# COMP 1633: Intro to CS II

# File I/O + Arguments

Charlotte Curtis

February 14, 2024

# Where we left off

- Grouping data with structures
- Functions + structures
- Arrays + structures

> *Textbook Chapter 10*

```
Struct Point {
   double x;
   double y;
}

Point p1 = {1.0, 2.0};
Point p2 = {3.0, 4.0};
```

# Today's topics

- Assignment 1 feedback
- File I/O: reading and writing files
- Command line arguments

# Assignment 1

- Generally very well done!
- Common issues:
  - Reading user input in `main` instead of `get_range`
  - Casting the result of `total_afflicted / N_TRIALS` to a `double` loses precision
  - The most common and insidious issue: floating point accumulation error

```
current_p = p_start + i * p_step;
```

vs

```
current_p += p_step;
```

# Assignment 2

- Due the Friday *after* reading week
- You should have all the necessary tools to complete it after today!
- File reading/writing, command line arguments, structures, arrays, functions... all kinds of fun stuff
- Make sure to use C-strings and primitive arrays, *not* `std::string` or `std::vector`

> *After this assignment you can use `std::string` for the next one*

# File I/O

- So far we've been abusing `cin` to read content from files
- Input redirection is pretty limited
  - Only one file at a time
  - Can't do user input and file input
- Solution: file streams!

```
#include <fstream>
... // int main, using namespace std, etc
ifstream in_file; // input file stream object
ofstream out_file; // output file stream object
in_file.open("some_input.txt"); // open the input for reading
out_file.open("some_output.txt"); // open the output for writing
```

# Reading from a file

## Python

```python
in_file = open("input.txt", "r")
out_file = open("output.txt", "w")

for line in in_file:
    out_file.write(line)

in_file.close()
out_file.close()
```

## C++

```cpp
ifstream in_file("input.txt");
ofstream out_file("output.txt");
char line[256];

while (in_file.getline(line, 256)) {
    out_file << line << endl;
}
in_file.close();
out_file.close();
```

- 3 steps: Open the file stream, do stuff with it, then close it
- Syntax to read/write is the same as `cin` / `cout`

# Reading from a file

- Reading from a file is exactly the same as reading from `cin`

```
ifstream in_file("time.txt");
char colon;
int hours, minutes;
in_file >> hours >> colon >> minutes;
```

- Writing to a file is exactly the same as writing to `cout`

```
ofstream out_file("time.txt");
out_file << hours << ":" << minutes;
```

- You can also use `getline`, format specifiers, etc
- In fact, `cin` and `ifstream` both **inherit** from `istream`!

# Side Tangent: `while (getline)`

- You could use `eof()`, but...

```
while (!in_file.eof()) {
    in_file.getline(line, 256);
    // do stuff with line
}
```

- Remember `eof` only triggers *after* you've tried to read past the end of file! The loop above is almost certainly a **logic error** as the last line will be processed twice
- `getline` returns a **reference to the stream object** (`basic_istream &`)
- If the read failed for whatever reason, `NULL` is returned and the condition evaluates as `false`

# The fail bit

Consider the following code snippet:

```
ifstream in_file("input.txt"); // can also declare and then open separately
// do stuff with in_file
in_file.close();
```

- What happens if the file doesn't exist?
- The `fail` bit is set when a read fails - this can be checked with `fail()`

  ```
  if (in_file.fail()) {
      cout << "Something went wrong!" << endl;
  }
  ```

- Alternatively, you can use `if (!in_file)` as a shorthand

# Command line arguments

- Typing in a file name is annoying because you can't use tab completion
- Other programs like `emacs` take arguments, so why can't we?

```
$ cd some_dir # cd is the command, some_dir is the argument
$ ./a.out input.txt # ./a.out is the command, input.txt is the argument
```

- This is pretty straightforward: add parameters to `main` as follows:

```
int main(int argc, char *argv[]) {
    // argc is the number of arguments
    // argv is an array of strings containing the arguments
}
```

# Command line arguments

- Command line arguments are mapped **1 to 1** from the command line to `main`

```
$ ./a.out hello world
```

```
int main(int argc, char *argv[]) {
    for (int i = 0; i < argc; i++) {
        cout << argv[i] << endl;
    }
}
```

- This means that `argv[0]` is the name of the program
- `argc` is the number of arguments, *including* the program name

# File I/O check-in 1/2

What is this code snippet doing? Assume the appropriate `#include` directives are present - it compiles and runs.

A. Nothing, it's just reading from a file

B. Displaying the contents of a file

C. Copying the contents of a file

D. Removing newlines from a file

```
ifstream in("input.txt");
ofstream out("output.txt");
const int BUFSIZE = 256;
char line[BUFSIZE];
while (in.getline(line, BUFSIZE)) {
    out << line << endl;
}
in.close();
out.close();
```

# File I/O check-in 2/2

What is forgotten in the following code snippet? Assume the appropriate `#include` directives are present - it compiles and runs.

A. Nothing, it's perfect

B. The file isn't closed

C. The file isn't used for anything

D. Should use `getline` instead of `>>`

E. The file is opened for writing, not reading

```
ifstream in("input.txt");
int x;
while (cin >> x) {
    cout << x << endl;
}
in.close();
```

# Command line plus file I/O example

Write a program that takes two command line arguments, an input file and an output file, and copies the contents of the input file to the output file. It should also replace any instance of the word "now" with "meow".

If the user does not provide two arguments, or if the files can't be opened, the program should print an error message and exit.

# Preview of pointers

- Declaring a variable reserves a chunk of memory at some **address**
- A **pointer** is a variable that stores the **address** of another variable
- We've (kind of) been using pointers already!
  - `void foo(int arr[], int size)`
  - `void bar(int &x)`
- Passing a variable by reference is actually passing a pointer to the variable
- Let's check out Python Tutor

# Coming up next

- Tomorrow's lab: create a C++ project from scratch, plus file I/O and command line arguments
- Reading week!!!
- After that: Pointers - aka where C++ gets *really* fun

> *Textbook Chapter 10*