Comp 251: Assignment 5

Instructor: Jérôme Waldispühl

Due on December 5 at 11h55:00

- Your solution must be submitted electronically on MyCourses.
- You are provided some starter code that you should fill in as requested. Add your code only where you are instructed to do so. You can add some helper methods, but this is at your own risk; helper methods could cause your methods to crash when called from other programs if not added responsibly. Do not modify the code in any other way and in particular, do not change the methods or constructors that are already given to you, do not import extra code and do not touch the method headers. The format that you see on the provided code is the only format accepted for programming questions. **Any failure to comply with these rules will give you an automatic 0.**
- The starter code includes a tester class. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors. We will grade your code with a more challenging set of examples. We therefore highly encourage you to modify that tester class, expand it and share it with other students on the myCourses discussion board. Do not include it in your submission.
- Your code should be properly commented and indented.
- **Do not change or alter the name of one of the files you must submit**. Files with the wrong name will not be graded. Make sure you are not changing file names by duplicating them. For example, main (2).java will not be graded. Make sure to double-check your zip file.
- Do not submit individual files. Include all your files into a .zip file and, when appropriate, answer the complementary quiz online on MyCourses.
- **You will automatically get 0 if the files you submitted on MyCourses do not compile.**
- To some extent, collaborations are allowed. These collaborations should not go as far as sharing code or giving away the answer. You must indicate on your assignments (i.e. as a comment at the beginning of your java source file) the names of the people with whom you collaborated or discussed your assignments (including members of the course staff). If you did not collaborate with anyone, you write "No collaborators". If asked, you should be able to orally explain your solution to a member of the course staff.
- It is your responsibility to guarantee that your assignment is submitted on time. We do not cover technical issues or unexpected difficulties you may encounter. Last minute submissions are at your own risk.
- Multiple submissions are allowed before the deadline. We will only grade the last submitted file. Therefore, we encourage you to submit as early as possible a preliminary version of your solution to avoid any last minute issue.

- Exceptionally, we will waive the late submission penalty for this assignment, but there will be no compilation check for submissions after the initial deadline. We will not accept any submission more than 72 hours after the deadline. The submission site will be closed, and there will be no exceptions, except medical.
- In exceptional circumstances, we can grant a small extension of the deadline (e.g. 24h) for medical reasons only. However, such request must be submitted before the deadline, and justified by a medical note from a doctor, which must also be submitted to the McGill administration.
- Violation of any of the rules above may result in penalties or even absence of grading. If anything is unclear, it is up to you to clarify it by asking either directly the course staff during office hours, by email at (`cs251@cs.mcgill.ca`) or on the discussion board on myCourses (recommended). Please, note that we reserve the right to make specific/targeted announcements affecting/extending these rules in class and/or on the website. It is your responsibility to monitor the course website and MyCourses for announcements.
- The course staff will answer questions about the assignment during office hours or in the online forum on MyCourses. We urge you to ask your questions as early as possible. We cannot guarantee that questions asked less than 24h before the submission deadline will be answered in time. In particular, we will not answer individual emails about the assignment that are sent sent the day of the deadline.
- You should compile all your files directly from command line without using a package, by using the command `javac *.java`

Assignment specifications: This assignment is different from the previous ones. Instead of being given a template with specific instructions, you are given three problems with a set of examples for each. Those three problems can be solved by using concepts seen in class, but we will not be enforcing a particular approach. You must submit three java classes `balloon.java` , `mancala.java` and `islands.java`.

Each of these classes must contain a main method that opens an input file with a fixed name (see each question for that file name), and writes the appropriate output to a text file. You will be given 6 example inputs and outputs in the appropriate file for each exercise. Do not change the name of the input file.

Although you are not restricted in the method you want to use to solve the problem, you must implement this method yourself, and as such you can only use the following import statements:

- `import java.io.*;`
- `import java.util.*;`
- `import java.util.regex.*;`
- `import java.math.*;`
- `import static java.lang.System.out;`

Each of the three questions is weighted 50 points. You can get full marks for the assignment by completing two questions, or you can choose to attempt all 3, with a maximum achievable mark of 120/100. If you choose to not do one of the problems, you must still submit the java class, but make it a dummy that does not write any file.

