CS 240: Programming in C
Spring 2025
Homework 2

Due February 5, 2025 at 9:00 PM

1 Goals

The purpose of this homework is to get you acquainted with file I/O in C.

2 Getting Started

1. Log in to one of the Purdue CS Linux systems. These include `data.cs.purdue.edu` and `borgNN.cs.purdue.edu` where `NN`=01,02, etc
2. Inside your `cs240` directory, setup a `hw2` directory by running the following commands...

   ```
   $ git clone ~cs240/repos/s2025/$USER/hw2.git
   $ cd hw2
   ```

3. This creates your hw2 directory, and also copies what is called a **Makefile** into it. This Makefile has a number of targets defined to make your life easier...

   When you are ready to try your work, you can run:

   ```
   $ make hw2_main
   ```

   to build hw2_main, or run:

   ```
   $ make hw2_test
   ```

   to build the test module. Or, just run:

   ```
   $ make
   ```

   to build both. For this assignment, the test module will produce a number of files in your directory. Running it many times will result in many files. You can run:

   ```
   $ make clean
   ```

   to clear out these temporary files.

   Note: any time you run `make`, your changes are automatically pushed to a Git repository in the `cs240` course account. As mentioned before, we **expect** you to always work on a Purdue CS Linux system. This mechanism cannot work otherwise.

3 The Big Idea

Purdue IT (PIT) provides various computer labs across Purdue's campus for students and faculty. Each lab tracks some statistics to help PIT gain insights about lab usage. The statistics store, for each login session, the date, lab name, username, session duration, number of applications accessed, most used application, and CPU-minutes used.

For this assignment, you will write functions that analyze PIT lab usage data. Your code will help PIT generate reports to optimize lab resources and ensure efficient use of computers and applications.

4 The Assignment

4.1 Functions you will write

For each of the below functions, the first argument contains the name of the input file you should read data from. See subsection 4.2 for the input file format.

```
int count_logins(char *input_file, char *lab, int year, int month, int day);
```

This function returns the number of logins (i.e. sessions) the given lab's computers facilitated on the specified day. In case of an error, return an appropriate error code as described in subsection 4.3.

```
double total_cpu_usage(char *input_file, char *lab, int start_date,
                       int end_date);
```

This function will compute the total CPU usage for the given lab within a range of dates. Since the total CPU usage may be a large number, convert the final value from CPU-minutes to CPU-hours.

Both date arguments are integers in the format `yyyymmdd`. For example, May 19, 2005 would be the number 20050519. The date range is inclusive.

Return the total CPU-hours used, or an error code as described in subsection 4.3.

```
double editor_wars(char *input_file);
```

Purdue IT is in shambles as its employees have engaged in a fierce war over the best text editor. Obviously, the right answer is Vi(m) — but they won't listen to you. To prove your stance, this function will analyze the lab session data to show empirically that Vi(m) is better than Emacs.

The popularity of each editor will be defined as:

$$\text{popularity} = \sum \frac{\text{session\_duration}}{\text{processes\_run}}$$

All sessions whose most popular program includes "vi" in the name will count towards Vi(m)'s popularity. Those which include "macs" will count towards Emacs' popularity.

Return the difference between Vi(m)'s and Emacs' popularity scores. If an error occurs, return an error code according to subsection 4.3.

```
int blame_the_kids(char *input_file);
```

Purdue IT's purchasing department noticed they've had to buy an unusally high number of CPUs lately. They suspect that one of the CS 240 students is purposely overwhelming the lab computers as a way to protest the code standard rules.

To help find the culprit, this function will find the student who has the highest total CPU usage. For this function only, the input file will be arranged such that all lines with the same username will be adjacent.

Return the (1-indexed) line number at which that student's username first appears, or an error code if appropriate. Break ties by returning the first tied student.

```
double efficiency_score(char *input_file, char *lab);
```

This function calculates an "efficiency score" for the given lab. The efficiency for the lab is defined as the average efficiency of each login session within that lab.

