Input/output in Rust

Reading from the standard input

```
pub fn main() {
    let mut input = String::new();

    std::io::stdin()
        .read_line(&mut input)
        .expect("Input could not be read");

    // use input
}
```

Reading from a file (whole content)

```
use std::fs;

fn main() {
    let data = fs::read_to_string("/etc/hosts").expect("Unable to read file");
    println!("{}", data);
}
```

stolen from https://stackoverflow.com/a/31193386

Reading from a file (line by line)

```
use std::fs::File;
use std::io::{BufRead, BufReader};

fn main() {
    let f = File::open("/etc/hosts").expect("Unable to open file");
    let f = BufReader::new(f);

    for line in f.lines() {
        let line = line.expect("Unable to read line");
        println!("Line: {}", line);
    }
}
```

Buffer: good practice

Can be used with `for`

stolen from https://stackoverflow.com/a/31193386

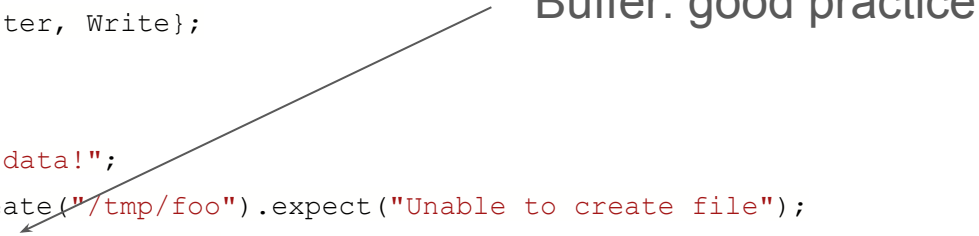
Writing to a file

```
use std::fs::File;
use std::io::{BufWriter, Write};

fn main() {
    let data = "Some data!";
    fs::write("/tmp/foo", data).expect("Unable to write file");
}
```

stolen from https://stackoverflow.com/a/31193386

Writing to a file

```
use std::fs::File;
use std::io::{BufWriter, Write};

fn main() {
    let data = "Some data!";
    let f = File::create("/tmp/foo").expect("Unable to create file");
    let mut f = BufWriter::new(f);
    f.write_all(data.as_bytes()).expect("Unable to write data");
    f.flush()
}
```



stolen from https://stackoverflow.com/a/31193386

I want to take argument from my program

Two solutions:

- args (not recommended)
- clap (recommended)

The “arg” way (avoid doing that)

```
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();

    let query = &args[1];
    let file_path = &args[2];

    println!("Searching for {query}");
    println!("In file {file_path}");
}
```

https://doc.rust-lang.org/book/ch12-01-accepting-command-line-arguments.html

The “clap” way (prefer using this)

```rust
use clap::Parser;

#[derive(Parser)]
#[command(version, about, long_about =None)]
struct Args {
   /// File to search the query in
   #[arg(short, long)]
   file_path: String,
   /// Query
   #[arg(short, long)]
   query: String,
}

fn main() {
   let args = Args::parse();

   println!("Searching for {}", args.query);
   println!("In file {}", args.file_path);
}
```

use /// for documentation

clap:

- communicates your intent
- adds a help
- prints your description from your cargo.toml

Parsing struct to json

```
struct Address {
    street: String,
    city: String,
}
```

Parsing struct to json

```
use serde::{Deserialize, Serialize};

#[derive(Debug, Serialize, Deserialize)]
struct Address {
   street: String,
   city: String,
}
```

Parsing struct to json

```
fn main() {
    let address = Address {
        street: String::from("10 Downing Street"),
        city: String::from("London"),
    };
    let filename = "adress.json";

    // Serialize to a JSON string and write to a file
    let json: String = serde_json::to_string(&address).expect("Unable to parse to string");
    std::fs::write(filename, json).expect("Unable to write file");

    // read from file and parse as JSON
    let json_from_file: String = std::fs::read_to_string(filename).expect("Unable to read file");
    let adress_from_json: Address = serde_json::from_str(&json_from_file).expect("Unable to parse file");

    println!("{:?}", adress_from_json)
}
```

Take home message

Use the libraries instead of reinventing the wheel