Inleiding

Een nieuwe programmeertaal, zoals Rust, leren vereist oefening! Een nieuwe syntax, nieuwe features andere standaard functies en bibliotheken. Om jullie een quick-start te geven in het gebruik van Rust, maken we dankbaar gebruik van rustlings. Deze set opgaven introduceren Rust op een manier die goed werkt voor programmeurs.

Opdrachten week 7 – Inleiding Rust

Je gaat deze week leren hoe je:

- een Rust IDE opzet;
- de vele aspecten van Rust moet gebruiken.

7.1 We beginnen met het installeren van alle benodigde tools. Als IDE gebruiken we *Visual Studio Code*. De toolchain-beheerder is *Rustup*. Rustup zelf maakt gebruik van diverse build-tools dus we zullen ook een subset van *Visual Studio* moeten installeren. Als je nog niet over git beschikt zul je ook dat moeten installeren. De installatiehandleiding voor Windows-gebruikers is hier te vinden. Mocht je Linux gebruiken dan kun je natuurlijk je eigen package manager gebruiken en dezelfde tools (exclusief Visual Studio BuildTools) installeren. Via WSL2 is ook een optie, dan kun je gewoon in VSCode een remote WSL sessie starten. Om USB-communicatie mogelijk te maken met WSL kun je usbipd-win gebruiken.

7.2 Nu alles is geïnstalleerd is het tijd om aan de slag te gaan!

A Open Visual Studio Code

B Start een terminal met CTRL + ‘. In de nieuwe terminal clone je de RTS10 rustlings repository met:

```
git clone https://github.com/rust-lang/rustlings.git
```

Je kunt natuurlijk ook eerst de repository forken naar je eigen account zodat je gemaakt werk kunt opslaan met behulp van git.

C Druk nu op *Open Folder...* en open de zojuist geclonede directory. Misschien verschijnt een venster met *Do you trust the authors of the files in this folder?*. Je kunt dan voor *Yes, I trust the authors* kiezen.

Het build-systeem van Rust heet Cargo. Er bestaat ook een commando met dezelfde naam. De meest gebruikte acties zijn `new <naam>`, `build` en `run`. De instellingen voor het build process staan in het bestand `Cargo.toml`. Details kun je vinden in het Cargo boek: https://doc.rust-lang.org/cargo/index.html. Om alle rustlings opgaven te kunnen maken zullen deze Cargo-commando's voldoende zijn. Volgende week gaan we aan de slag met de embedded versie van Rust en dan komen we Cargo iets meer tegen.

7.3 In de `rts10_rustlings` folder kun je opnieuw een terminal openen met CTRL + `. Installeer en/of selecteer eerst de stable toolchain met `rustup install stable` en/of `rustup default stable`. Voer dan het commando `cargo build` uit. Wacht totdat alle dependencies en rustlings zelf is gecompileerd. Hierna kun je rustlings uitvoeren met het commando `cargo run watch`. Het laatste woord `watch` is een parameter wat wordt doorgegeven aan rustlings en hoort niet bij het cargo build-systeem.

7.4 Rustlings zelf is eenvoudig. De eerste keer zie je figuur 1. Je kunt klikken op het bestand `intro1.rs` terwijl je CTRL ingedrukt houdt. Dit opent het bestandje in vscode. Rustlings zal pas verder gaan naar de volgende opdracht wanneer een opgave is opgelost en `//I AM NOT DONE` wordt weggehaald uit het bestand.

Figuur 1: Eerste keer uitvoeren van rustlings.

7.5 Tijdens de rest van deze les is het doel deze opgaven uit te werken. Dit wordt niet getoets, het is puur voor jezelf om snel gewend te raken met rust.

7.6 Als laatste kan het leerzaam zijn nog een integrale opdracht te doen. Het berekenen van de score van een bowlingpartij blijkt lastig te zijn! Elke partij bestaat uit 10 beurten (frames). In elke beurt krijgt de speler twee worpen (rolls). Er zijn nu per beurt drie mogelijkheden:

- Strike! Eerste keer alle 10 de pinnen om. De beurt is voorbij. Een uitzondering in beurt 10; als er dan een strike is gegooid krijgt de speler nog 2 extra worpen.
- Spare! In twee gooien alle pinnen om. De beurt is voorbij. Een uitzondering in beurt 10; als er dan een spare is gegooid krijgt de speler nog 1 extra worp.
- Open. Er blijven na twee maal gooien nog pinnen over. De beurt is voorbij.

De scores worden als volgt berekend:

- Strike: De som van de eerste worp (10) plus de volgende twee worpen daarna.
- Spare: De som van de eerste twee worpen (10) plus de eerste worp daarna.
- Open: Enkel de som van de twee worpen van de beurt.

Voorbeeld:

1. Beurt 1: worp 1: 2 pinnen om, worp 2: 3 pinnen om.
2. Beurt 2: worp 1: 10 pinnen om, strike!
3. Beurt 3: worp 1: 8 pinnen om, worp 2: 2 pinnen om, spare!
4. Beurt 4: Worp 1: 6 pinnen om, worp 2: 3 pinnen om.
5. Alle overige beurten worden geen pinnen omgegooid.

De score is dan (2+3) + (10+8+2) + (8+2+6) + (6+3) = 50.

Implementeer nu een `struct` genaamd `BowlingGame` met één associated function en twee methodes (methods):

1. Een associated function `new()` om een nieuw spel aan te maken.
2. Een methode `frame(&mut self, roll1: u32, roll2: u32)` met als parameters het aantal pinnen wat is omgegooid in de eerste en tweede worp van deze beurt. Deze methode geeft als returnwaarde een `Result <u32, Error>`. De `u32` is het aantal tot nu toe behaalde punten. De `Error` is een `enum` met in ieder geval de volgende waarden: `InvalidFrame` en `GameAlreadyEnded`. Deze methode wordt 10 of 11 keer aangeroepen, afhankelijk van de situatie in beurt 10.

3. Een methode `end_score(&self)` die het aantal behaalde punten teruggeeft als het spel (game) afgelopen is. Als het spel nog niet afgelopen is, geeft deze methode 0 terug.

Een testprogramma vind je hier.