COMP2212 Programming Language Concepts Coursework Submission 2

Deadline: Friday May 5, 4pm

Here are the second set of problems that I would like you to solve in your new Tiling Language. The input and output conventions are exactly as before but I repeat them here for convenience.

The Input and Output Format

For each problem we will declare the name of one or more input files in a simple text-based tile format. The input file name will correspond to a file in the current working directory with an extension ".tl". For example, if the problem input file is called "foo" and your interpreter is executed in directory "C:/Home/Users/jr/TSL/" then the input file will be found at "C:/Home/Users/jr/TSL/foo.tl".

The input files will be in a simple (text-based) tile format as follows :

- each row of the tile will be given on a separate line of the file (separated by a newline character)
- each row will consist of a string of the characters '0' and '1' only where '1' represents a filled cell and '0' represents an empty cell.
- the length of the string in each row will be the same for every row.
- there will be the same number of rows as the length of each row.

For each stated problem, the output should be in the same simple format. The output should not contain any other messages or formatting information and **the output should always be printed to Standard Out**. The output required will be specified as best we can and a visual representation of an example output will be provided in each case. In addition to this, an actual example output file in the simple tile format will also be provided for you to check your own output against.

Additional Problems

Problem 6 - Three way alternation with chamfered corners

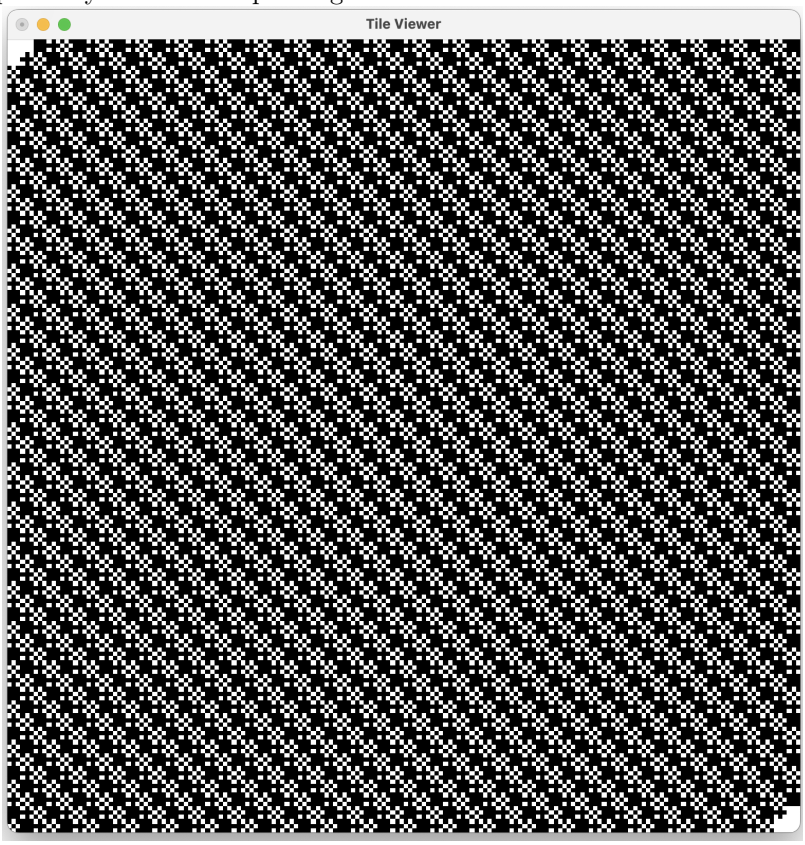
Assume three input tiles `tile1`, `tile2` and `tile3` all of the same size N. Output a single tile pf size $60N$ that comprises the three input tiles in an alternating pattern as follows.

```
Blank-Blank-tile1-tile2-tile3-tile1-tile2-tile3-tile1 ...          ... tile2-tile3-tile1
Blank-tile2-tile3-tile1-tile2-tile3-tile1-tile2-tile3 ...          ... tile1-tile2-tile3
tile3-tile1-tile2-tile3-tile1-tile2-tile3-tile1-tile2 ...          ... tile3-tile1-tile2
tile2-tile3-tile1-tile2-tile3-tile1-tile2-tile3-tile1 ...          ... tile2-tile3-tile1
tile1-tile2-tile3-tile1-tile2-tile3-tile1-tile2-tile3 ...          ... tile1-tile2-tile3
...
tile1-tile2-tile3-tile1-tile2-tile3 tile1-tile2-tile3 ...          ... tile1-tile2-Blank
tile3-tile1-tile2-tile3-tile1-tile2 tile3-tile1-tile2 ...          ... tile3-Blank-Blank
```

