1-3: First Rust Project (Theory)

Artem Pavlov, TII, Abu Dhabi, 03.05.2024

Crates

- Crate is a compilation unit for Rust
- Cargo.toml file is a “manifest” which describes the crate
- Cargo.lock file contains exact dependency versions and their hashsums used by a crate (or workspace)
- Crates can be divided into library and application crates
- Library crate contain src/lib.rs
- Application crates contain src/main.rs

Manifest format

- TOML file with the format described in https://doc.rust-lang.org/cargo/reference/manifest.html

Workspaces

- Crates can be part of “workspace”
- Workspace is a collection of crates with common list of resolved dependencies and cache of build artifacts (the target/ folder)
- Workspace can be “virtual”, i.e. not tied to any particular crate
- Cargo.toml in the repository root creates “virtual” workspace

Hybrid crates

- Crates can be simultaneously library and application, i.e. contain both src/main.rs and src/lib.rs
- The latter can use items imported from the latter and from the crate dependencies

Crate binaries

- Crate can contain additional binaries in src/bin/ and examples/ folders
- They can be build by running cargo build --bin bin_name and cargo build --example example_name
- The binaries can use items from crate library and its dependencies

Crate tests

- Crate can contain unit and integration tests
- Unit tests usually reside near tested code (i.e. in src/ folder) and can test private API
- Integration tests test public API of the crate and usually reside in the tests/ folder
- Additionally crates can have documentation tests (i.e. pieces of code provided in documentation)
- All tests can be executed using cargo test

Crate benchmarks

- Crate benchmarks reside in the benches/ folder
- Benchmarks can be executed using cargo bench
- Note that the default benchmark engine is currently unstable and available only on Nigthly
- A more advanced benchmarking facility is available in the criterion crate

Primitive types: integers

- Unsigned integers: u8, u16, u32, u64, u128, usize
- Signed integers: i8, i16, i32, i64, i128, isize
- Size of usize and isize depends on compilation target
- Integers can be created using literals (e.g. 42 or 0xA4)
- Literals do not have a type per se
- For better readability it’s recommended to use underscore with big literals, e.g. 10_000_000 or 0xABCD_1234
- Rust has no automatic type coercion!

Primitive types: boolean

- bool type which can be in two states: true and false

Operations over integers

- Operators:
  - + for addition
  - - for subtraction
  - * for multiplication
  - / for division
  - % for remainder
- Overflow behavior: panic in debug builds and wrapping in release builds
- Types also contain inherent methods like wrapping_add

Variables

- Variables are defined as `let x = 42;`
- Type can be explicitly annotated as either `let x: u32 = 42;` or `let x = 42u32;`
- Compiler is able to infer types:

```
let x = 42;

let y: u32 = x;
```

Function signatures

- Function signature defines function name, accepted arguments, and return type
- Function arguments act as variables for its body

```
fn add_one(x: u32) -> u32 {
    x + 1
}
```

Late initialization

- You can postpone variable initialization:

```
let x: u32;
x = 1;
```

- But compiler will not allow to use uninitialized variable:

```
let x: u32;
let y = x + 1;
```

Mutable variables

- If variable is mutated, then it should be defined as:

```
let mut x: u32 = 0;
x += 1;
```

- Similarly for functions:

```
fn add1(mut x: u32) -> u32 {
    x += 1;
    x
}
```

Modules

- Defined using mod keyword
- Can be either blocks mod { … } or point to a different file mod foo; will look for foo.rs
- If module foo.rs defines submodule bar, then compiler will look for file foo/bar.rs

Visibility

- By default items are visible only in current module and its submodules
- Visibility can be modified by adding:
  - pub: makes item fully public
  - pub(crate): public for whole crate
  - pub(super): makes item public for module above

Importing

- Items are imported using use keyword
- To make imported item visible in the module pub use can be used
- Import happens either relative to the current module, from current crate, or from external crate

Questions?