# Systems Programming Project 1

March 19, 2024

## 1 Hashassin

For Projects 1 and 2 we will be working on a library (and front end) called *Hashassin*. Hashassin is a tool for generating and using hashes. For Project 2, you will be adding functionality to handle rainbow tables, but for Project 1 we are going to be working on the basic building blocks.

You can accept the GitHub classroom assignment at `https://classroom.github.com/a/B4CBWnvw`.

This is a group project with a maximum of *three* students per group.

## 2 Instructions

This project is relatively straight forward as we are mostly interested in making sure that you are able to write a relatively complicated program that uses threading and deals with IO. To that end, you will be creating a workspace with two crates, one binary and one a library.

Your binary crate should be in a directory called `cli` and it should compile to a binary named `hashassin`.

Your library crate should be in a directory called `core` and it should be available called `hashassin-core` and usable by other crates via `use hashassin_core::Whatever`.

Your CLI will support two commands: 1) `gen-passwords` and 2) `gen-hashes`.

`gen-passwords` will generate random passwords and save them to a file. `gen-passwords` *must* have several options that can be set:

1. `--min-chars`, which specifies the minimum number of characters to be used in generated passwords. This value *must* be greater than zero and the maximum value should be the maximum length of an array on whatever system the program is being run on. `--min-chars` should have a default value of 4.

2. `--max-chars`, which specifies the maximum number of characters to be used in generated passwords. This value *must* be greater than zero and the maximum value should be the maximum length of an array on whatever system the program is being run on. It must also be greater than or equal to whatever the `--min-chars` value is. `--max-chars` should have a default value of 4.

3. `--out-path`, which, *if present*, will write the output of the command to the specified file, one passwords per line (using Unix new lines and no empty lines between them). If not present, results should be written to `stdout`.

4. `--threads`, the number of threads to use to generate passwords. This value *must* be greater than zero and the maximum value should be the maximum length of an array on whatever system the program is being run on. `--threads` must have a default value of 1.

5. `--num-to-gen`, which is the number of passwords to generate. It must be greater than zero and the maximum value should be the maximum size of an array on the system it's being run.

All passwords generated should be valid ASCII, both caps and lower case, and including punctuation and spaces but *not* including non-printable characters like
`n` or
a.

**NB:** You can add additional functionality that can restrict or expand the set of characters to be used in generated passwords, but by default, generated passwords should adhere to the above guidelines.

`gen-hashes` will generate hashes from a set of input passwords. `gen-hashes` *must* have several options that can be set:

1. `--in-path`, which specifies the path to read plaintext passwords from.

2. `--out-path`, which, *if present*, will write the output of the command to the specified file, one hashed password per line (using Unix new lines and no empty lines between them). If not present, results should be written to `stdout`

3. `--threads`, the number of threads to use to generate hashes. This value *must* be greater than zero and the maximum value should be the maximum length of an array on whatever system the program is being run on. `--threads` must have a default value of 1.

4. `--algorithm`, which should be the algorithm used to generate hashes.

You are free to add other options and functionality, but whatever you add should not be required to be set by a user. I.e., it should either be optional or have sensible defaults.

## 3 Grading

This project is relatively straight forward. You will receive points according to the following list. Please note that there are more than 100 points available. Also please note that some items on the list are *required* to be implemented by groups with at least one graduate student and do not grant additional points (undergrad only groups can complete these for additional points).

- -1,000 points: If you do not update `CREDITS.md` with the names of your group members, as well as an honest break down of the work each group member did, you will receive **NEGATIVE ONE THOUSAND** points. I.e., it would be next to impossible to get anything higher than a zero on this project.
- -1,000 points: If running `cargo fmt` results in any change to your repository, you will get `NEGATIVE ONE THOUSAND` points. Be sure to run `cargo fmt`!!!!!!!!!
- 25 points: Program compiles, as well as the items noted below as required for graduate groups.
- 20 points: **All** required functionality of `gen-passwords` is implemented (e.g., setting `--threads` actually uses multiple threads, etc.)
- 20 points: **All** required functionality of `gen-hashes` is implemented (e.g., setting `--threads` actually uses multiple threads, etc.).
- 5 points: Comprehensive documentation. Points will be determined by looking at the documentation built when running `cargo doc --document-private-items --no-deps`.
- 10 points: No warnings from `cargo check` with no use of directives that would suppress default warnings (e.g., `#[allow(dead_code)]`)
- 10 points: No warnings from `cargo clippy` with no use of directives that would suppress default warnings (e.g., `#[allow(dead_code)]`)
- 5 points: Support one hashing algorithm.
- 0.25 points: Support an additional hashing algorithm (0.25 points per algorithm)
- 5 points: **REQUIRED FOR GRADUATE GROUPS.** Proper error handling. Create and use proper error types in library code and appropriate handling in any user code (e.g., CLI).
- 5 points: No unwraps or expects. Add `#![deny(clippy::unwrap_used, clippy::expect_used)]` to the top of your `main.rs` and `lib.rs`. If running `cargo clippy` doesn't result in any errors, then you get the 5 points.
- 5 points: **REQUIRED FOR GRADUATE GROUPS.** Add proper logging. Using the `tracing` crate to generate logs. Points will be determined on comprehensiveness of logs, as well as the usage of different log levels.

- Unlimited points: Cool factor. Points are determined subjectively by me, but if I find anything about your project to be particularly unique, difficult, clever, etc., you can get some extra points. Be sure to point out anything we should pay special attention to in your `README.md` file.

**In addition to the above, you must have a README.md to summarize what you did and give any and all instructions on how to run things! I'm going to go off what's in the README.md to grade, so make sure you tell us everything you did and test all the instructions, etc.**

## 3.1 Allowed Crates

Generally speaking, you are free to use any crates listed on `https://blessed.rs`.

Any crates not on Blessed.rs need to be approved by Jeremy.

Be sure to note what crates you use in your `README.md`!

## 3.2 Due Date

April 3rd, 6:00 AM.

**NB:** Write access to your repositories will be revoked at the deadline!

## 3.3 Academic Honesty Statement

In addition to the code in your repository, you *must* include an academic honesty statement as per the class syllabus. If you do not include this academic honesty statement in your repository, you will receive a *ZERO* on the project.

**REMEMBER THAT IF YOU USE A MACHINE LEARNING TOOL THAT IS CHEATING, YOU WILL RECEIVE A ZERO ON THE PROJECT (i.e., your max grade in the class will be a 66%, which is an F), AND THE CLASS WILL REVERT TO IN CLASS TESTS AND QUIZZES!!!!!**