– Lösung zur Praktikumsaufgabe 2 –

Thema: *Einfache Programmierung in Rust*

1. Am Ende des lokalen Blockes muss ein Ausdruck stehen, der den Wert des Blockes repräsentiert (der `v` zugewiesen wird). `x+=2` ist aber eine *Anweisung*. Man könnte also erstens `x` als letzte Zeile einfügen. Alternativ können Sie auch den lokalen Block unverändert lassen – dann würde dieser nichts zurückliefern. Dann müssen Sie aber das Makro zu `assert_eq!(v, ())` anpassen; weil `v` durch die Zuweisung genau diesen Wert erhält.

2. Der Code enthält insgesamt 8 Fehler, die in der Tabelle aufgeführt sind (in Zeile 11 verstecken sich gleich 2).

Zeile	*Fehler*
1	ungenutzter Import
3	Definition einer unbenutzten Funktion („Dead Code“)
5	unnötige Klammerung um `return`
9	Definition einer unbenutzten Variable (`x`)
10	`mut`-Schlüsselwort ist unnötig, da Variable nicht verändert wird
11	Variablenname entspricht nicht der Konvention (Kleinbuchstaben); außerdem wird diese Variable nicht benutzt
13	unnötige Klammerung um `while`-Bedingung

   Alle Fehler resultieren in Warnungen; d.h., der Build-Vorgang läuft trotzdem bis zum Ende.

```
warning: `cargo-fix-exmpl` (bin "cargo-fix-exmpl") generated 8 warnings
(run `cargo fix --bin "cargo-fix-exmpl"` to apply 6 suggestions)
    Finished dev [unoptimized + debuginfo] target(s) in 0.21s
```

   Die Funktion `fix` von `cargo` kann 6 der 8 Warnungen beheben:

```
$ cargo fix --bin "cargo-fix-exmpl" --allow-dirty
    Checking cargo-fix-exmpl v0.1.0 (/home/robge/txt/job/htw/rust/src
/cargo-fix-exmpl)
       Fixed src/main.rs (6 fixes)
warning: function `foo` is never used
 --> src/main.rs:3:4
  |
3 | fn foo () -> f64
  |    ^^^
```

```
   |
   = note: `#[warn(dead_code)]` on by default

warning: variable `_BAD_NAME` should have a snake case name
  --> src/main.rs:11:9
   |
11 |     let _BAD_NAME = y * 3.0;
   |         ^^^^^^^^^ help: convert the identifier to snake case: `_bad_name`
   |
   = note: `#[warn(non_snake_case)]` on by default

warning: `cargo-fix-exmpl` (bin "cargo-fix-exmpl") generated 2 warnings
    Finished dev [unoptimized + debuginfo] target(s) in 0.15s
```

`cargo` getraut sich nicht, die ungenutzte Funktion `foo()` komplett zu eliminieren und den Variablennamen `_BAD_NAME` in Kleinbuchstaben zu konvertieren. Ungenutzte Variablen werden auch nicht eliminiert, sondern erhalten einen Unterstrich vor dem Namen.

3.
```rust
fn main() {
    let e_ref = 2.71828182845904523536028747135266249775724709369995957496696762;
    let mut k = 2.0_f64;   // index
    let mut e: f64 = 2.0;  // initial approximation
    let mut inc: f64 = 1.0;

    while inc > 0.0 {
        inc /= k;   //
        e += inc;
        println!("{} e={} delta={}", k, e, e - e_ref);
        k += 1.0;
    }
    let mut f = e - e_ref;
    let mut c = 0;
    while f < 1.0 {
        f = f*10.0;
        c += 1;
    }
    println!("e is correct up to the {}th decimal place. Not bad.", c-1)

}
```

Listing 1: Approximation von e und Ermittlung der erzielten Genauigkeit (`euler`)

4.* Der Unterschied zwischen C und Rust bezüglich der Größe der erzeugten Binärimages ist gewaltig, wie die folgende Tabelle demonstriert.

Binary	*Größe* [Bytes]
Rust; Debug-Target	3.792.552
Rust; Release-Target, stripped	358.808
C; stripped	14.416

Um das ausführbare Programm weiter zu verkleinern, kann man es beispielsweise in Assembler implementieren. Listing 2 zeigt eine Möglichkeit. Das resultierende Binärimage ist 4456 Byte groß; der Gewinn ist also überschaubar.

```
# hello64.s
# "Hello, world!" as x86_64 assembler program
# gcc -no-pie -nostdlib -o hello64 hello64.s

        .text
        .global _start

_start:
        movq $1,%rax
        movq $1,%rdi
        movq $msg,%rsi
        movq $0xe,%rdx
        syscall
        movq $60,%rax
        movq $42,%rdi
        syscall
msg:
        .ascii "Hello, world!\n\00"
```

Listing 2: „Hello, world!“ als 64-Bit-Assemblerprogramm für x86_64-Prozessoren und Linux