The efficiency of a single login session is defined as:

$$\text{efficiency} = \frac{d - p^2}{c}$$

where d is the session duration, p is the number of processes run, and c is the CPU usage in minutes.

Return the lab's efficiency score, or an error code according to subsection 4.3.

```
int generate_report(char *input_file, char *output_file, char *lab,
                    int start_date, int end_date);
```

This function generates a report containing a summary of the information in the input file. This report should be written to the file whose name is specified by the second argument.

The report written to the file should have the following format. Angle brackets (</>) denote a placeholder for a value. Print floating-point values with 2 decimal places of precision.

```
Lab: <lab>
Start date: <yyyy>-<mm>-<dd>
End date: <yyyy>-<mm>-<dd>

Total logins: <logins>
Total CPU usage (hours): <cpu-hours>
Average CPU usage per login (minutes): <avg-cpu>
Processes executed: <processes>
Efficiency score: <efficiency>

Programs used:
<progs>
```

For `<processes>`, output the total number of processes run.

For `<progs>`, output the name of the most used program for each login session. Print one program name on each line. You do not need to de-duplicate the names. Output them in the order the sessions appear in the input file.

Return an error code as described by subsection 4.3.

4.2 Input File Format

The input file will contain records of the following format. Each line represents one login session.

```
yyyy-mm-dd,"lab_name",username,duration,procs_run,"top_program",cpu
```

- `int yyyy`: Year. Must be positive.
- `int mm`: Month. Must be between 1 and 12, inclusive.
- `int dd`: Day. Must be between 1 and 30, inclusive.
- `char lab_name[MAX_NAME_SIZE]`: Lab room name.
- `char username[MAX_NAME_SIZE]`: Username of student.
- `double duration`: Duration of the session in minutes. Must be positive.
- `int procs_run`: Number of processes run. Must be positive.
- `char top_program[MAX_NAME_SIZE]`: Name of the program used the most during this session.
- `double cpu`: CPU minutes used. Must be positive.

Here are a few sample rows:

```
2024-10-25,"LWSN B148",turkstra,53.56,45,"vi",2.69
2024-10-26,"DSAI B069",may5,40.53,33,"gcc",36.34
2024-10-27,"HAAS G056",grr,5.0,2,"man",0.1
```

4.3 Error Codes

- `SUCCESS`: no errors encountered.
- `FILE_READ_ERR`: Indicates that the file cannot be opened for reading.
- `FILE_WRITE_ERR`: Indicates that the output file cannot be opened for writing.
- `NO_DATA`: Indicates that the input file contains no data, or that there are no entries that match the specified search contraints.
- `BAD_RECORD`: Indicates that a record contains unexpected characters, fields, or the wrong data type. This does not apply to dates; use `BAD_DATE` for that instead.
- `BAD_DATE`: Indicates that a date in the input file or a date passed in as an argument is invalid. Dates must have a positive year, a month between 1 and 12, and a day between 1 and 30.

These Standard Rules Apply

- You may add any `#include`s you need to the top of your `hw2.c` file.
- You may not create any global variables other than those that are provided for you.
- You should check for any failures and return an appropriate value.
- Do not assume any maximum size for the input files.

Special Rules

- To ensure you get comfortable with format strings, the only file I/O functions you are allowed to use for this assignment are `fopen()`, `fclose()`, `fscanf()`, `fprintf()`, and `rewind()`.

- For this assignment, you will have to read a string name from a file into a fixed-size buffer. You may hard-code the value of the maximum length of that string into the format specifier for `fscanf()`.

5 Submission

To submit your program for grading, type:

```
$ make submit
```

In your `hw2` directory. You can do this as often as you wish. Only your final submission will be graded.

5.1 Grading

The operation of your functions will be graded out of 100 points. The point breakdown will be determined by the test program.

This homework will also have a style grade based on 20 points, with 2 points deducted for each code standard violation found.

Your code must compile successfully using `-Wall -Werror -std=c17` to receive any credit. Code that does not compile will be assigned an automatic score of 0.

If your program crashes (e.g., segmentation fault) at any point during the testing process, your score will also be an automatic 0.