CSCI3180 Assignment 1 Report

Name: Ling Leong Student ID: 1155062557 Email Address: alanalan0124@yahoo.com.hk

1. Conveniences and difficulties
    a. Reading file in certain format
    For COBOL, it is good at reading file format that is table like, but for the irregular file format used in game of life, we have to read the whole line into a string and then parse it afterwards. For FORTRAN, it is relatively easier as there is a proper way to define the read format so that we do not have to parse it manually.
    b. Producing next generation
    This part is relatively easy as both languages do have 2D arrays. Simulation and producing next generation is just like other languages such as C, by using multiple nested for loop. The problem is that we are not allowed to use for loop so we need to implement a pseudo for loop ourselves, by using if and goto. It makes the code look really ugly, especially in FORTRAN.
    c. Case control
    This is the more in inconvenient part mainly due to the assignment do not allow us to use control statement except IF statement. We have to use if and goto to create some pseudo code block in FORTRAN. In COBOL, we are allowed to use IF THEN END-IF so it is a bit more convenient. However, more of these inconveniences are due to the assignment limitation.
    d. File output and formatting
    This is a trickier part to handle as we have to use an old version of both languages than do not have must string processing function, so we have to implement some helper functions to help us format the output properly. Dealing with '\r' in UNIX system also take some times.
2. Compare FORTRAN and COBOL with Ruby
    a. Paradigm
    FORTRAN is an imperative programming language. COBOL is also an imperative programming language. Ruby is a multi-paradigm language that support functional, object-oriented, and imperative programming paradigms
    b. Data type
    In Ruby, everything is an object that has a class, including all data types, such String, Fixnum, Integer, Numeric, Float, Numeric, Nil, Hash, Symbol, Array and Range. For FORTRAN and COBOL, they are not object-oriented so their data types are not objects. FORTRAN only have a few data types, including CHARACTER, COMPLEX, DOUBLE PRECISION, INTEGER, LOGICAL, and REAL. COBOL also have small number of data types which includes Numeric, Alphabet, Alpha-numeric, Sign, and Assumed decimal point
    c. Parameter passing
    In FORTRAN, parameters are passed by reference. In COBOL, since all variables are global variables, there is no parameter passing. In Ruby, parameters are passed by value.
3. Functionality of submodules and the program flow
    a. Submodules
        i. Read file: Parse the input file to get the necessary information and store the pattern in array
        ii. Copy pattern: Copy the array storing the pattern to another array
        iii. Pattern simulation: Loop though each cell in pattern to produce next generation
        iv. Pattern comparison: Compare current generation with previous one to check for still life
        v. Count Surrounding cells: Used in simulation to count the number of surrounding living cell
        vi. Still life handling: Check which case the pattern is and generate corresponding string
        vii. Write file: Format the pattern and string properly and write to output file
    b. Program flow
    First we read the file and pattern to extract and then save the information to corresponding variables. Then we start to simulate the pattern. We do that in a loop and run until we reached the stated generation or the pattern become still life. In the loop, we first copy the pattern to another array as the previous generation pattern, afterward we perform counting on each cell in the pattern to get the number of living surrounding cells. Depend on the number to we write different value to the pattern array. Next, we check if the pattern is still life by comparing it to previous generation. If it is we break the loop and generation the corresponding string, and finally write the pattern and string to file. We also break the loop if we reached the number of generation required to simulate. Otherwise if it is not still life we continue the loop until we meet the previous stated condition.