This document describes a Rust program, sim, for implementing a cache simulator.
Written by Andy Davis for the University of London's Computer Systems module within the Computer Science Masters Program.

The program reads a valgrind trace file containing memory accesses simulates the behavior of a cache based on user-specified cache parameters and outputs statistics about the cache performance.

The program defines three structures: Blocks, Sets, and Cache.

- Blocks represents a single block in the cache and stores its validity and tag.
- Sets represents a set in the cache and stores a vector of Blocks structures.
- Cache represents the entire cache and stores a vector of Sets structures, set bits from the argument flag, block bits from the argument flag, number of sets calculated from the set bits, hits, misses, evictions, event string for verbose reasons, and finally the function to check the cache.

The program consists of only a couple of functions that will be described on a general level; please see the code documentation/comments for further information as to what each step of the function does.

The check_cache function, from the Cache structure, takes an address as an input and does the heavy lifting for the program. If the address is found in the cache, the function updates the cache's hit value as well as updating the event to indicate said hit. If the address is not found, the function updates the block's valid and tag values as well as updating the cache's miss value and event to indicate the miss. If the block's valid value was previously marked as true, then an eviction even happens, updating the cache's eviction value and event to indicate said eviction.

The cache_sim function takes a trace file and the command line arguments as inputs, initializing the cache and looping through the lines of the trace file. Starting with initializing the Cache structure using the required command line arguments as the structure's arguments. For each line of the trace file, the function parses out each element (denoted from the brief) to determine what type of operation is performed as well as the address. Depending on the operation of the line, the check_cache function will either be called once or twice. At the end of each loop, if the verbose argument was included, the function will print out the line from the file plus what events (from Cache.events) had happened to that line. Finally, the function will report the findings of each hit, miss, and eviction.

As for determining the command line arguments, there are the optional_flags and print_usage functions. The optional_flags function takes an array of arguments (collected from the main function) and attempts to parse through the array, using the getopts crate to help deisgnate

said command line arguments and check if they are included. The print_usage function is used for the help argument or if any arguments are missing, printing out an example of how to run the program and exits at the start.

Finally, there is the main function. This function collects the command line arguments into a list for the optional_flags function and tries to open up the trace file. If the trace file was incorrect, then the function quits. If everything is ready to go, the function then calls the cache_sim function to begin the work.

Overall, this program provides a basic cache simulation framework that can be customized by adjusting the cache parameters.

In regards to writing this program, my main IDE was IntelliJ since there is a wonderful Rust extension. I was able to set up a couple of configurations to test different command line arguments so that testing was as easy as switching between the configurations. This assignment probably took me more time than I would have liked, being a subject that for whatever reason my brain just did not want to comprehend. There are a couple of links included in my comments under the check_cache function in which I used as reference to help build my code. Even reading multiple chapters of books and talking to teachers and students one on one, I am still uncertain on how everything works properly (mostly the LRU part that I had implemented, that one could be better). Sure, there were some struggles with writing in Rust, but that is common when using a new language for the first time, but this language is one of the easiest to pick up (given that when you build the file it gives you the clearest understanding of what issue you are having). There was way more trial and error and I would have liked, and at one point I was just going to give up and turn in an uncompleted assignment, but I am glad I stuck it out and ended with something that I can be happy with turning in.