CSCI3180 – Principles of Programming Languages – Spring 2014

Assignment 1 — Crushing Jimmy in FORTRAN and COBOL

Deadline: Feb 9, 2014 (Sunday) 23:59

1 Introduction

In this assignment, you are required to implement the "Crushing Jimmy" game and save Jimmy. You need to implement it once using FORTRAN and once using COBOL.

2 Assignment Details

2.1 Crushing Jimmy

The Crushing Jimmy (Figure 1) game is a variation of general match-three games in which the player manipulates objects to make three objects of the same type adjoin each other in order to make them disappear.

Figure 1: Crushing Jimmy

Jimmy is lost in a candy world. He is locked in a room and is trying to escape. Jimmy is given a target score and needs to play a game to get score. If his score reaches the target one, the door will be opened for Jimmy to escape. The roof of one wall of the room is full of different colored candies and they would fall down the wall until the wall is filled. This roof is magical and would generate candies constantly. Jimmy needs to swap the positions of two horizontally or vertically adjacent candies and match them to make them crushed. A **matching** takes place when candies of the same color form a shape in one of the ways shown in Figure 2 or an **extended** one. Extended matchings are obtained by lengthening some ends of a matching shape with the same colored candies. If it is an extended matching, you cannot consider only a part of it as a matching. Other shapes of matchings would not be considered in this game. Jimmy can only do one swapping. When the candies are matched, they would become very fragile and thus be crushed and vanish. After a crush, more candies in the roof would fall down and fill the wall again. It would continue to match, crush and fill until there are no more matchings or Jimmy is saved. The total number of candies falling down from the roof is the score Jimmy gets. If it reaches the target score, Jimmy would be safe.

In the following, we offer the necessary game rules for you to build your program. Information about the roof and the wall are specified in the given input file. Your main goal is to print the final state of the wall and tell whether Jimmy is safe or not.

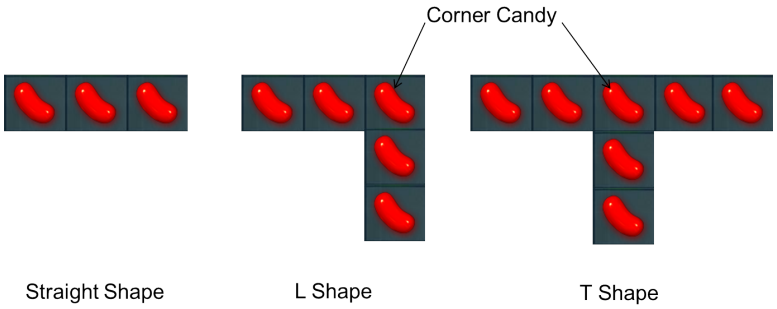

Figure 2: Matching Shape

2.1.1 Game Rules

Initially, you are given the dimensions of the roof and the wall, all the candies initially generated by the roof, the swapping position and the target score. The roof and the wall are in the same horizontal dimension and the vertical dimension of the roof is bigger than that of the wall. You need to use these given candies to fill the wall. There are six types of colored candies: red, orange, yellow, green, blue and purple (denoted by 1, 2, 3 ,4, 5 and 6 respectively), and two types of special candies: striped candies and wrapped candies (denoted by # and @ respectively). Matching would not take place (a) inside the magical roof and (b) for the initial candies filling the wall before swapping is performed. The following gives an example where "|" denotes a location of the candy being crushed because of a vertical matching.

```
      4232                                                  2
      3123                                                  1
Roof  3125                                                  1          2
      1211                                                  2          1
      6224                                                  2          1
      3424       4232       4232       4232       4232       4432       4232
      ---------------------------------------------------------------------------
                 3123       3123       3123       3 23       3 23       3223
                 3125       3215       3|15       3 15       3 15       3415
Wall             1211   =>  1211   =>  1|11   =>  1 11   =>  1211   =>  1211
                 6224       6224       6|24       6124       6124       6124
                 3424       3424       3424       3424       3424       3424
       (0)        (1)        (2)        (3)        (4)        (5)        (6)
```

(0) gives the initial state of the 6×4 roof which is full of candies and the 5×4 empty wall. There are no matchings in the initial state of the roof. The last five rows' candies would fall down into the wall and fill it, as shown in (1). If we swap the second and third candies 12 in row 2, there would be a matching of three vertical 2s (shown in (2)) and all these three candies would be crushed (shown in (3)). After candies are crushed and vanish, candies above it would fall down (shown in (4)). There are vacant spaces on the wall now. The candies in the magical roof would continue to fall down into the wall (the 2). When a column in the roof uses up its candies, the magical roof would generate new candies on demand by recycling the initial candy types of the column (green ones shown in (5)) and fall down to fill the wall. State (6) shows the newly filled wall and the candies in the roof.

