CSCI3180 – Principles of Programming Languages – Spring 2013

Assignment 1 — The Game of Life Simulator in FORTRAN and COBOL

Deadline: Feb 10, 2013 (Sunday) 23:59

1 Introduction

In this assignment, you have to implement a Game of Life simulator. You need to implement it once using FORTRAN and once using COBOL.

2 Assignment Details

2.1 The Game of Life

The Game of Life, despite its name, is not a competitive game in the conventional sense. Instead, it is a pattern generation game. Given the initial pattern, your program should create a different pattern (called the *next generation* of the initial pattern) based on a few simple rules. Further generations of the pattern can also be constructed, just by repetitively taking the above step.

The initial pattern of the Game of Life is given on a two-dimensional array of square cells. The square cells can be in one of two different states, one being *live* and another being *dead*. Except the cells on the edge of a pattern, each cell has eight *neighbours*, including the horizontal, vertical, and diagonal ones. The following gives an example of a pattern, where the letter '*' denotes a live cell and the letter '0' denotes a dead cell:

```
00000000000000000000000000000000000000000000000000
000000***00000000000000000000000000000000000000000
00000*000*00000000000000000000000000000000000000000
0000**0000*00000000000000000000000000000000000000000
000*0*0**0**000***00***00000000000000000000000000000
00**0*0000*0**0***0*000*00000000000000000000000000000
0*0000*000*0000*0*000000*00000000000000000000000000
0000000000000*00000**000*00000000000000000000000000
0**0000000**0000000000*00000000000000000000000000000
00000000000000000000000*0*00000000*0000000000
00000000000000000000**0*0**00*00000*000000000
0000000000*00000000*0*0*0*0***00000*000000000
000000000*00000*00**0*0*0*0*0*0*****000000000
000000000*00000***0*0*0*0*0*00000000000000000
000000000*****0*0*0*0*0*0*0*00000000000000000
00000000000000000*0*0*0*0*0*00000000000000000
00000000000000000*0*0*0*0*0*0*0*****000000000
00000000000000000*0*0*0*0*0***00000*000000000
000000000*****0*0*0*0*0*0**00*00000*000000000
000000000*00000***0*0*0*0*00000000*0000000000
000000000*00000*00**0*0**000000000000000000000
0000000000*00000000*0*00000000000000000000000
00000000000000000000*0000000000**0000000**0
00000000000000000000*000**00000*0000000000000
00000000000000000000*000000*0*0000*000*0000*0
00000000000000000000*000*0***0**0*0000*0**00
00000000000000000000***00***000**0**0*0*000
00000000000000000000000000000000*0000**0000
00000000000000000000000000000000*000*00000
00000000000000000000000000000000***000000
00000000000000000000000000000000000000000000000000
```

The next generation of a pattern also contains a grid of cells with the same size, i.e. the same number of rows and columns. To construct the next generation, the number of live neighbours of each cell is counted, and, based on the following three rules, the state of the cell in the next generation is determined:

1. **Birth**
   A dead cell with exactly three live neighbours becomes a live cell in the next generation.

   In the following examples, the cell at the center will become alive in the next generation:

```
0*0     **0     00*
*0*     000     *00
000     00*     0*0
```

2. **Survival**
   A live cell with two or three live neighbours stays alive in the next generation.

   In the following examples, the cell at the center will stay alive in the next generation:

```
0*0     *00     00*
***     0*0     0*0
000     00*     0*0
```

3. **Overcrowding or Loneliness**
   In all other cases, a cell dies or remains dead in the next generation.

   In the following examples, the cell at the center will become dead in the next generation:

```
0*0     *00     00*     00*
***     0*0     00*     00*
0*0     000     0**     000
```

The following example gives an initial pattern and two generations after taking 1 step and 2 steps respectively:

```
0000        00*0        0000
0***   =>   *00*   =>   0***
***0        *00*        ***0
0000        0*00        0000
```

One kind of pattern that interests us is called **still life**. A *still life* pattern has a property that the pattern remains the same from step to step. No live cells die and no dead cells are re-born in the next generation. The simplest example of still life is called *block*:

```
0000
0**0
0**0
0000
```

You are encouraged to apply the three rules to check if the pattern is a still life or not. Those who are interested in the Game of Life can check out the site for your reading pleasure:

`http://www.math.com/students/wonders/life/life.html`

2.2 General Specification

You are required to write two programs, one in FORTRAN and the other one in COBOL, to simulate the pattern generation of the Game of Life, and to check whether the given pattern is a still life.

