

4-1: Iterators, stdlib collections, IO (Theory)

Artem Pavlov, TII, Abu Dhabi, 17.04.2025

The `Iterator` trait

- The `Iterator` trait has `Item` associated type and `next` method
- `next` method returns `Option<Self::Item>`
- `Some(Item)` is returned as long as there are elements, and once they've all been exhausted, will return `None` to indicate that iteration is finished

Creating iterators from collections

- Collections commonly implement 3 methods:
 - `iter()`, which iterates over `&T`
 - `iter_mut()`, which iterates over `&mut T`
 - `into_iter()`, which iterates over `T`

for loops and Intolterator

- When you use `for value in values { .. }` language implicitly uses the `Intolterator` trait to transform values
- Meanwhile, `value` is a pattern which can be used to destructure iterated values
- Note that iterators implement `Intolterator` via the blanket impl `impl<I: Iterator> Intolterator for I`

Vec and for loop

- `Intolterator` takes `self` by value
- Passing owned vector to `for` loop results in iteration over owned values taken from vector
- After the `for` loop finishes, the vector will be deallocated

Iterator adapters

- **Iterator** provides a big number of “iterator adapters”:
<https://doc.rust-lang.org/std/iter/trait.Iterator.html>
- Most of iterator adapters are “lazy” and do not do any work until “polled” with **next** or **for_each**
- Ranges implement **Iterator** and can be used with adapters
- Additional adapters can be found in third-party crates, e.g. **itertools**

std collections

- Rust provides efficient implementations of the most common general purpose programming data structures
- Using common collection types is important for interoperability
- Provided collections:
 - Sequences: `Vec`, `VecDeque`, `LinkedList`
 - Maps: `HashMap`, `BTreeMap`
 - Sets: `HashSet`, `BTreeSet`
 - Misc: `BinaryHeap`

Vec<T>

- A contiguous generic heap-allocated growable array type
- Zero-length **Vecs** do not perform any allocations
- Can be created using inherent method or with **vec!** Macro
- It's recommended to pre-allocate capacity using **Vec::with_capacity** or with the **reserve** method

String

- A UTF-8–encoded, heap-allocated growable string.
- String does not use the “small string optimization”, use third-party crate for it (e.g. **smallstr**)
- **String** effectively is a thin wrapper around **Vec<u8>**

HashMap

- A generic **HashMap**
- Implementation based on **SwissTable**
- By default uses DoS-resistant randomly seeded hash function
- Key type must implement **Eq** and **Hash** traits (warning: correctness of **HashMap** relies on correctness of these impls)
- It's recommended to pre-allocate capacity using **HashMap::with_capacity** or with the **reserve** method

Result helper methods

- **Result** provides a number of helper methods which help with error handling:
 - **map**: maps **Ok** value using provided closure
 - **map_or**: Returns the provided default (if **Err**), or applies a function to the contained value (if **Ok**)
 - **map_err**: maps **Err** value using provided closure
 - **and_then**: calls closure if the result is **Ok**, otherwise returns the **Err** value of **self**
 - **ok**: converts from **Result**<T, E> to **Option**<T>
 - **err**: converts from **Result**<T, E> to **Option**<E>

Option helper methods

- **and_then**: returns **None** if the option is **None**, otherwise calls **f** with the wrapped value and returns the result
- **filter**: returns **None** if the option is **None**, otherwise calls provided filter function
- **flatten**: converts from **Option<Option<T>>** to **Option<T>**
- **map**: maps an **Option<T>** to **Option<U>**
- **ok_or**: transforms the **Option<T>** into a **Result<T, E>**, mapping **Some(v)** to **Ok(v)** and **None** to **Err(err)**

Basic `io` traits

- IO with “streaming” sources is done using `io::Read`, `io::Write`, and `io::Seek` traits
- They have only one-two required method and a number of provided methods
- Warning: note that `Read::read` and `Write::write` may read and write smaller amount of bytes than requested/provided!
- `std::io` module also provides utilities which work on top of these traits (e.g. `io::copy`)

BufReader and BufWriter

- In many cases it's beneficial to buffer IO
- Rust std provides buffering helpers **BufReader** and **BufWriter** which wrap implementors of **io::Read/Write**
- **BufReader** and **BufWriter** implement **io::Read** and **io::Write** respectively
- **BufReader** also provides additional functionality, e.g. the **lines** method

Stdin, Stdout, and Stderr

- Can be constructed using `io::stdin()`, `io::stdout()`, and `io::stderr()` respectively
- Can be explicitly locked for exclusive access
- `Stdin` has `read_line` helper method for reading input lines
- Locked `Stdin` implements the `BufRead` trait and provides the `lines` method
- `println!` macro locks stdout on each execution
- You can use `eprintln!` macro for quick printing into stderr

std::fs::File

- An object providing access to an open file on the filesystem.
- “Owns” the underlying file descriptor
- Automatically closed when it goes out of scope
- Methods for opening and creating files are generic over path type using `AsRef<Path>`, which is satisfied by `String` and `&str`

std::net::{TcpStream, TcpListener}

- **TcpListener** is a TCP socket server, listening for connections.
- **TcpStream** is a TCP stream between a local and a remote socket.
- **TcpListener** can act as an iterator of incoming **TcpStreams** using the **incoming** method

Questions?