Assignment Part 5 – Scripting Languages

# Program Testing Procedure

Run programs and observe output. Each matched file should be output on a separate line. Make sure only files ending in “.conf” are matched (not directories). Unfortunately, cannot test that all such files are found, as the whole filesystem is too big to search in a reasonable amount of time using scripting languages.

# Weekly Question

Perl was the hardest to write program in. This is because I had the least experience with and knowledge of Perl, in addition to Perl being quite tricky to learn/decipher.

In Bash, I effectively already knew how to do the task, so a brief search online and review of the man pages allowed me to write it quite quickly. Finding files is a typical task for a shell language like Bash, so there is a well-established, well-documented, well-designed, single way to do it (find). The only Bash syntax that was required to navigate was invoking a program with arguments, which is trivial.

I had seen snippets of Ruby previously, so the general syntax was not particularly new to me. After an online search for how to find files, I quickly found documentation for a library function that recursively iterates over directories and files (Find.find()). The documentation included a code example, which, thanks to Ruby’s simple, structured syntax, was very readable and easy to figure out. Finally, I knew from previous reading of the provided Ruby guide (and knowing that Ruby is a sane language) that the string class probably has some sort of “does this string end with x?” (String.end\_with?()) function, which I used to check if the file name ends with “.conf”. (I figured that using regexes would be a harder way to perform the task, since Ruby likely does not have any extremely convenient regex functionality like Perl.)

For Perl, however, I had no idea at all what to do – I did not know the syntax nor built-in/library functionality. Searching online for how to find files in Perl yielded several different ways to perform the task, none of which seemed to be particularly simple or intuitive. (I even found two third party libraries, to perform such a simple task!) Additionally, reading the documentation did not help me much, because Perl’s quirky syntax and behaviour made it difficult to understand anything. Eventually, I decided on the “simplest” option (File::Find::find()), which unfortunately could not do the filename matching itself. I had to implement for myself a check to see if the file name ends with “.conf”. I did not bother trying to find functionality that could check if a string ends with another string like in Ruby, because I figured that Perl, having so many features, likely has built-in regex matching so concise (once you learnt the syntax) to use that the advice would be to “just use a regex”. It turns out that I was correct, and I therefore had to examine the provided Perl manual to find out how to perform regex matching. I expected there would be some sort of obscure, built-in syntax or operator for it, and I was correct; there was no way I would have figured it out on my own.

# Reflection

Ruby’s adherence to the regularity and simplicity principles via focus on object orientation increases readability. Most functionality in the program is achieved via object/class method calls, which are largely intuitive and self-documenting. The result is that the code is very readable and easy to understand (even if you did not know Ruby, you could probably figure out what the program does easily).

Perl’s violation of the syntactic consistency principle decreases readability of the program. Many aspects of the syntax are effectively random characters, whereby it is very difficult to decipher what is happening. For example:

* Acquiring a pointer to a function: \&func
* The function parameter list: $\_
* The regex match operator: =~
* The regex syntax: /pattern/ (regex by itself is already a mess of special characters)