1. **Input and Output Specification**
   Your programs should read the input file, which contains the name of the initial pattern, the number of steps to simulate, the number of rows and columns of the game, and the initial pattern of the game. The simulation result, together with the result of still life checking, should be stored in an output file.

   For FORTRAN, the name of the input file should be passed to your program as parameters in the command line:

   ```
   ./life input.txt
   ```

   For COBOL, you should "hardcode" the name of the input file in your program **EXACTLY** as "input.txt".

   The simulation result, together with the result of still life checking, should be stored in an output file. The naming of the output files should be as follows:

   Output ASCII filename for FORTRAN: **XXXXXfor.txt**
   Output ASCII filename for COBOL: **XXXXXcob.txt**

   The **XXXXX** in the output filenames is the name of the pattern specified in the input file. For example, if the name of the pattern is *boat*, the output filenames for FORTRAN and COBOL should be **boatfor.txt** and **boatcob.txt** respectively. Your programs can overwrite the output files.

2. **Limitation on FORTRAN and COBOL**
   In order to let you program as in the old days, ONLY 2 keywords are allowed in selection and loop statements: "IF" and "GOTO". You are not allowed to use modern control constructs, such as if-then-else or while loop. Using any other keywords will receive marks deduction.

3. **Error Handling**
   The programs should also handle possible errors gracefully by printing meaningful error messages to the standard output. For example, failure to open a non-existing file. However, you **CAN** assume that the format of the input file is free of error.

4. **Good Programming Style**
   A good programming style not only improves your grade but also helps you a lot when debugging. Poor programming style will receive marks deduction. Construct your program with good readability and modularity. Provide enough documentation in your codes by commenting your codes properly but not redundantly. Divide up your programs into subroutines instead of clogging the main program. The main section of your program should only handle the basic file manipulation such as file opening and closing, and subroutine calling. The main purpose of programming is not just to make it right but also make it good.

5. **Other Notes**
   You are **NOT** allowed to implement your program in another language (e.g. Assembly/C/C++) and then initiate system calls or external library calls in FORTRAN and COBOL. Your source codes will be compiled and PERUSED, and the object code tested!

   Do not implement your programs in multiple source files. Although FORTRAN and COBOL do allow you to build a project with subroutines scattered among multiple source files, you should only submit one source file for each language.

   NO PLAGIARISM!!!! You are free to design your own algorithm and code your own implementation, but you should not "steal" codes from your classmates. If you use an algorithm or code snippet that is publicly available or use codes from your classmates or friends, be sure to cite it in the comments of your program. Failure to comply will be considered as plagiarism.

   A crash introduction to FORTRAN and COBOL will be given in the upcoming tutorials. Please DO attend the tutorials to get a brief idea on these two languages, and then learn

the languages by yourselves. For a more in-depth study, we encourage students to search relevant resources on the Internet (just Google it!).

2.3 Input File Format Specification

The input file is a plain ASCII text file. Each line is ended with the characters '\r\n'. You should strictly follow the format as stated in the followings:

1. The first line of the file gives you the name of the pattern. The name of the pattern is a string with maximum length of 80 characters, containing **no** spaces.
2. The second line has an integer that tells you the number of generations required to simulate. The maximum number of generations is 10000.
3. The third line contains two integers. The first one is the number of rows occupied by the pattern, while the second one is the number of columns occupied by the pattern. The two numbers are separated by exactly **one** space. The maximum number of rows is 100, while the maximum number of columns is 80.
4. The subsequent lines are the given pattern. A dead cell is represented by `0` and a live cell is represented by `*`. The pattern occupies exactly the number of rows and columns, as stated before, without any spaces in between two cells.

The following is an example of the input file:

```
loaf
3
6 7
0000000
00**000
0*00*00
00*0*00
000*000
0000000
```

2.4 Output File Format Specification

Each output file should contain two sections. Each line is ended with the characters '\r\n'. Section 1 is the generated pattern. The generated pattern should be the same size as the input pattern, having the same number of rows and columns. A dead cell should be represented by `0` and a live cell should be represented by `*`.

