Universität Bielefeld
Technische Fakultät

AG Genominformatik
Dr. Lucas Robidou

The Rust programming language
Summer 2024 / 2025

Exercises

Number 2, Discussion: 2025 April 23 Some exercises are extracted from rustling. See the first "exercise" sheet on how to install rustling for a more interactive experience.

1 Control flow

1. From rustling: fix this code so that it compiles and prints correctly.

```
fn bigger(a: i32, b: i32) -> i32 {
    // TODO: Complete this function to return the bigger number!
    // If both numbers are equal, any of them can be returned.
    // Do not use:
    // - another function call
    // - additional variables
}
```

2. Write an enum whose variants are your favorite dishes (or animals, or whatever) (at least 3 variants). Write a function that takes your enum as a parameter[1]. Inside the function, using a match, print a small sentence about each possible variant.

2 Functions

1. Write a function that takes a u32 and returns its square. Do not use a return statement.
2. Write a function that takes a String and prints it. What are the consequences for the people using your function? How can they call your function multiple times with the same variable?
3. Same as above, but take a reference to the String. What are the consequences for the caller of your function?
4. (advanced) Explain in your own words a potential issue of taking a reference to a String (i.e. &String). Fix it.

3 Structs

3.1 The `Point` structure

1. Write a struct called `Point` that has two members: x and y, both float (64 bits wide)[2]. This represents a point in a 2d space.
2. Derive `Debug` and `Clone` for `Point`s.

[1] Just like structs, enums can be used as parameters of functions, e.g. `fn fonc(x: &YourEnum) {}`
[2] the type is `f64`

3. Place that struct in its own file. Make the struct public, but let the members be private. Do not forget to declare the module in your main.rs and to import the struct (e.g. `use point::Point`).
4. Ho no! People can't create new `Point`s now. Implement a `new` method for `Point`s to fix this. This method should take two integers and return a `Point`.
5. Did you forget to make the method public? If so, fix it.
6. Write a public method, called `distance`, that computes the distance between two points in a 2d space.[3]

```
impl Point {
    // the new method

    fn distance(&self, other: &Point) -> f64{
        //
    }
}
```

`f64` implements a sqrt() method : to get the square root of a f64, just use `number.sqrt()`.

3.2 The `Rectangle` structure

1. In your main file, write a new struct called `Rectangle`, that consists of two `Point`s, its `top_left` point and its `bottom_right` point. Assume the rectangle's sides are always horizontal or vertical.
2. Implement a `new` method that takes two Points. Do not take references here.[4]
3. Implement a method that returns the area of a rectangle.
4. Implement a method that returns whether or not a rectangle contains a Point (the return type should be `bool`).

3.3 The `Circle` structure

1. In your main file, write a new struct called `Cirle`, that consists of a `Point` and a radius.
2. Implement a method that returns the area of a circle.
3. Implement a method that returns whether or not a circle contains a Point.

[3] If you are unfamiliar with math, return a random integer. Do not let this block you.
[4] You might be tempted to store references to Point in your Rectangle. We will see how to do that later.