Here `Blank` refers to a blank tile of size N. For example, if the three input tiles are

```
111          010          101
111   and    111   and    010
111          010          101
```

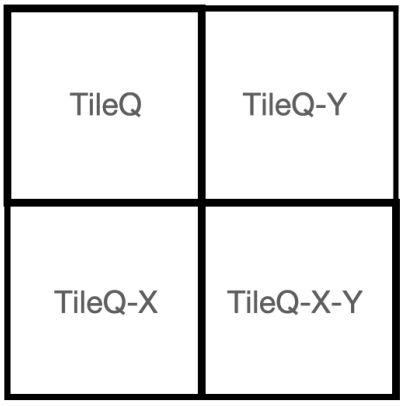
respectively then the output might be rendered as



Problem 7 - Repeated Scaling

Given a single input tile called `tile1` of size N, produce an output tile as follows: scale the input tile to 100 times its original size. So you should have a tile of size $100N$. Now, scale the input tile again but this time scale it by 98 times and pad the top two rows and two left columns with blanks to produce another tile of size $100N$. Repeat this for scaling sizes 96, 94 etc all the way down to scale size 2. You should now have 50 tiles of size $100N$. Create the XOR of all of 50 of these tiles. Let's call this *TileQ* - it should be size $100N$.

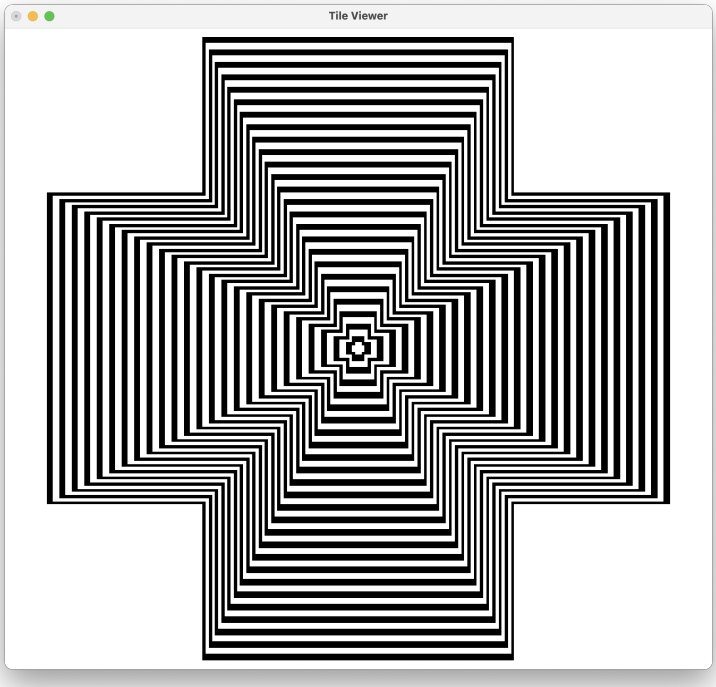
Now, construct the output laid out according to the diagram below



where *TileQ-X* refers to *TileQ* reflected in the X-axis, *TileQ-Y* refers to *TileQ* reflected in the Y-axis, and *TileQ-X-Y* refers to *TileQ* reflected in both the X- and Y-axes. The output tile should therefore be of size $200N$. For example, if the input tile is of size 2 and is given as

