---

title: Homework 1
assigned: 2025-08-27T23:59:00-05:00
due: 2025-09-04T23:59:00-05:00
description: Practice reading, writing, and plotting data, running code with command line arguments.
summary: Exercises with input/output.
author: Gordon Erlebacher
TA: Kunal Watasabe

Homework 1

You are allowed to use AI, but cannot copy from each other. I recommend using GPT5.0, GPT 4.0 or Gemini 2.5 Pro (available to all students). Feel free to read papers or process them through NotebookLM and converse with them. Use VSCode and github.

- Search the internet for at least two files with one or multiple columns of 1-D numerical data. Examples might be temperature, velocity, weight, etc. The should be at least 1000 rows in the file. Name this file.
- For each data file, create an input file with code parameters. Code parameters might include:
  - the number of lines to read from the file
  - the columns to read from the file
  - the name of the file that contains the data.
  - any other parameter the code depends on.
- Add at least one command line argument to the code: the name of the file that contains the code parameters.
- The code should do the following:
  - read the data from the data file
  - compute the data mean and standard deviation of the data (each returns a scalar)

    - write out to a file the number of parameters read in, the mean and the standard deviation.
    - write out the data read in after linear transformation to the range [0, 1].
    - execute the code once for each data file.
- Write a python program to read the transformed data and plot it with matplotlib.
  The plot contains a number of curves equal to the number of data files you processed.
- Add a small description of every function you create in your code.

Constraints

- Use std::functions for input and output
- Use std::iomanip to print floating point numbers with two decimal places
- Store program parameters in an input file
- Read the input file using command-line arguments.

Includes to use

- `<iostream>`
- `<iomanip>`
- `<vector>`

Answer the following questions

Include the answers to these questions in a homework report named “REPORT.md”, stored in your repository.

Here are 10 quiz-style questions tightly aligned to your Homework 1 requirements. Students should be able to answer these without tools, based on what they implemented.

1. **Command-line I/O**

   Explain how your program uses `argc` / `argv` to accept the name of the input parameter file. What user error cases did you guard against (e.g., missing filename, unreadable file), and how?

2. **Parameter file design**

   Describe the format of your parameter file: what keys/values it contains, and how your code parses them. Why is a parameter file preferable to hard-coding the values for this assignment?

3. **Data ingestion + brief description**

   You were asked to print a 1–2 line description of the data you read. What did you choose as your 1-D

dataset (not temperature), and why is it “more interesting than temperature” in this context? What exactly do those 1–2 lines communicate to the reader?

4. **Statistics: mean/variance/std-dev**

   Give the formulas you used for mean, variance, and standard deviation. If you implemented an online or two-pass method, explain why (numerical stability, clarity, etc.). What are typical pitfalls with single-pass variance?

5. **Output formatting with `<iomanip>`**

   Show (from memory) how to print a floating-point value to two decimal places using `<iomanip>`. What is the difference between `std::fixed` and `std::scientific`, and how do they interact with `std::setprecision`?

6. **Data structures: `std::vector`**

   Why is `std::vector<double>` an appropriate container for your 1-D data? Contrast it with a raw array in terms of safety and usability. What is the amortized complexity of `push_back`, and when might `reserve` be helpful?

7. **Timing the code**

   Which C++ facilities did you use to time your program (e.g., `std::chrono::steady_clock`)? What exactly are you timing (I/O only, compute only, end-to-end), and how did you report the timing results?

8. **Batch runs over a range of parameters**

   You ran the code with multiple parameter sets / input files. How did you automate these runs (e.g., a loop in the program vs. a Makefile phony target vs. a shell script)? What naming convention did you use for outputs so that runs don’t overwrite each other?

9. **Build system and IDE**

   Explain how your `Makefile` is organized: list the main targets (e.g., `all`, the executable name, `clean`, possibly `run`) and what each does. In your IDE (VS Code or Cursor), how do you configure the build task so it invokes `make`, and where do you pass program arguments at run time?

