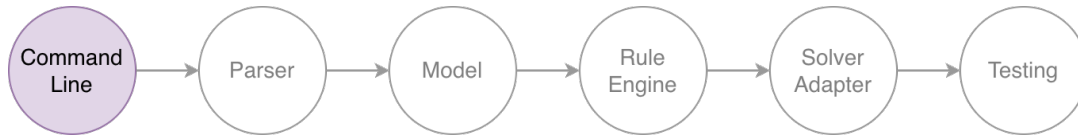


Oxidelings: **Command Line**

Task 1 of 6

Last Modified 17/12/2025



1 Background

The first stop on our journey is the **Command Line Interface** (CLI). The `conjure-cp-cli` crate contains all the source code for the interface, so this is where our work will occur for this task. Within the `conjure-cp-cli/src` directory, `main.rs` is the entry point for execution. If we follow the general flow:

1. `main()` calls `run()`
2. `run()` parses the command line arguments...
3. ...and then passes off the details it found in parsing to the `run_subcommand()` function
4. `run_subcommand()` matches to a structure that implements Clap's `Subcommand` trait ...
5. ... to decide the specific function to run, and runs it with the corresponding arguments.

Clap is a public crate for implementing command line interfaces. It does clever things like take the comments above a function and display that on the help page, and take a lot of the boilerplate out of parsing very standard command line structures. We use structs, enums, and traits to provide clap with all the information it needs. More info on clap can be found [here](#)

Structs Similar to C++ (at least on the surface), `structs` store various fields. We use structs in `cli.rs` to create structures that can store the arguments the user has passed.

Enums Define a type that can be one of many variants. We use enums in `cli.rs` to enumerate the various command options available.

Traits Define what behaviour that types can implement. So our `Cli` struct implements `Parser`, meaning that we can tell clap some general information about the interface we want to provide. The `Command` enum implements `Subcommand`, meaning clap knows about the various command options.

2 The Task

Oh no! Hexer-the-Vexer (our mischievous scoundrel for this story) has infiltrated the repository and wreaked havoc. Hexer has removed the ability to run the `solve` command; how will the people solve?! We need **you** to reimplement it so that all of the conjurers can keep solving combinatorial problems.

Files: You will only need to edit: `main.rs`, and `cli.rs`. The solution is only a few lines total.

There are hints on the next page, so if you are struggling do not fret! You will get the most out of it if you give it a go first though, so hold off on turning the page just yet.

3 Hints

3.1 Hint A

We use `Clap` to provide the command-line interface, and this functionality is added by `clap::Args`, `Parser`, `Subcommand`. The enum that implements `Subcommand` represents the different commands available, so this will need extended.

4 Guide

You can use the below steps to fully implement this challenge to how it is in the actual repository. Please only use this after making an attempt yourself.

Telling Clap about the Command: To properly add the command, clap needs to know that it is an option. This is done in the `pub enum Command {}` in `cli.rs`, which implements `Subcommand` from the clap crate.

1. in `cli.rs`, go to `pub enum Command {}` and add `Solve(solve::Args)`, to create a command variant for 'solve' that takes the solve arguments
2. add a line above such as `/// Solve a model` so that the `-help` option returns useful information.

Extending the match statement: now we have an additional variant in the enum, we need to add the requisite arm to the match statement such that when someone *does* call `solve`, it actually calls the `run_solve_command()` function.

1. Go to `fn run_subcommand()` in `main.rs`
2. Add an arm to the match statement to catch the new `Solve(solve_args)` variant, and call `run_solve_command()` function.