1. (50 points) **Fun with balloons**
   After a birthday party held in McIntyre 504, N balloons are left in the room, floating from left to right. Bored TAs like to play with bows to practice their hunting abilities. During such a practice, they position themselves at the left end of the room and shoot an arrow towards the right end of the room from an arbitrary height they choose. Because Computer Science TAs are not very good at physics, assume the arrows fly in a straight line at the height H chosen by the shooter. When an arrow comes in contact with a balloon, the balloon disappears and the arrow continues its way from left to right at a slightly lower height H-1, in a straight line. In other words, if the arrow travelling at height H comes in contact with a balloon at position i, then at position i+1, the arrow will be at height H-1, and will maintain that height until it hits another balloon or the wall. Your task is to write a java class `balloon.java` that, given an assignment of balloon positions, will compute the numbers of arrows necessary to destroy all the balloons.
   The input file is named `testBalloons.txt`. Line 1 contains an integer n that represents the number of problems included in the file. Line 2 contains n integers m_i separated by a space, each representing the total number of balloons of the ith problem. Starting at line three, every line describes a new problem. Each line contains m_i integers h_j separated by a space, which represent the respective height of the jth balloon from left to right.
   The output file, named `testBalloons_solution.txt`, contains n lines, one per problem. Each line includes a single integer, representing the number of arrows that need to be shot to destroy all the balloons of the corresponding input.

   To summarize, the command `java balloon` should generate the output file. Same goes for the other two exercises.

2. (50 points) **Mancala Leapfrog**
   When TAs have nothing else to do, they like playing games. Because some of them do not have many friends, they particularly like single-player games, like a specific variant of mancala, played with small pebbles. In that game, the player is given a board with 12 cavities on a single line. Some of them contain a pebble, and some don't. The goal of the game is to remove as many pebbles as possible from the board. Pebbles are removed through moves. A move is possible when there is a straight line of three adjacent cavities A, B, C where A is empty but both B and C contain a pebble. The move consists of moving the pebble from C to A, and removing the pebble at B from the game. Such a move can be executed in both directions.
   Unfortunately, some TAs struggle with the game, and they ask you to write a program to help them. Your java class `mancala.java` should, given an assignment of pebbles over a mancala board of size 12, find a sequence of moves such that as few pebbles as possible are left on the board.
   The input file, named `testMancala.txt`, has, like exercise 1, the number of included problems as its first line. Other lines each contain a problem, represented as a sequence of 12

integers. 0 represents an empty slot, and 1 encodes the presence of a pebble at that position. The output file, named `testMancala_solution.txt`, contains n lines, one per problem. Each line includes a single integer, representing the minimum number of remaining pieces at the end of the optimal move sequence. For example, if the input position can be solved in such a way that no pebbles are left after 4 moves, the output should be 0.

3. (50 points) **Discovering islands in the ocean**
   Back in the 1830s, one of the first grad students at McGill University had the idea of using satellites images to search the oceans for new islands to avoid very expensive exploration voyages. Unfortunately, the resolution on satellite images was not stellar in those times, so the bitmap images they produce require some post-processing. Your task is to help this visionary grad student by writing a java class `islands.java`, which should be given as input an image, more precisely a 2D array of black or white pixels, and output the number of islands found in that image. Each pixel in such an image is either black (#) or white (-). Black pixels are associated with water, and white pixels are parts of an island. All pixels that are adjacent vertically or horizontally should be considered to be part of the same island.
   The input file is named `testIslands.txt`. Line 1 contains an integer n that represents the number of problems included in the file. The rest of the lines contain n input arrays of pixels. Each input begins with a line containing a pair of integers $1 \leq m, n \leq 100$. This line is followed by m lines, each containing exactly n pixels.
   The output file, named `testIslands_solution.txt`, contains n lines, one per problem. Each line includes a single integer, representing the number of unique islands observed in the corresponding input. Part of an island can still be counted as an observation of an island.

You will submit your three java files in a single zip file.