10. **Plotting pipeline and artifacts**

    You were asked to write 3–5 output files and plot 3–5 curves with a Python script that includes labeled axes, a title, and a legend. Describe the exact file format you produced, how your Python code reads those files, and how you verified that each curve in the legend corresponds to the correct run/parameter setting.

These questions track the specific Homework 1 requirements on arguments, parameter files, timing, statistics, plotting, and required headers.

Practice questions for the upcoming quiz (not graded) (Answer key below)

1. Which `main` signature is best for a program that must read the input parameter filename from the command line?

```
a) `int main()`
b) `int main(int argc)`
c) `int main(char** argv)`
d) `int main(int argc, char* argv[])`
e) `void main(int argc, char* argv[])`
```

---

1. If the program requires exactly one argument (a path to a parameter file), which check is correct?

```
a) `if (argc < 1) { /* error */ }`
b) `if (argc != 1) { /* error */ }`
c) `if (argc != 2) { /* error */ }`
d) `if (!argv) { /* error */ }`
e) `if (sizeof(argv) < 2) { /* error */ }`
```

---

1. Given:

```
std::ifstream in(fname);
if (!in) {
    // ...
}
```

What does `!in` correctly detect?

```
a) The stream has unread characters
b) The file is empty
c) The file failed to open / stream is in a bad state
d) The file has non-ASCII characters
e) End-of-file was reached
```

1. Which header must be included to use `std::setprecision`, `std::fixed`, and `std::scientific`?

```
a) `<iostream>`
b) `<iomanip>`
c) `<ios>`
d) `<format>`
e) `<cstdio>`
```

1. To print a floating-point value with exactly two digits after the decimal point, which sequence is correct? For example, 25.35253 should print out as 25.35\;.

```
a) `std::cout << std::setprecision(2) << x;`
b) `std::cout << std::fixed << std::setprecision(2) << x;`
c) `std::cout << std::scientific << std::setprecision(2) << x;`
d) `std::cout << std::hexfloat << std::setprecision(2) << x;`
e) `std::cout << std::precision(2) << x;`
```

1. What is the effect of `std::endl` compared to `'\n'`?

```
a) No difference
b) `std::endl` inserts `'\n'` and flushes the stream
c) `std::endl` inserts `'\r\n'` on all platforms
d) `std::endl` disables buffering
e) `std::endl` prints the current line number
```

1. Consider reading doubles from a data file:

```
std::vector<double> v;
double x;
while (/* A */) {
    v.push_back(x);
}
```

Which loop condition is correct and avoids using EOF as a sentinel?

```
a) `!in.eof()` after `in >> x` inside the loop
b) `while (in) { in >> x; v.push_back(x); }`
c) `while (in.good()) { in >> x; v.push_back(x); }`
d) `while (in >> x) { v.push_back(x); }`
e) `while (true) { in >> x; v.push_back(x); }`
```

1. Given `std::vector<double> v;`, which statement is true?

```
a) `v.size()` equals allocated capacity
b) `v.capacity()` is the number of initialized elements
c) `v.reserve(n)` increases `size()` to `n`
d) `v.resize(n)` changes the number of elements returned by `size()`
e) `v.shrink_to_fit()` is guaranteed to reduce capacity
```

1. To reduce reallocations when reading N values of unknown size but with an upper bound estimate, which is best?

```
a) Use `v.shrink_to_fit(N)` before reading
b) Use `v.reserve(N)` before reading
c) Use `v.resize(N)` before reading and never change it
d) Use `v.max_size(N)` before reading
e) Avoid vectors; use raw arrays
```

1. Pick the best function signature for a mean function that does not modify its input:

```
a) `double mean(std::vector<double> v)`
b) `double mean(const std::vector<double>& v)`
c) `double mean(std::vector<double>& v)`
d) `double mean(std::vector<double>* v)`
e) `void mean(const std::vector<double>& v, double& out)`
```