Section 2 follows section 1 immediately, and is a statement concerning whether the pattern is a still live. There are three different cases:

1. If the given pattern is already a still life, the statement should be:

   ```
   It is a still life initially.
   ```

2. If the pattern is a still life after several steps, say 3, the statement should be:

   ```
   It is a still life after 3 steps.
   ```

   The number of steps should be printed properly without any extra spaces or zeros before or after it. Note the plural form of the word *step*. It should be in singular form if the number of steps is 1. Note that you must have finished 4 generations to get the above result.

3. If the generated pattern is not a still life, the statement should be:

   ```
   It is still not a still life even after N steps.
   ```

   where N is the number of generations specified in the input file. The number of steps should be printed properly without any extra spaces or zeros before or after it. Note the plural form of the word *step*. It should be in singular form if the number of steps is 1. Note that you must have finished N+1 generations before you can conclude with the above result.

Here is the content of the output file, based on the input file above:

```
0000000
00**000
0*00*00
00*0*00
000*000
0000000
It is a still life initially.
```

2.5 Report

You simple report should answer the following questions within one A4 page:

1. Compare the conveniences and difficulties in implementing the Game of Life Simulator in FORTRAN and COBOL. You can divide the implementation into specific tasks such as "reading file in certain format", "producing next generation", "case control" and so on. Give code segments in your programs to support your explanation.
2. Compare FORTRAN and COBOL with modern programming languages (e.g. Java/C++/...) from different aspects (e.g. paradigm, data type, parameter parsing, ...). You are free to pick your favorite modern programming language.
3. In your program design, how do you separate the tasks into submodules? Tell us briefly the functionality of each submodule and the main flow of your program in terms of these submodules.

3 Submission Guidelines

Please read the guidelines CAREFULLY. If you fail to meet the deadline because of submission problem on your side, marks will still be deducted. So please start your work early!

1. In the following, **SUPPOSE**

   your name is *Chan Tai Man*,
   your student ID is *1155234567*,
   your username is *tmchan*, and
   your email address is *tmchan@cse.cuhk.edu.hk*.

2. In your source files, insert the following header. REMEMBER to insert the header according to the comment rule of FORTRAN and COBOL.

```
/*
 * CSCI3180 Principles of Programming Languages
 *
 * --- Declaration ---
 *
 * I declare that the assignment here submitted is original except for source
 * material explicitly acknowledged.  I also acknowledge that I am aware of
 * University policy and regulations on honesty in academic work, and of the
 * disciplinary guidelines and procedures applicable to breaches of such policy
 * and regulations, as contained in the website
 * http://www.cuhk.edu.hk/policy/academichonesty/
 *
 * Assignment 1
 * Name : Chan Tai Man
 * Student ID : 1155234567
 * Email Addr : tmchan@cse.cuhk.edu.hk
 */
```

The sample file header is available at

`http://www.cse.cuhk.edu.hk/~csci3180/resource/header.txt`

3. Make sure you compile and run the FORTRAN file without any problem with f77 on Solaris (sparc machine).

4. Make sure you compile and run the COBOL file without any problem with Fujitsu COBOL 3.0 on Windows.

5. The report should be submitted to VeriGuide, which will generate a submission receipt. The report and receipt should be submitted together with your FORTRAN and COBOL codes in the same ZIP archive.

6. The FORTRAN source should have the filename "life.for". The COBOL source should have the filename "life.cob". The report should have the filename "report.pdf". The VeriGuide receipt of report should have the filename "receipt.pdf". All file naming should be followed strictly and without the quotes.

7. `Tar` your source files to `username.tar` by

```
tar cvf tmchan.tar life.for life.cob report.pdf receipt.pdf
```

8. `Gzip` the `tarred` file to `username.tar.gz` by

```
gzip tmchan.tar
```

9. `Uuencode` the `gzipped` file and send it to the course account with the email title "HW1 *studentID yourName*" by

```
uuencode tmchan.tar.gz tmchan.tar.gz \
| mailx -s "HW1 1155234567 Chan Tai Man" csci3180@cse.cuhk.edu.hk
```

10. Please submit your assignment using your Unix accounts.

11. An acknowledgement email will be sent to you if your assignment is received. **DO NOT** delete or modify the acknowledgement email. You should contact your TAs for help if you do not receive the acknowledgement email within 5 minutes after your submission. **DO NOT** re-submit just because you do not receive the acknowledgement email.

12. You can check your submission status at

`http://www.cse.cuhk.edu.hk/~csci3180/submit/hw1.html`.

13. You can re-submit your assignment, but we will only grade the latest submission.

14. Enjoy your work :>