170D WOBC Module J Practical 1B

Pottery Failure

A group of people are taking a pottery class led by a sadistic instructor. n potters are sitting in a circle next to each other, numbered from 0, 1, 2, …, n–1.

The instructor must check each student's work, and will do so by going around the room in order from a starting student. It takes 5 minutes for the instructor to check a single student's work. This means that some students will have more time to finish than others: if the instructor starts at student f, then f+1 has 5 extra minutes to finish, and f+2 has 10 extra minutes, and so on. If the instructor gets to potter n–1, they keep moving in the same direction, checking potter 0, then potter 1, etc., until all projects have been checked.

The goal is to maximize the number of students who do not finish before the instructor starts evaluating their work. Write a program that will print out whom the instructor should start with, and a list of people who are unable to finish their projects as a result of that starting person.

DIcE Rubric

Document	Design Plan	Does the design plan provide a clear general overview of the project?	3%
		Is the design plan easy to understand?	2%
	Project Writeup	Does the writeup document challenges and successes encountered?	2%
		Does the writeup document any lessons learned?	3%
	Writing	Is the project free of grammatical and spelling errors?	4%
		Is non-code formatting consistent?	1%
	Code Formatting	Does `indent -linux` produce no warnings?	4%
		Are appropriate names chosen to enable code readability?	2%
		Are comments added where appropriate and aid understanding of the logic?	2%
		Is any outside code cited appropriately?	2%
	Total		**25%**
Implement	Version Control	Does the project have the correct name and default branch?	1%
		Were commits broken down into appropriate scopes?	3%
		Are commit messages simple and informative?	1%
	Architecture	Are effective and efficient data structures used?	5%
		Was the code designed and constructed in a modular fashion?	5%
		Were generally sound decisions made with regard to architecture?	15%
	Total		**30%**

Execute	Safety	Does the program avoid crashing or infinite loops, even on invalid input?	5%
		Does `valgrind` report no errors or warnings?	5%
	Builds	Does the program build with no warnings?	5%
		Do all required build targets get built correctly?	5%
	Requirements	Were all inputs parsed correctly and yield the correct output?	10%
		Are all other requirements met?	10%
	Performance	Does the program scale appropriately with input and data?	4%
		Does the program execute in a timely manner?	1%
	Total		**45%**

Requirements

Area	Requirement
Document	All documentation must be in PDF format unless otherwise specified.
Document	All documentation must be located in a `doc/` folder at the top level of the project.
Document	The design document must be located in `doc/design.pdf`
Document	The project writeup must be located in `doc/writeup.pdf`
Document	All code must match the Linux kernel style guide, with the exception of blocks *always* having braces.
Implement	Project must be stored in the assigned VCS account, under the project name `potter`.
Implement	Each logical portion or feature must be built in its own branch.
Implement	Merge (without fast-forwarding) all branches to master branch and tag releases appropriately.
Implement	The default branch to clone should be master.
Implement	No third-party header files/libraries may be used unless signed off by the Program Manager or Instructor.
Implement	Project must use appropriate data types or structures.
Implement	Project must use functions appropriately.
Implement	If present, all automated tests and test code must be located in a `test/` folder at the top level.
Implement	The instructor must pick a starting person that yields the largest number of students who fail to finish their project.
Implement	If multiple starting students would yield the same, largest amount of incomplete projects, pick the potter that occurs first in the input.

Area	Requirement
Execute	Project must build and run on the class machine.
Execute	Project must not crash or get stuck in an infinite loop, even on invalid input.
Execute	Project must build `potter` in the top-level directory in response to `make`.
Execute	Project must build `potter` with debugging symbols in response to `make debug`.
Execute	If present, project must build and run any automated tests in response to `make check` (which may assume the program has already been built).
Execute	Project must clean all project-generated files in response to `make clean`.
Execute	Project must build against C18 (`-std=c18`)
Execute	Project must build with no warnings from `-Wall -Wextra -Wpedantic -Waggregate-return -Wwrite-strings -Wvla -Wfloat-equal`
Execute	Program must be invoked as `potter [file]`, where *file* is a list of students and their time left to finish their individual project, in minutes.
Execute	When invoked with no arguments, the program must read the list of potters from `stdin`.
Execute	Invoking the program with an invalid or incorrectly formatted input must produce an appropriate error message.
Execute	Program must follow the required output format.

Input/Output Formats

The format of the input to the program is one name per line, followed by some amount of whitespace, followed by the number of minutes that name needs to finish their project, followed by a newline.

Example Input

```
Herodotus           11
Carl    Sagan       31
Hassan-i Sabbah 21
Tommy Chong          14
Seth Rogen           8
Hunter S. Thompson   0
```

The output must first be the name the instructor should start checking work at. After a blank line, the rest of the output should be, starting from that initial name, all names that are unable to finish work prior to the instructor visiting them. No other output (other than error messages) is permitted.

If the initial collection point is unable to finish their project (i.e., their time is not 0), their name should appear twice: both as the initial point and the first name to be unable to finish.

Example Output

```
Seth Rogen

Seth Rogen
Herodotus
Carl    Sagan
Hassan-i Sabbah
```

Suggested Extra Credit Features

Area	Feature	+
Document	Write man(1) page to document the program.	+2
Document	Write the writeup and design plan using TeX or LaTeX language (be sure to include the source files in the doc directory).	+2
Document	Write a test plan. This will become a requirement in the next exam.	+2
Implement	Write automated unit tests. This will become a requirement in the next exam.	+3
Execute	Add support for the `-s start` flag. The instructor must start checking at the first person named `start`, regardless of how many can finish their project in time.	+3