Computer Science 301
C Programming in Unix
Siena University
Fall A 2025

Assessment 3: A Ratio Summer

Due: 9:00 AM, Wednesday, October 1, 2025

In this assessment, you will work with structures and pointers to implement a simple calculator for ratio values.

You may work alone or in groups of size 2 or 3 on this assessment. However, in order to make sure you learn the material and are well-prepared for the exams, those who work in a group should either collaborate closely while completing the problems or work through the problems individually then discuss them within your group to agree on a solution. In particular, the "you do these and I'll do these" approach is sure to leave you unprepared for upcoming tasks and the exams and is prohibited.

This assessment will be graded as a programming assignment. Please review the syllabus description of the expectations for a programming assignment as opposed to a practice program.

AI assistance is not permitted on this assignment, as the purpose of this assignment is to practice and to gain a deeper understanding of the intricacies of memory management when working with structures in C.

Learning goals:

1. To work with structures reqiuring explicit and careful memory management

---

Getting Set Up

In Canvas, you will find a link to follow to set up your GitHub repository, which will be named `ratiosum-assessmentt-yourgitname`, for this assessment. Only one member of the group should follow the link to set up the repository on GitHub, then others should request a link to be granted write access.

All GitHub repositories must be created with all group members having write access and all group member names specified in the `README.md` file by 11:59 PM, Friday, September 26, 2025. This applies to those who choose to work alone as well!

Development Process Expectations and Reporting

Important: in order to encourage a step-by-step problem-solving approach and good software development practices, you must have no fewer than 5 commits of partially-working versions of this program, with good commit messages (e.g., "reading input file correctly", "first working version of the method to add ratios", etc.). You will likely have more than 5 total commits.

Please add the 5 commits you wish to have considered for this requirement to the table that you will find in your repository's `README.md` file. To find the information to include in this table, go to you repository on GitHub. Then go to its commits page. At the end of each commit listed there, you will see an icon that looks like this: <>. This is a link to the GitHub repository as it existed after that commit. Copy that link and the corresponding commit message into the table in the `README.md` file.

A penalty up to 50% will be applied if these commits are not listed in the `README.md` and/or do not demonstrate an appropriate development process.

The Ratio Summer

Write a program with a `main` function in `sum_ratios.c` that reads in a series of lines representing ratios from an input file and prints the sum of those ratios in lowest terms at the end.

The program should take a single command-line parameter which is the name of the file that contains the list of ratios, one per line.

Properly formatted lines in the file should look like this, where `n` and `d` are `int` values:

```
   n/d
```

Note that with `scanf` and `fscanf`, you can "consume" unwanted characters and not use them in your results. For example, if you know the input is going to be 3 comma-separated numbers, such as

```
23,7,21
```

there is no need to read the commas into variables. You can read all three numbers and consume the commas with a single `scanf`:

```
  int matched, first, second, third;
  matched = scanf("%d,%d,%d", &first, &second, &third);
```

which should place the 23 in `first`, 7 in `second`, 21 in `third`, and a 3 in `matched`. Your input won't have commas, but will have that slash, and this technique will simplify your input. Numerators and denominators may be any nonnegative integer. If 0 is given as a denominator value or a negative value is given for either the numerator or denomonator, a warning message should be printed and that input line should be otherwise ignored.

Your program should stop reading input and print the final result when it encounters an incorrectly formatted line or the end of the file. (Reminder: `fscanf`'s return value, like `scanf`'s before it, is your friend.)

You do not need to store, nor should you store, a collection of all of the ratios. The only ones you need are the current one and the running sum. You should use the `ratio.c`, `ratio.h`, `gcd.c`, and `gcd.h` files, unmodified, from the earlier lab. They are provided in your repository. For each ratio you read in, you will call `create_ratio` to obtain a structure to store it, and to add two ratios, you must call `add_ratios`.

Provide your own `sum_ratios.c` and a working `Makefile` that will build your program on noreaster (and hopefully any other system on which you wish to run it).

Be sure to perform appropriate error checking including meaningful error reporting, and free all memory your program allocates. The challenge here is in managing the memory properly.

Your program should compile with no warnings. Even if you develop somewhere else, be sure to compile and run on noreaster before you finalize your submission to ensure there are no warnings generated.

Memory Diagram

Draw a memory diagram for your final program showing the state of all stack and heap memory in the following situation. Your program has read in the input

```
1/2
1/4
1/8
```

and has called `add_ratios` to obtain the latest running sum and return it in a new `ratio`. Your diagram should show the state of memory just before the `return r;` on the last line of the function. Commit and push a PDF of a document containing your diagram as `diagram.pdf`.

Submission

Commit and push (and document your 5 commits)!

Grading

This assignment will be graded out of 65 points.

Feature	Value	Score
input file processing	6	
warnings for 0, negative	3	
correct sum of inputs	15	
proper use of provided functions	6	
no unnecessary memory allocated persistently	7	
all memory is freed	7	
documentation and style	10	
`Makefile`	1	
memory diagram	10	
Total	65	