständen `Start`, `Loop` und `End`, wobei nur Zustand `Start` erzeugt werden kann, Transitionen zwischen `Start` und `Loop`, `Loop` und `End` sowie `Loop` und `Loop` möglich sind und nur `End` ein Endzustand ist. Wir geben Ihnen dafür die folgenden Traits vor, deren Trait-Bounds Sie so ergänzen sollen, dass sowohl Nutzer, die diese Traits implementieren, als auch Nutzer, die die implementierenden Strukturen benutzen, keine Fehler in der Benutzung machen können, die nicht schon zur Übersetzungszeit diagnostiziert werden.

```
1 trait State {}
2 trait Terminal {}
3
4 trait Transition<T> {
5     » fn transition(self) -> T;
6 }
7
8 trait Terminate {
9     » fn terminate(self);
10 }
```

 [exercise2.zip](#)

7. Oktober 2020, 13:52

Abgabestatus

Nummer der Einreichung	Dies ist Einreichung 1
Abgabestatus	Keine Einreichung
Bewertungsstatus	Nicht bewertet
Fälligkeitsdatum	Do, 19. November 2020, 00:00
Verbleibende Zeit	1 Tag 9 Stunden
Zuletzt geändert	-

Abgabe hinzufügen

Sie haben bisher keine Lösungen abgegeben.

◀ Quiz zu Aufgabenblatt 1

Wechseln zu ...

Quiz zu Aufgabenblatt 2 ▶