CIS*3190 Fortran Maze Reflection

Geofferson Camp - 0658817

The execution flow of my program is as follows:

1. The user is asked to enter a maze filename.
2. The program reads from the file and stores the maze in a 2-dimensional array.
3. One stack data structure is initialized using the dimensions of the maze array.
4. The maze traversal algorithm uses the stack functions to traverse the maze, marking the path in the maze array, until the exit is found.
5. The maze array is printed to standard out.
6. The stack is emptied.

In order to solve the maze traversal problem, I decided it would be best to initially code the input algorithm, proceed to set up a stack substitute within the main program, program the main maze algorithm, and then finalize the stack data structure inside a module. I took this approach as I thought it would be the best way to create the program alongside learning Fortran. I reasoned that the most efficient way to become familiar with the syntax was to code a part of the program that did not require much logical thought, but included many different aspects of the syntax. This worked out well.
I ended up becoming proficient working with variables, I/O, arrays, dynamic memory, and loops all before having to worry about anything going wrong logically. I was initially hesitant to set up the data structure in its final form as I was not sure how the module feature of Fortran worked. By taking this approach, I was able to make good use of the syntax I had just learned and implement the logic portion of the assignment without having to worry about if the module was working correctly. After I had the program functioning correctly, I was feeling confident and felt comfortable taking my time researching and learning about modules. This made the transition process easy and I had no issues implementing the module data structure.

Completing the file I/O portion of the program was the most tedious part of the assignment. I decided the best way to pick up the language was through experimenting, so getting the file I\O to work correctly involved a lot of trial and error. The algorithm I ended up with involved reading in each line as a list and then looping though that buffer to assign each character to its respective row and column in the maze array. I experimented with the ability to format the “read” input but ultimately decided to take the nested iterative approach as I understood it thoroughly and was struggling to find helpful documentation on how to use the formatting features.

The other components of the program were straight-forward to design. I chose to implement a depth first stack approach as I was able to easily follow the logic of the sample algorithm provided in the project description and could not think of any major benefits to using the queue alternative. One characteristic of the maze grid that gave me some trouble was the orientation of the 2D array. I had originally tried to increment the active cell north by adding to its y value and south by subtracting. My initial assumption may have been a result of the opposite orientation in a previously used language. The flow of the maze algorithm working with the stack functions can be modeled with the following steps, which are repeatedly carried out while the stack is not empty:

1. Read the top of the stack.
2. Remove from the top of the stack.
3. Check if the read value is the exit.
4. If yes, print completion message and break out of loop.
5. If no, add the east, west, north, and south cells to the stack.

While designing the operation functions for the stack I decided to make the “pop” and “top” functions independent to make the data structure more reuse friendly.

The greatest challenges faced while designing this algorithm were related to file I/O and the design of the stack data structure. The file input design was difficult not only because of the syntax learning curve, but because of how line reading behaves in Fortran; opposed to the modern languages I am familiar with. Perfecting the input algorithm required a significant amount of time due to characteristics of the language such as the difference between character and string arrays as well as the “settings” required to specify how the read function behaves. The issues I had designing the stack structure were rooted in how arrays work in Fortran. Fortran seems to provide better support for multi dimensional arrays compared to a language similar to C; but, because I was mostly unfamiliar with how to take advantage of those strengths, after attempting to implement a 2D array in Fortran style, I resorted to using two 1D arrays. My resulting implementation is a good example of how different Fortran is from other languages less friendly to data analysis, as familiarity won out over functionality.

The module feature of Fortran made creating the stack data structure straight-forward. When I started programming I wasn't completely sure how it would be done but as I became more familiar with modules it became obvious how the structure would be set up. The specification section of the module was well suited to store the stack details and the "contains" section of the module worked well as a location to store the stack operation functions.

The module in general is an interesting feature of fortran. It is evident that the module is Fortrans version of what a class is in object oriented programming. The differences, for example, a section dedicated to holding data, are good examples of Fortrans strength as a scientific and data analysis tool. Further thought on the subject reveals even more about how the language evolved to serve a different purpose than object oriented languages. The lack of features such as the ability to extend a class or inherit features of a parent class would make it difficult to produce a flexible fortran program in comparison to an OOP equivalent; more proof that Fortran is a job specific language.

I enjoyed programming in Fortran because it feels simple and bare-bone, similar to C, but also seemed more refined in supporting specific actions. Examples of this are its array handling and dynamic memory allocation. In C, it can often be confusing how to properly initiate and use dynamic memory but the same actions in Fortran did not require a second thought. In regards to array handling, the most obvious example is the ability to set arbitrary index ranges. This is convenient and makes working with the code more clear when needing to reference specific values that don't necessarily start at 0.

I do not think it would have been easier to write this program in a language such as C. As Fortran has been around and growing for a significant amount of time, and this program was not very intensive, I believe that the downsides of using Fortran to create a program for purposes other than data analysis went largely unnoticed. On the other hand, I was able to take advantage of some helpful features, which I mentioned above regarding memory and finely tuned features. In short, anything I needed to do in C I was also able to accomplish in Fortran and often in a more organised and concise manner.

One aspect of my program that I think could have been improved on was the splitting of the stack arrays. They should have been merged into a 2D array capable of storing both a full set of coordinates. Within the current set up, the two arrays which make up the stack are at risk of becoming unsynced. Properly making use of a 2D array would have avoided this issue.

Given my knowledge regarding programing, Fortran was fairly simple to learn. The differences I experienced between C and Fortran were not difficult to understand. The syntax of the language, as well as how it is organized is logical, intuitive, and easy to pick up. How the language operates and is structured is similar enough to C that the same pseudo code could be used to create the same program in both languages; for the case of the maze algorithm at the very least.

In summary I enjoyed learning and working with Fortran. I probably won’t be using it to build highly functional software as I am sure its restrictions will become more evident, but if I ever need to perform data analysis I will be sure to give it a second thought.