

CSC 211 Assignment 3

The Unix `cat` command

Due Wednesday, February 21st by 11PM

Background

One of the many command-line utilities available on Linux (and other Unix-like systems) is `cat`, which is short for **concatenate**. In its simplest form, `cat` can display the contents of a text file; it can also be used to concatenate together many text files.

For this assignment, you are going to write your own version of `cat`.

Before you proceed, you should first play with `cat` on the command line. Get into your Docker environment (or alternative, for those of you using Virtualbox or CodeAnywhere or another solution).

- Type `cat /etc/hosts` and notice that the contents of the file `/etc/hosts` are printed out.
- Now, type `cat /etc/hosts /etc/hosts` and see if you can make sense of the output.
- Now, try typing `cat /etc/hosts /etc/hosts > test.txt`
 - Followed by `cat test.txt`.
 - Note that the file `test.txt` contains two copies of the contents of `/etc/hosts`.
 - This is because the `>` redirected the output from `cat` to the new file `test.txt`.

What `cat` does is that it takes one or more filenames as command-line arguments. It prints out the contents of each file in turn. By using the Unix shell feature of **redirection** (the `>` character), we can combine all of those files into one file. Undoubtedly, you can imagine circumstances when this would be useful.

So, your `cat` must do the same. Like the `echo` program you wrote for Lab 2, `cat` takes several command line arguments. But, it must try to open a file with the same name as each command-line argument. If it can't open a file, it must print "could not open file" to the special output stream `STDERR` followed by the name of the offending file.

Academic honesty

Remember, this is a **solo assignment**. You may not show your code to any classmate, or look at any classmate's code.

Getting Started

- Get into your Docker development environment (or appropriate alternative, such as CodeAnywhere.com)
- Download the assignment framework with `git clone https://github.com/csc211/a3`
- You will see this `Readme.md` along with a C++ source file (`cat.cpp`) and a compile script `compile`
- You can now compile the assignment by typing `./compile` and it will produce an executable program called `cat`. You can run `cat` by typing `./cat` However, it won't do anything yet!
- Read the existing comments in `cat.cpp` carefully and be sure you understand them **before doing anything else!**

Requirements

Your program must take one or more command-line arguments, each of which should be a file name. It must print out the contents of each file in the order given by the command-line arguments. If any file specified cannot be opened, the message “cat:” followed by the filename, followed by “: no such file or directory” must be printed to `STDERR`. Note that just as `std::cout << "hi"` prints the message “hi” to `STDOUT`, `std::cerr << "hi"` prints it to `STDERR`. The difference between these different **output streams** will be discussed in class.

If given no command-line arguments, your program should simply exit (your `main()` can issue a `return 0`).

Comments

Your comments should **explain the contract of any function you write**. That is what arguments does a function expect, and what value does it return, and **what is the relationship between the two**.

Comments should not focus on explaining the C++ language to the reader, though at this point, if doing so helps you understand your own code, it's fine. Any code you write that you think is particularly “clever” should be explained (for example, non-obvious corrections for off-by-one errors, or a for loop that starts at an index other than 0).

Hints

We have seen two approaches to read from a file. The first is **character-oriented** input:

```
stream.get() stream.put()

ifstream infile(filename);

char ch;
while (infile.get(ch)) {
    cout << ch;
}
infile.close()
```

The second is **line-oriented input**

```
getline(stream, str)
```

Note that this is a bare function, rather than a method on the `ifstream` class.

```
ifstream infile;
infile.open(filename);
std::string line;
while (getline(infile, line)) {
    std::cout << line << std::endl;
}
infile.close();
```

Testing your own code

One benefit to reimplementing a common program like `cat` is that you can test your program side-by-side with `cat` to make sure the output is the same.

- Type `cat compile`. What do you see?
- Type `cat compile compile`. Now, what do you see?
- Type `cat compile nothing`. Assuming there is no file called `nothing` in your project directory, what's going on?
- Now, if your program is working, running the same commands with your own `cat` program should produce the same output:
- Type `./compile` to compile your program.
- Type `./cat compile` to run your own `cat` program. Does your program behave the same as the built-in `cat` command?
- Try it with other inputs, and make sure it behaves the same. Once it does, you should submit it to Mimir.
- Note that you have limited submissions on Mimir, so you must test **locally** (in Docker or CodeAnywhere or Virtualbox, or whatever alternative you have set up).

Grading Rubric

For this assignment, correctly passing all tests on Gradescope is worth 80% of your grade. The remaining 20% is based on reasonable commenting habits, and the structure and organization of your code.

Submitting

You will submit `cat.cpp` via Mimir, where its functional correctness will be graded automatically. For this assignment, you only get 5 submissions to Mimir, so make sure to **test locally** first.