Lehrstuhl für Softwaretechnik und Programmiersprachen
Bachelor-Seminar: Programmiersprachen
John Witulski

Heinrich Heine Universität Düsseldorf

Übungsaufgabe zu Rust - Der Schlüssel zur System- und Code-Integrität?

Aufgabe 1.1

Um die Rust-Toolchain auf Ubuntu zu installieren, geben Sie auf der Konsole den folgenden Befehl ein:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs \
| sh -s -- --help
```

Nach dem Befehl haben Sie die Rust-Toolchain installiert. Prüfen Sie Ihre Installation mit:

```
rustc --version
```

Außerdem sollte folgender Befehl funktionieren:

```
cargo --version
```

Geben Sie bitte Bescheid, falls es Probleme bei der Installation gibt.

Jetzt können Sie mit der Bearbeitung der Aufgaben anfangen: Klicken Sie auf diesen Link oder geben Sie ihn ein: `https://github.com/Moonpepperoni/rust-uebung`. Klonen Sie dieses Repository auf Ihren Rechner. Navigieren Sie anschließend in den geklonten Ordner und öffnen ihn in einem Editor Ihrer Wahl. Geben Sie den folgenden Befehl ein:

```
cargo run --bin greet "Ihr␣Name"
```

Sie sollten von den Aufgaben begrüßt worden sein.

Aufgabe 1.2

Öffnen Sie in dem Ordner src/bin die Datei „ aufgabe1.rs“.
Ich bin ein großer Fan von „The Office “ und ich mache Sie zu einem neuen Arbeitskollegen dieser Spaß-Truppe. Unter dem TODO sollen Sie sich zunächst zu den Mitarbeitern hinzufügen. Leider haben sich im Quellcode mehrere gravierende Fehler eingeschlichen. Versuchen Sie (zunächst ohne den Compiler) alle Fehler zu beheben. Führen Sie dann das Programm aus. Nutzen Sie dazu den Befehl:

```
cargo run --bin aufgabe1
```

Im Idealfall haben Sie alle Fehler richtig behoben und sehen auf der Standardausgabe eine kleine Vorstellungsrunde.

Alternativ gibt Ihnen der Compiler nun Hinweise, an welchen Stellen noch Fehler vorliegen. Nutzen Sie die Hilfe vom Compiler, um auch die restlichen Fehler zu beheben. Lesen Sie dazu die Fehlerbeschreibung aufmerksam durch und wählen Sie einen der Fehler aus, der für Sie zunächst behebbar aussieht. Manche Fehler sind nur behebbar, wenn

die zugrunde liegende Ursache behoben wurde. Eventuell verschwinden gleich mehrere Fehlermeldungen durch eine Änderung.

Aufgabe 1.3
Öffnen Sie die Datei „aufgabe2.rs“

In dieser Aufgabe wollen wir Markdown-Dateien rendern. Einen Renderer von Hand zu schreiben ist enorm aufwendig. Deshalb wollen wir uns zunutze machen, dass es bereits andere Markup-Sprachen gibt, für die es bereits Renderer gibt. HTML ist eine solche Markup-Sprache, die von allen Browsern unterstützt wird. Schreiben Sie ein Programm, das als Konsolenargument den Pfad zu einer Markdown-Datei bekommt. Diese sollen Sie vollständig in einen String einlesen. In Zukunft soll unser Programm mehr als nur HTML als Zielsprache unterstützen, deshalb müssen Sie zunächst die Markdown-Elemente in geeigneter Weise in Datenstrukturen aus Rust einlesen. Einen minimalen Startpunkt finden Sie bereits in der vorgegebenen Datei. Vervollständigen Sie dazu die Funktion parse(). Anschließend soll ihr Programm die einzelnen eingelesenen Markdown-Elemente als HTML in die geöffnete Datei schreiben. Vervollständigen dazu die Funktion write_as_html(). Außerdem müssen Sie in der Main-Funktion an den Stellen, wo sich „???“ befinden, den passenden Code einfügen. Zusammengefasst soll das Programm eine gegebene Markdown-Datei zu einer HTML-Datei übersetzen.

Folgende Teile von Markdown sollen Sie zu HTML übersetzen:

```
# Ueberschrift 1
## Ueberschrift 2
### Ueberschrift 3

Normaler Text ohne besondere Syntax

*Kursiver Text*

Gemischter *kursiver* Text

- Item 1
- Item 2
- Item 3

## Auch in einer Ueberschrift kann *kursiver* Text vorkommen
```

Um Ihr Programm zu testen, können Sie folgenden Befehl benutzen:

```
cargo run --bin aufgabe2 assets/simple.md
```

Im Ordner „assets“ finden Sie einige Hilfsdateien. Da können Sie sehen, wie die Übersetzung aussehen soll.

Wenn Sie eine zusätzliche Herausforderung haben wollen, dann verzichten Sie auf das

Kopieren von Teil-Strings. Verwenden Sie nirgendwo im eigenen Code die String struct und arbeiten Sie ausschließlich mit String-Slices &str.

Wenn Sie es sich leichter machen wollen, lassen Sie am besten den kursiven Text aus. Damit können Sie sich einen komplizierten Zwischenschritt sparen.

Code-Schnipsel, die Ihnen helfen sollen: Kommandozeilen-Argumente einlesen:

```
env::args()
```

Liefert einen Iterator auf die Kommandozeilen-Argumente. Achtung: Das 0. Element ist der Name des ausgeführten Programms.

In eine Datei schreiben:

```
writeln!(outfile, "some message {}", argument for braces);
write!(outfile, "some message {}", arugment for braces);
```

Beachten Sie außerdem dass man in Rust Strings nicht direkt indizieren kann:

```
let s = String::from("Hello")
// Das hier geht nicht
let c = &s[0];

// Stattdessen haben wir zwei Alternativen

// 1. mit dem UTF-8 Iterator chars()
let c : char = s.chars().nth(0).unwrap();

// 2. mit dem byte-Slice
let bytes : &[u8] = s.as_bytes();
let c : char = bytes[0]; // bytes.get(0).unwrap();
```

Für hilfreiche Methoden auf Vec, String und Iterator besuchen Sie die folgenden Links:

1. Vec : `https://doc.rust-lang.org/std/vec/struct.Vec.html`
2. String : `https://doc.rust-lang.org/std/string/struct.String.html`
3. Iterator : `https://doc.rust-lang.org/std/iter/trait.Iterator.html`

Viel Erfolg bei der Bearbeitung!