Thomas DeMasse
OPL
Due Date: 02/10/2023

Assignment 1: Comparing Languages Report

The description of this assignment was as follows:

In this assignment, you will solve a simple problem in each of five different programming languages:

• Ada
• OCaml or Haskell
• C#, Go, Rust, or Swift
• Python or Ruby
• Prolog

The problem:

“There are 100 doors in a row that are all initially closed. You make 100 passes by the doors. The first time through, visit every door and toggle the door (if the door is closed, open it; if it is open, close it). The second time, only visit every 2nd door (door #2, #4, #6, ...), and toggle it. The third time, visit every 3rd door (door #3, #6, #9, ...), etc., until you only visit the 100th door.” What is left? Doors that are open are perfect squares.

Prolog:

I started by first compiling code off of Rosetta Code to ensure it in fact worked. I found Prolog was challenging at first because when I first started compiling, I kept getting an error message and I couldn’t figure out why. When I ran it in “user mode” - it worked. However, when trying to compile from a working directory it did not. After some troubleshooting, what I noticed was I created a file named “Doors.pl.” Professor Wilkes explained to me (and the class as a whole on discord) that “Prolog has the convention that variables begin with a capital letter, whereas constants, predicate names, function names, and the names for objects must begin with a lowercase letter.” Therefore, when I renamed the file name “doors.pl” it compiled successfully.

After successfully compiling, I examined every piece of code to better understand what was happening and how to run it. Overall I thought Prolog was a cool language to learn but I wouldn’t want to use it in the future. I don’t like how you have to type the parameters when you’re extracting certain information. I thought it ran pretty fast. I thought it was moderately challenging to learn.

Ada:

For Ada I compiled with the command “gnatmake doors.adb.” It created two files “doors.ali” and “doors.o.” Once it was compiled I ran the executable with “./doors.”

It starts by defining a procedure to follow. Next it defines a set of values or a set of operations that can be performed on that value. For this problem it defines whether a door can be open or closed. Then it defines an array of doors from positive values 1 to 100. Then two “for loops” are created which toggle the doors open or closed based on the current state of the doors.   Finally it utilizes another “for loop” to print the results of the final state of the doors. I thought Ada was easy to compile. It’s once simple command. Getting the logic correct in order for it to compile is a different story. It was moderately challenging to learn.

Python:

Python was our own original code. Python used similar logic as Ada. First it defines what two states a door can be (either true or false or open or closed). Then the main function is declared. It starts by initializing an array of 100 doors that all start “closed. Then it states if the door if the door is open close it, if it’s closed then open it. The logic is the same as the previous languages. Synthetically, it’s a bit different. However, all of the same way of thinking is still there. Python was the easiest of the languages to get to work. Combined with the makefile makes it even easier. I would enjoy using python in the future.

Ocaml:

To compile the command is “ocmaml filename.” First, we declare a max constant of doors. In this case the max number of doors is 100. A new function show_doors is then defined. This new function show_doors takes an array of Boolean values and prints the status of each door and with the use of Array.iteri - this function can iterate over the array. The next function is called flip_doors. This function called flip_doors takes the array of Booleans and returns a new array where the state of each door is toggled based on it’s index at the current position “i.’ Next is an if statement that makes sure that the function keeps calling itself (recursion) only if idx (being the index of the door in the array) is less than max_doos (100). Recursion will stop once al the doors have been gone through. Basically, it’s checking each door and if it’s “open” the door will “close. And if the door’s current state is “closed” it will “open.” Finally, the output is printed. It compiled fast. It used the same logic. I wouldn’t mind using Ocaml in the future.

<u>Rust:</u>

Rust was the slowest of all the languages to compile. The logic was the same for Rust compared to the other languages we studied. However, even though it compiled slowly, it was the easiest to understand. It compiles using the command “rustc fileName.” It first starts by declaring a main function. Then a mutable array of 100 boolean values is created. A while loop nested in a for loop then checks while there’s less than or equal to 100 doors – if the door is open then close it. If the door is closed then open it. A final for loop is created to print the results. It iterates through the 100 passes while enumerating the correct state of the door.