# Breakout / Lab 12

REPL: Read–Eval–Print Loop

CSCI 1730 – Fall 2015

## Problem / Exercise

A read–eval–print loop (REPL), also known as an interactive top-level or shell, is a simple, interactive computer programming environment that takes single user inputs (i.e. single expressions), evaluates them, and returns the result to the user. For this breakout lab, you need to design a REPL that can print out information about a job pipeline that a user has described as input.

Here are some examples (first line is a prompt plus user input; subsequent lines are output):

- `$ cat file.txt`

```
Job STDIN  = STDIN_FILENO
Job STDOUT = STDOUT_FILENO
Job STDERR = STDERR_FILENO

0 pipe(s)
1 process(es)

Process 0 argv:
0: cat
1: file.txt
```

- `$ cat file.txt | less`

```
Job STDIN  = STDIN_FILENO
Job STDOUT = STDOUT_FILENO
Job STDERR = STDERR_FILENO

1 pipe(s)
2 process(es)

Process 0 argv:
0: cat
1: file.txt

Process 1 argv:
0: less
```

- `$ cat file.txt | grep // > out.txt`

```
Job STDIN  = STDIN_FILENO
Job STDOUT = out.txt (truncate)
Job STDERR = STDERR_FILENO

1 pipe(s)
2 process(es)

Process 0 argv:
0: cat
1: file.txt

Process 1 argv:
0: grep
1: //
```

- `$ cat | grep // | less < in.txt >> out.txt`

```
Job STDIN  = in.txt
Job STDOUT = out.txt (append)
Job STDERR = STDERR_FILENO

2 pipe(s)
3 process(es)

Process 0 argv:
0: cat

Process 1 argv:
0: grep
1: //

Process 2 argv:
0: less
```

- `$ cat file1.txt file2.txt > out.txt e>> log.txt`

```
Job STDIN  = STDIN_FILENO
Job STDOUT = out.txt (truncate)
Job STDERR = log.txt (append)

0 pipe(s)
1 process(es)

Process 0 argv:
0: cat
1: file1.txt
2: file2.txt
```

- `$ cat file1 file2 file3`

```
Job STDIN  = STDIN_FILENO
Job STDOUT = STDOUT_FILENO
Job STDERR = STDERR_FILENO

0 pipe(s)
1 process(es)

Process 0 argv:
0: cat
1: file1
2: file2
3: file3
```

- `$ echo "my \"cool\" shell" | less`

```
Job STDIN  = STDIN_FILENO
Job STDOUT = STDOUT_FILENO
Job STDERR = STDERR_FILENO

1 pipe(s)
2 process(es)

Process 0 argv:
0: echo
1: my "cool" shell

Process 1 argv:
0: less
```

# 1 Group Brainstorm

Breakup into groups based on your seating and brainstorm about how to solve the problem or exercise. Make sure everyone understands the problem, and sketch out potential ways to move towards a solution. Perhaps something that was discussed during lecture might be useful?

# 2 Submit Individual Brainstorm

Login to eLC and submit a version of your group's brainstorm, written in your own words. You may add additional information if you want. You need to write enough in order to convince the grader that you understand the problem or exercise and that you have a plan for moving forward towards a solution. Please include the last names of the other people in your group in your submission. The brainstorm submission should be available on eLC in your assignment dropbox. We prefer that you submit your individual brainstorms before the end of your breakout period, however, you generally have until 3PM on the day of your breakout (as indicated on eLC) to submit them.

**NOTE:** Submissions that do not include an individual brainstorm will not be graded. Also, once you have completed your individual brainstorm, please use the remainder of the lab period to actually work on the lab. Failure to do this may result in the taking of attendance.

# 3 C++ Code

Make sure that all of your files are in a directory called `LastName-FirstName-lab12`, where `LastName` and `FirstName` are replaced with your actual last and first names, respectively. The code for this lab should be written in a file called `chmod.cpp`.

## 3.1 `Makefile` File

You need to include a `Makefile`. Your `Makefile` needs to compile and link separately. Make sure that your `.cpp` files compile to individual `.o` files. The resulting executable should be called `lab12`.

### 3.2 `README` File

Make sure to include a `README` file that includes the following information presented in a reasonably formatted way:

- Your Name and 810/811#
- Instructions on how to compile and run your program.
- A Reflection Section. In a paragraph or two, compare and contrast what you actually did to complete the problem or exercise versus what you wrote in your initial brainstorm. How will this experience impact future planning/brainstorms?

**NOTE:** Try to make sure that each line in your `README` file does not exceed 80 characters. Do not assume line-wrapping. Please manually insert a line break if a line exceeds 80 characters.

### 3.3 Compiler Warnings

Since you should be compiling with both the `-Wall` and `pedantic-error` options, your code is expected to compile without `g++` issuing any warnings. For this lab, compiling without warnings will be one or more of the test cases.

### 3.4 Memory Leaks

You are expected to ensure that your implementation does not result in any memory leaks. We will test for memory leaks using the `valgrind` utility. For this lab, having no memory leaks will be one or more of the test cases.

## 4 Submission

Before the day of the next breakout session, you need to submit your code. You will still be submitting your project via nike. Make sure your work is on `nike.cs.uga.edu` in a directory called `LastName-FirstName-lab12`. From within the parent directory, execute the following command:

```
$ submit LastName-FirstName-lab12 cs1730a
```

It is also a good idea to email a copy to yourself. To do this, simply execute the following command, replacing the email address with your email address:

```
$ tar zcvf LastName-FirstName-lab12.tar.gz LastName-FirstName-lab12
$ mutt -s "lab12" -a LastName-FirstName-lab12.tar.gz -- your@email.com < /dev/null
```