Sudoku Reflection Document

CIS*3190 Assignment 2 - March 4, 2016

Geofferson Camp 0658817

The flow of my sudoku program is as follows:

1. Acquire input and output file paths/names.
2. Read input file and construct 2D array representation of the sudoku puzzle.
3. Solve puzzle.
4. Print results.

Building the array was done with two loops. One for reading each line of the file and one for each number of any given line. It was a fairly straightforward process and I used both a while loop and a for loop for more exposure to Ada. I got the algorithm to solve the puzzle from: http://codereview.stackexchange.com/questions/37430/sudoku-solver-in-c. The solution is a brute force recursive method. The algorithm cycles through each location on the board and iterates the location values until it find one that satisfies the game requirements; backtracking when required. To print the puzzle to both screen and file, one procedure sends duplicate text to both destinations.

The most challenging part of designing a sudoku algorithm in Ada was reading-in the game file, specifically handling varying lengths of strings. Once I figured out how it worked it was not difficult to implement but the required steps were not intuitive. The differences in string handling compared to a language like C seem logical given the language is focused on safety and security. It makes sense that you might want to ensure variable characteristics such as length and type before using the data to perform “mission critical” actions. Another drawback I noticed was the requirement of organising all support functions above the main body of the program, if you aren’t going to split your code into multiple files. This is more an issue relating to style than functionality, as (in my opinion) it can be distracting to have to scroll through all the support function before reaching the main code. It is also possible that this is a rather niche problem that only affects programs of a similar size to my sudoku solution as any program with a little more code should probably be split into multiple files anyway. The final issue I had with Ada was the array numbering. When I initially noticed the range could be set to the programmer’s preference and the default was 1, I thought it would be convenient. As I started programming I realised I am more comfortable starting the array index from 0. Additionally, the algorithm I used to solve the puzzle was originally written in C and the difference between index starting points caused a math issue when converting between

C and Ada. At that point, the entirety of my program had been written with respect to a starting index of 1 and changing the initial value to start at 0 was not seriously considered.

Implementing recursion was fairly simple and straightforward. The process does not differ significantly, from what I am used to. The one issue I did have was the need to switch between a function and a procedure. I had initially set the recursive sub-program to be a function to emphasise the returned value. However, it became apparent a procedure call would be better as it allowed me to pass the “board” 2D array by reference and change its contents in each recurrence function call. The drawback was having to handle the return value in a less “confident” manner. I was able to compensate by spending more time on the return handling ensuring it was correct, but I would try to avoid such an approach on a larger project.

While I mentioned previously I was not a fan of how programmers are forced to organize their subprograms above the main code, I do like that the same restriction is applied to variable declarations. I find declaring and initializing variables in C to be awkward, I am always left unsatisfied with how they are organised. I did not have the same feeling when they had their own “home” at the top of the program. I can also see how this would be helpful in a “mission critical” programs where organization and coding style are of utmost important when trying to avoid bugs.

I think it would have been easier to write the program in C, but only because of familiarity. I do not think the features required by this program were extensive enough to get the full Ada experience; both with respect to its limitations and unique abilities. There were some minor syntax differences that hindered me for some time but, ultimately, the general strategy I used in Ada is very similar to the one I would have used in C.

If I were to rewrite the program, the only thing I would change would be the algorithm used to solve the puzzle. The brute force method is very inefficient, where an insolvable puzzles has O(n^n). After learning Ada, building this program would not be difficult and I do not think it is intensive enough that redoing any other part of the program would provide a significant advantage, other than the puzzle algorithm. Given my knowledge of programming Ada was fairly simple to learn. After getting over the string “hump” it was very similar to other languages such as fortran and C. The “in” and “out” feature of Ada procedures make the language more usable. If I were to build another program of similar requirements and size, I would choose Ada because of how much easier it is to pass values by reference. The ability to easily change between “in”, “out”, and “in out” also made making changes to the program easy. On more than one occasion I had to reevaluate how how I was approaching the recursion sub-program and the “in out” options made the required changes easy to implement as I had to redo a much smaller portion of the code compared to if I was working in C.

In short, I enjoyed working in Ada. It was nice to learn something new, especially where I could understand why its differences existed and how they might be beneficial over methods I am more familiar with. The struggles I had learing Ada originated in unfamiliarity and I don’t think the language itself can be blamed. While strings are difficult to work with in Ada, I can understand why the designers implemented them how they are, this makes working with them less frustrating.