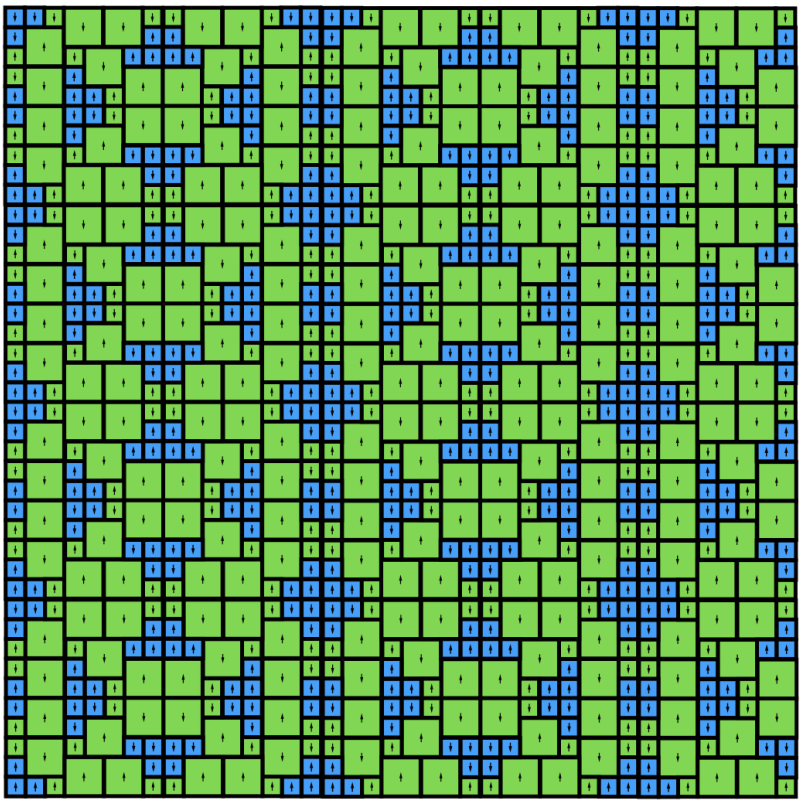
```
01
11
```

then the 400x400 output might be rendered as



Problem 8 - Awkward Composition

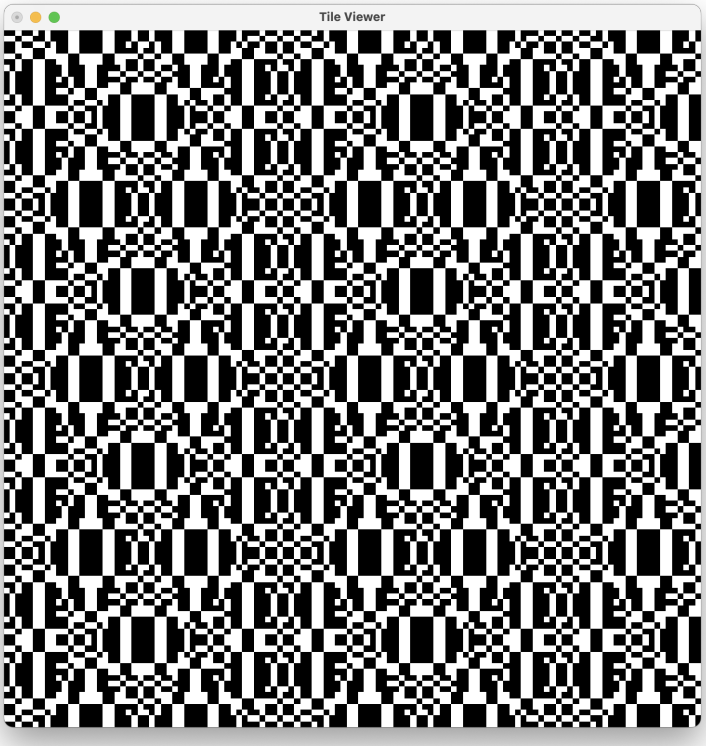
Given two input tiles called `tile1` and `tile2` both of size N, create the output tile of size $40N$ by following the layout template below. The blue tiles are `tile1` and the green tiles are `tile2`. You will need to use double size instances of `tile2` also. The arrows in the diagram also represent the orientation of the tile. An upwards arrow represents the tile as it is given, the downwards arrow represents the tile flipped in the X-Axis.