When an extended straight matching, an L/T-shape matching or their extended one takes place, a **special candy** would be created. These special candies have super power. They can match with any colored candies and generate an explosion when they are in a matching. We call a matching a **normal matching** when there are no special candies. A **super matching** is a one that involves a special candy. *A matching can contain at most one special candy.*

With matchings defined, the crushing game would be played in the following way.

1. Fill the wall.
2. Do the swapping.

3. Find all the matchings (identify the three matching shapes or extended ones).
4. If there are no matchings, the game ends and Jimmy is locked up forever.
5. Otherwise, do the crushing and create special candies if necessary. Also, if there is a super matching, do the explosion.
6. After candies are crushed and vanish, fill the vacant spaces on the wall with candies above and those generated by the magical roof.
7. Accumulate the total number of candies falling down from the roof. If it has reached the target score, Jimmy wins the game and is safe. Otherwise, go to step 3.

There are two kinds of special candies: striped candies and wrapped candies. These special candies are formed in the following way where "-" denotes a location of the candy being crushed because of a horizontal matching and "O" denotes a location of the candy being exploded.

1. **Striped Candy**
   When an extended straight matching is formed, a striped candy would be created in the following ways:
   (a) If it is a horizontal matching and resulting from the swapping, the striped candy should be generated in the position of the swapped candy that is matched.
   (b) If it is a horizontal matching and not resulting from the swapping, the striped candy should be generated in the middle matched candy's position for an odd number matching or the middle left matched candy's position for an even number matching.
   (c) If it is a vertical matching, the created striped candy should be generated in the lowest matched candy's position.

   After a striped candy is created, if it can compose a super matching with other candies, the entire row involving this candy would be **exploded**. In other words, the entire row would be crushed and vanish. Note that we do not explode the column involving the striped candy. In case of extended straight matchings, L/T-shape matchings or their extended ones, the newly created special candies would also be exploded if they are in an exploded position.

   In the following example, assume the target score is 15. The wall is first filled with candies as shown in state (1). If we swap the second candies of rows 2 and 3, four candies are matched in row 3 (four 1s). These four candies are crushed and vanish but a striped candy # is created in the matched position of the swapped candy. After the wall is filled as shown in state (2), three candies fall down from the roof and the score is 3 now. The striped candy # in state (2) can form a three-candy super vertical matching with the candies above and below. The four 2s in column 3 also form a four-candy matching. Thus a striped candy # is created in the lowest position. The super matching would generate an explosion (the four Os) and thus 9 candies in total would be crushed and vanish. Eight candies fall down from the roof to fill the wall, resulting in state (3), and the score is now 11. The striped candy # in state (3) can also form a three-candy super matching with its adjacent candies, causing an explosion in the entire row (the four Os). Filling the wall again gives state (4). The score is now 15. It reaches the target score, and Jimmy is saved. Therefore state (4) is the final state for this game.

```
3123      3123      4132      4132      4132      3214      3214      6224
3125      3225      3223      3||3      3||3      4422      4422      3214
1211  =>  -#--  =>  3#25  =>  3||5  =>  0000  =>  3223  =>  3223  =>  4422
6224      6224      6224      6||4      6||4      6134      6134      3223
3424      3424      3424      34#4      34#4      34#4      0000      6134
 (1)                 (2)                           (3)                 (4)
```

   In case one swapping forms two matchings. In each matching, a striped candy # is generated in the position of the swapped candy, as in the following simple example.

```
11211       11111       --#--
22122  =>   22222  =>   --#--
```

2. **Wrapped Candy**
   When an L/T-shape matching or an extended one is formed, a wrapped candy would be created. The created wrapped candy should be in the position of the corner candy of the matching. After a wrapped candy is created, if it can compose a matching with other candies, it would **explode** the 3×3 block with this wrapped candy being the center of the block.

   In the following example, assume the target score is 20. The wall is initially filled with candies as shown in state (1). If we swap the first two candies in row 3, five candies are matched in an L shape (the five 1s). These five candies are crushed and vanish but a wrapped candy @ is created in the corner position. Four candies fall down from the roof and the score is 4. The wall is in state (2). Wrapped candy @ in state (2) can form an L-shape super matching with its left candy and the three 2s in column 3. First, this L-shape super matching creates a wrapped candy @ in the corner. An extra effect for this super matching is that it explodes the 3×3 block surrounding this wrapped candy (the nine Os). The newly created wrapped candy for the L-shape super matching is in the explosion's scope and thus it is destroyed by the explosion. Totally ten candies are crushed and vanish. Ten candies fall down from the roof and the score is now 14. That gives state (3). There are no more matchings, meaning the state is final and Jimmy is locked up forever.

```
3123        3|23        3432        3432        3432        6122
3125        3|25        3223        3223        0003        3213
1211  =>    2@--  =>    2@25  =>    --@5  =>    0005  =>    4225
6224        6224        6224        62|4        0004        3424
3424        3424        3424        34|4        34|4        3434
 (1)                     (2)                                 (3)
```

2.1.2 Matching Priority

The following is the priority of discovering candy matchings. Earlier rules dominate latter ones.

