– Praktikumsaufgabe 2 –

Thema: *Einfache Programmierung in Rust*

Zielstellung: Kennenlernen von Anweisungen und Ausdrücken, Automatische Fehlerbehebung mittels `cargo`, Vergleich der Größe ausführbarer Dateien in C und Rust

1. Korrigieren Sie den Fehler im folgenden Programm *auf zwei verschiedene Arten* durch jeweils eine kleine Modifikation. `assert_eq!` ist ein Makro, das die Identität der beiden übergebenen Parameter prüft und bei Ungleichheit die Abarbeitung abbricht.

```
fn main() {
    let v = {
        let mut x = 1;
        x += 2
    };
    assert_eq!(v, 3);
    println!("Success!");
}
```

Listing 1: Programm mit syntaktischem Fehler

2. Im Programm in Listing 2 „verstecken“ sich 8 Fehler. Versuchen Sie zunächst, diese zu identifizieren. Nutzen Sie danach die (nicht in der Vorlesung behandelte) Funktion `fix` von `cargo`, um so viele Fehler wie möglich automatisiert zu beheben.

   Hinweis: Das nötige Kommando lautet

   ```
   $ cargo fix --bin "<projektname>" --allow-dirty
   ```

3. Schreiben Sie ein Rust-Programm, das mit Hilfe der Reihenentwicklung

$$e \approx \sum_{k=0}^{\infty} \frac{1}{k!}$$

   die Eulersche Zahl e approximiert. Bestimmen Sie (programmtechnisch) die Anzahl korrekt ermittelter Nachkommastellen unter der Voraussetzung, dass

$$e \approx 2.71828182845904523536028747135266249775724709369995957496696 76\ldots$$

   gilt.

4.* Vergleichen Sie die Größe des „Hello, world“-Binaries

   a) als Debug-Target

   b) als Release-Target nach Anwendung von `strip`

   c) als C-Programm.

   Gibt es Methoden, die Größe eines solchen Binaries weiter zu reduzieren?

```rust
use std::io;

fn foo () -> f64
{
    return (2.7)
}

fn main() {
    let x = 3;
    let mut y:f64 = 3.1415;
    let BAD_NAME = y *3.0;

    while (16.0 > 15.0) {
        println!("Pi is approximately {}.", y);
    }
}
```

Listing 2: Ein Programm mit automatisch behebbaren Warnungen (cargo-fix-exmpl)