For example, if the two input tiles are

```
110          010
001    and   101
110          101
```

respectively then the output might be rendered as



Problem 9 - Diamond and Gradients

You will be given a single input tile `tile1` of size N that is symmetric in both X and Y-axes and each of the 90 degree rotations. Create an output tile of size $101N$ as follows: by considering the input tile as being unit size, the output can be considered as having 101 rows and 101 columns, both numbered from 0 to 100.

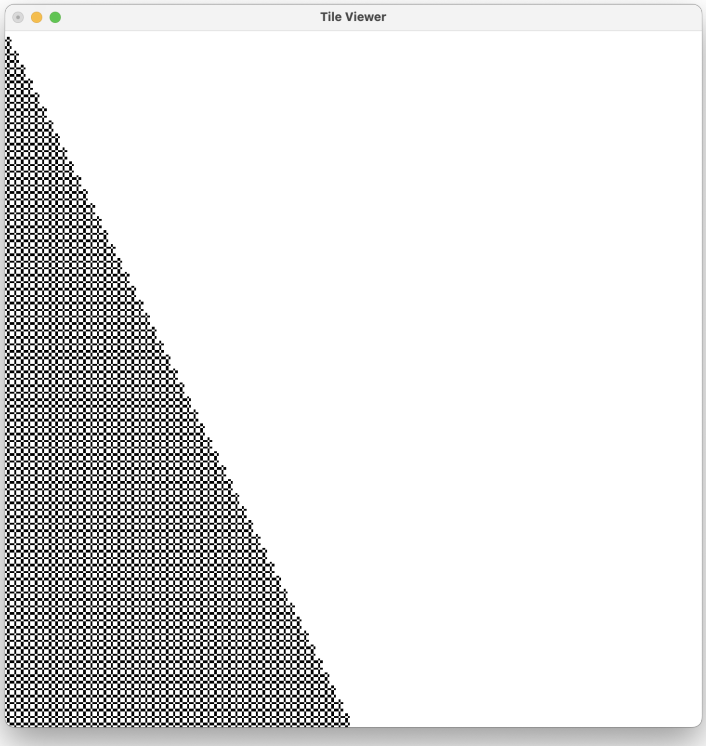
Start by creating a base tile that can be described as follows:

- for entries at (col, row) where $2 * col < row$ there should be `tile1`.
- for all other entries there should be the blank tile of appropriate size.

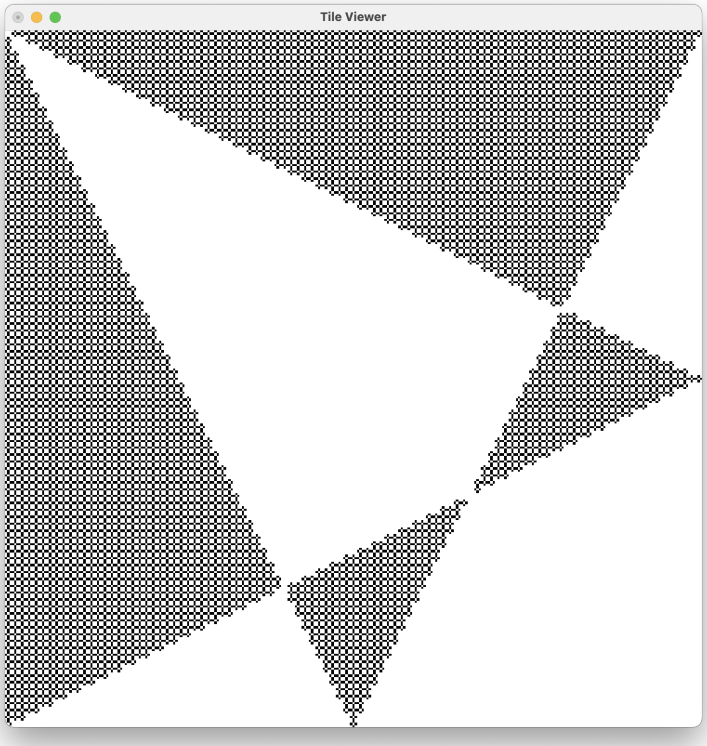
For example, if `tile1` is a tile of size 3 given by

```
010
101
010
```

then the base tile would look like



Now use rotations, flipping and boolean operations on this base tile to create the overall output as depicted below



The actual output file will be provided for you to check your solution against.