1. Super matchings have higher priority than normal matchings.
2. L/T-shape matchings and their extended ones have higher priority than straight matchings or their extended ones.
3. Often there are more than one matching possibility for a given wall pattern. We should scan for matchings from top to bottom and from left to right. As soon as a matching is discovered, it is committed.

To make it clearer, we give an example in the following. The striped candy # is above the wrapped candy @, so that it has higher priority according to rule 3. Thus the striped candy with the candies above and below form a super matching, while the wrapped candy does not.

```
3223        3|23        3|23
3#25        3|25        0000
1211  =>    1|11  =>    1|11
6@24        6@24        6@24
3224        3224        3224
```

2.2 General Specification

You are required to write two programs, one in FORTRAN and the other one in COBOL, to implement Crushing Jimmy. They should show the final state of a given game and tell whether Jimmy is safe or not.

1. **Input and Output Specification**
   Your programs should read the input file, which contains the dimensions of the roof as well

as that of the wall, the initial candies in the roof and the two swapped candies' coordinates on the wall (e.g. (2,1) means the candy in row 2 and column 1). The final state of the wall and the game result should be stored in the output file.

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

```
./crushing input.txt
```

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

The naming of the output files should be as follows:

Output ASCII filename for FORTRAN: **forXXXX.txt**
Output ASCII filename for COBOL: **cobXXXX.txt**

The **XXXX** in the output filenames is the target score in the input file. For example, if the target score is *20*, the output filenames for FORTRAN and COBOL should be **for20.txt** and **cob20.txt** respectively. Your programs should overwrite the output files.

2. **Limitation on FORTRAN and COBOL**
   In order to let you program as in the old days, ONLY the "IF" and "GOTO" control constructs may be used. No modern control constructs, such as if-then-else and while loop, should be used. You are also not permitted to use recursion either. Any infringement will result in a mark 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, wrong coordinates or invalid swap are possible errors. However, you **CAN** assume that the FORMAT of the input file is free of error.

4. **Good Programming Style**
   A good programming style does not only improve your grade, but also helps you debug your programs. Poor programming style will receive mark deductions. Construct your program with good readability and modularity. Provide enough documentation in your codes by commenting your codes properly but not redundantly. Divide 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. Programming is not only about making it right but making 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 across 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 ends with the characters '\r\n'. You should **strictly** follow the format as stated in the followings:

1. The first line has two integers which are the dimensions of the roof. The first is the number of rows and the second is the number of columns. The two numbers are separated by exactly **one** space. The maximum number of rows is 100 and the maximum number of columns is 80.

2. The second line contains two integers, denoting the dimensions of the wall. The first is the number of rows and the second is the number of columns. The two numbers are separated by exactly **one** space. The maximum number of rows is 100 and the maximum number of columns is 80.

3. The third line contains four integers. The first two are the coordinates of the first swapped candy. The last two are the coordinates of the second swapped candy. The four numbers are separated by exactly **one** space.

4. The fourth line contains one integer which is the target score. The maximum target score is 1000.

5. The subsequent lines are the candies in the roof composed of numbers 1 to 6, # and @. These candies occupy exactly the number of rows and columns given in the first line, without any spaces in between two cells.

The following is an example input file:

```
6 4
5 4
2 2 3 2
15
4232
3123
3125
1211
6224
3424
```

2.4 Output File Format Specification

The output file should contain the final state of the wall and one sentence "Jimmy is safe!" or "Jimmy is locked forever!". Each line is ended with the characters '\r\n'.
Here is the content of the output file, based on the input file above:

```
6224
3214
4422
3223
6134
Jimmy is safe!
```

2.5 Report

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

1. Compare the conveniences and difficulties in implementing Crushing Jimmy in FORTRAN and COBOL. You can divide the implementation into specific tasks such as "reading a file in a certain format", "producing next state", "case control" and so on. Give code fragment in your programs to support your explanation.

2. Compare FORTRAN and COBOL with a modern programming language (e.g. Java/C++/...) in different aspects (e.g. paradigm, data types, parameter passing). You are free to pick your favorite modern programming language.

3. In your program design, how do you separate the task 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 syntaxes 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 "crushing.for". The COBOL source should have the filename "crushing.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 crushing.for crushing.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 :>