1. Which formula is the unbiased sample variance (Bessel's correction)?

```
a) `Σ(xᵢ − μ)² / n`
b) `Σ(xᵢ − μ)² / (n − 1)`
c) `Σxᵢ² / n − μ²`
d) `Σxᵢ / (n − 1)`
e) `Σ|xᵢ − μ| / n`
```

1. Why is a two-pass variance computation often preferred over a naïve one-pass approach?

```
a) It is always faster
b) It avoids dynamic memory
c) It is more numerically stable for large or ill-scaled data
d) It produces the population variance automatically
e) It avoids division altogether
```

1. With `<numeric>`, which call computes the sum of all elements in `v` as a `double`?

a) `std::accumulate(v.begin(), v.end(), 0)`
b) `std::accumulate(v.begin(), v.end(), 0.0)`
c) `std::reduce(v.begin(), v.end(), 0)` (C++11)
d) `std::sum(v.begin(), v.end())`
e) `std::accumulate(v, 0.0)`

1. Which is the most appropriate clock for measuring elapsed wall-time of your program?

a) `std::chrono::system_clock`
b) `std::chrono::steady_clock`
c) `std::chrono::file_clock`
d) `std::chrono::utc_clock`
e) `std::chrono::process_cpu_clock`

1. Fill in the blanks to measure milliseconds:

```
auto t0 = std::chrono::steady_clock::now();
// ... work ...
auto t1 = std::chrono::steady_clock::now();
auto ms = std::chrono::duration_cast</* A */>(t1 - t0).count();
```

What should replace `/* A */`?

a) `std::chrono::milliseconds`
b) `std::chrono::seconds`
c) `std::chrono::nanoseconds`
d) `std::chrono::duration<double>`
e) `std::chrono::time_point<std::chrono::steady_clock>`

1. Which include set is sufficient for reading doubles from a file into a vector and printing with fixed precision?

a) `<iostream>, <vector>`
b) `<iostream>, <fstream>, <vector>`
c) `<iostream>, <iomanip>, <vector>`
d) `<iostream>, <fstream>, <iomanip>, <vector>`
e) `<cstdio>, <vector>`

1. Given:

```
std::ofstream out("stats.txt");
out << std::fixed << std::setprecision(2) << mean << '\n';
```

Which statement is true about resource management?

```
a) The file must be manually closed with `fclose`
b) The file will close automatically when `out` goes out of scope
c) The file closes only after calling `std::flush`
d) The OS never buffers file output in C++
e) You must call `out.sync()` to write any data
```

1. Which loop best prints all values in `v`, one per line?

```
a) `for (double x : v) std::cout << x << std::endl;`
b) `for (auto& x : v) std::cout << x << '\n';`
c) `for (auto x : v) std::cout << x << '\n';`
d) All of a), b), and c) are acceptable for printing values
e) Only a) is correct in standard C++
```

1. Suppose you parse a simple parameter file `alpha=0.5` using streams. Which approach is most idiomatic?

```
a) `double alpha; std::cin >> "alpha=" >> alpha;`
b) `double alpha; std::string k, eq; in >> k >> eq >> alpha;` expecting `alpha = 0.5`
c) Read each line as `std::string line`, then split on `'='` to get key/value
d) Use `scanf("alpha=%lf", &alpha);`
e) `std::getline(in, alpha);`
```

1. Which statement about `std::vector<double>` is correct?

```
a) `push_back` is O(n) amortized
b) `push_back` is O(1) amortized
c) Accessing `v[i]` checks bounds and throws on error
d) `v.at(i)` is undefined behavior on out-of-range
e) `v.data()` returns a copy of the underlying array
```

Answer key (for instructor)

1. d ok
2. c ok
3. c
4. b
5. b
6. b
7. d
8. d
9. b
10. b
11. b
12. c
13. b
14. b
15. a
16. d
17. b
18. d
19. c
20. b