Problem 10 - Partial Replacement

Given two input tiles, `tile1` of size at least 20, and `tile2` of size 5, construct the output tile as follows:

Output `tile1` but with the subtile of size 5 of `tile1` with top left corner at column 12 and row 12 replaced entirely with `tile2`.

For example, if `tile1` is

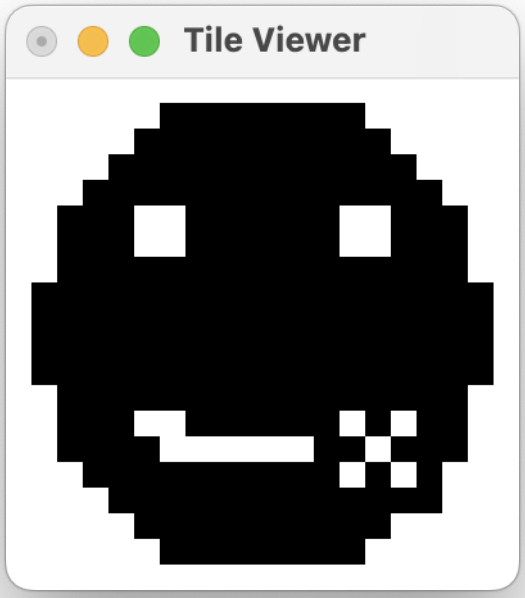
```
00000000000000000000
00000011111111000000
00000111111111100000
00001111111111110000
00011111111111111000
00111001111110011100
00111001111110011100
00111111111111111100
01111111111111111110
01111111111111111110
01111111111111111110
01111111111111111110
00111111111111111100
00111001111111001100
00111100000000011100
00011111111111111000
00001111111111110000
00000111111111100000
```

```
00000011111111000000
00000000000000000000
```

and `tile2` is

```
11111
10101
11011
10101
11111
```

then the output might be rendered as



Submission instructions

What to submit. You will need to submit a single zip file containing:

First, provide the source code for five programs named (`pr6.tsl`, `pr7.tsl`, `pr8.tsl`, `pr9.tsl`, `pr10.tsl`) written in your language that solve the additional problems. We will run our tests on your solutions and award marks for solving the additional problems correctly.

DO NOT RESUBMIT YOUR INTERPRETER CODE UNLESS YOU NEED TO.

Second, a **five** page report on your language **in pdf format** named 'report.pdf' that explains the main language features, its syntax, including any scoping and lexical rules as well as additional features such as syntax sugar for programmer convenience, type checking, informative error messages, etc. In addition, the report should explain the execution model for the interpreter, e.g. what the states of the runtime are comprised of and how they are transformed during execution.

This report, together with the five programs will be evaluated qualitatively and your marks will be awarded for the elegance and flexibility of your solution and the clarity of the report.

Third, a text file called 'group.txt' that contains, the names and usernames of all students in your group, as well as an agreed declaration of the relative contribution amongst the members of your group. For example, write "equal contribution", or "40%-40%-20%" etc. If such an agreed declaration cannot be made then please state this in your declaration instead and arrange for each member of the group to submit separately. Otherwise, please make sure that there is only one submission per group.

How to submit. All submissions must be done through handin.

Marks. This coursework counts for 40% of the total assessment for COMP2212. There are a total of 40 marks available. These are distributed as follows:

There are 20 marks available for functional correctness allocated as 2 marks available for each of the ten problems. Marks are awarded solely on whether the submitted code passes the automated tests. For each problem, if the code passes all tests for that problem then you are awarded 2 marks, code that partially passes ($\geq 50\%$) the tests is awarded 1 mark.

Submission two also has 20 marks available for the qualitative aspects of your solution, as described in your programming language report.

You have the option of resubmitting the interpreter, for a 50% penalty on functional correctness of all 10 problems. However, you are not allowed to modify and resubmit your solutions to the first five problems. If you decide to resubmit your interpreter in the second submission the maximum possible total coursework mark is therefore capped at 30 marks.

Any late submission to either component will be treated as a late submission overall and will be subject to the standard university penalty of 10% per working day late.