# Interpreter Project, Part 4*

You are not required to submit this assignment. But it is fun!

Your interpreter is sophisticated enough to be able to execute Racket programs of substantial complexity. Aside from a few examples (most notably, the stream examples), your interpreter can evaluate all the code you have written in this course so far. In fact, with just a few more modifications, it's possible to run the interpreter itself in the interpreter. Interpreters that can do this are called "meta-circular" interpreters. Of course, the resulting "meta-interpreter" can do the same thing, yielding a "meta-meta interpreter", which of course can do the same thing, yielding...

Believe it or not, you don't need to write any more code to get meta-circularity to work. However, the code you already wrote might not work as well you thought it did – and you may need to go back and alter a few things. To make things worse, debugging a meta-circular interpreter is quite difficult.

## 1 Testing your meta-circular interpreter

One way to test whether or not your interpreter is meta-circular is to copy and paste all of your code, excluding any lines that begin with a # sign (e.g. "#lang racket"), or begin with (require ...), (trace ...), or (provide ...) from interpreter.rkt onto the INTERPRETER> prompt. (Be sure you press Enter after you paste your code or DrRacket won't even begin evaluating what you've pasted!)

Unfortunately, doing this is EXTREMELY slow – probably due to a bug in DrRacket – and for some strange reason, this takes a while.

The other way to test it, and the way that I will test it after you have handed it in, is to simply include your file into your interpreter. The eval-include function will read any file, evaluating each statement one-by-one, and return void. To help you in debugging problems, the eval-include function prints out the result of evaluating each statement in the file you are including.

I've written all of the code for including files, you just need to add the following line to your i-eval function:

((include? exp) (eval-include exp env))

To help you try it out, I have included a small file in the i-6 directory. First run your interpreter, then at the INTERPRETER> prompt, type (include "test.rkt"). The test program will define two functions which compute factorial: a recursive process solution, factorial-r, and an iterative process solution, factorial-i. When each function is included, it's name will be displayed. Once it is included, try out each function:

(factorial-r 5) (factorial-i 5)

*Thanks to Rich Wicentowski for developing the original version of this project.

## 2 Patching up your meta-circular interpreter

You may found that you have used Racket language features in your interpreter implementation that you have not implemented in your interpreter. For instance, maybe you used map and fold, but did not add them to your interpreter, since they were extra credit questions in Part 3.

You can either patch this by implementing these missing language features, or rewriting your interpreter to avoid them. This process may take some time.

To make my interpreter metacircular, the major changes involved rewriting functions that used letrec to use helper functions instead; rewriting functions that used match; and implementing the define syntactic sugar for functions.

## 3 Metacircularity

Assuming that things worked, it's time to try including your interpreter into itself. At the INTERPRETER> prompt, type (include "interpreter.rkt"). You will see each function name printed out as it is included into your interpreter. It may not work the first time... or the second time... or the third time... or... Getting your interpreter to read in all of its own code can be challenging.

When you get the INTERPRETER> prompt after including the interpreter.rkt file, your interpreter has actually included all of its own code into itself... which means you can now type (repl) and run your interpreter inside of itself...

If, after typing (repl) you get the INTERPRETER> prompt, that prompt is actually from your interpreter running inside of itself!

Now it's time to see if your interpreter running inside of itself works. Many times, you can get to the INTERPRETER> prompt but some simple things (like primitives) don't work. So, try running some simple tests of your interpreter running inside itself. When you are confident it is working, try some more difficult tests.

When you are really sure you think it's working, it's time to try including your interpreter into your interpreter which is already running in your interpreter. Type (include "interpreter.rkt") and press enter. This will now include your interpreter code into your meta-circular interpreter. When it is complete (it should take 2-5 minutes depending on the speed of your machine, with the bulk of the time running the setup-env procedure), and you are presented with the INTERPRETER> prompt, it means you can just run (repl) again, and now you are running your interpreter in your interpreter in your interpreter, in DrRacket. Test it out on something really simple, like (+ 3 4). It will take about one or two minutes to give you an answer.

Finally, if you are daring, and you have a lot of time to kill, you can include your interpreter into itself again. Expect to wait a very long time. I'd recommend going to lunch, taking a walk, and going to dinner, maybe even going to sleep, and waking up the next morning to see if it finished. I started mine running and came back 4 hours later and it was still going... it can be very very slow. However, if you get a new prompt, type (repl) to run the meta-meta-meta-meta-interpreter and expect to wait a very long time for anything to happen. Theoretically you could keep doing this forever, but I doubt you'd want to try – or wait.