CMPUT 350 Lab 10 Exercise Problems

Rules:

- You can use all course material and man pages, but no other information such as web pages, books, or written notes. Using other information sources during the exercise constitues cheating.
- Your programs must compile without warning using

  ```
  g++ -Wall -Wextra -O -g -std=c++17 ...
  ```

  In case there are compiler warnings or errors, marks will be deducted.
- Test your programs with different values. For now, the speed of your program is irrelevant. So don't spend time on optimization
- You must check for the appropriate preconditions/postconditions. Your program shouldn't crash or have undefined behaviour (**hint**: use asserts)!
- Your programs must be well structured and documented. Use ctrl-x t in Emacs to pretty-print it. Marks are assigned to functionality, program appearance, and comments.
- In case your program hangs, use ctrl-c to terminate it.
- Remember that you need to include the appropriate header files. To find out which ones you need for specific functions such as printf, use then man command.

Submit your solution files `Solve.cpp` on eClass under "Lab 10 / Submission".

Important: Submit often (the system will only accept solutions submitted before 16:50)

In this exercise you will complete the matrix game tool you worked on in the today's prep phase. Start by downloading the files provided.

---

1. [8 marks] In file `Solve.cpp`, implement function

```
void best_response_to_col(const Matrix &A, const Vector &strat2);
```

which gets invoked when using option -c and is analogous to the `best_response_to_row()` you implemented in the prep phase.

Instead of finding a best reponse to a row player strategy, you now need to compute and print a best response strategy to a given column player strategy, together with the row player's value.

Test your implementation by comparing its output to that of `./solve -c < ...` with various input files (`*.mgc` and some you create).

---

2. [10 marks] In file Solve.cpp, implement function

```
void solve(Matrix &A);
```

which gets invoked when using option -s.

It reads a payoff matrix for the row player from `stdin` and computes MiniMax strategies for the row and column players and the game value for the row player, and writes the results to `stdout`.

Strategies are computed by writing an LP to a file, then running external program `lp_solve` on that file, and finally extracting results from the output it generates.

Your task is to write LP representations created from the input payoff matrix to the file `lp_solve` reads, using the LPs for solving matrix games developed in the notes (AI Part 5, pp.25-26). See the comments in `solve()` for details.

HINT: Test your implementation by comparing its output to that of `./solve -s < ...` with various input files (`*.mgs` and some you create). Run `./solve -s < test2x4.mgs` and look at `p1.lp`, etc. for the expected format of how to write to these files in your solution code.

For example, for input `RPS.mgs` :

```
3 3
0 -1 +1
+1 0 -1
-1 +1 0
```

the output of `./solve -s < RPS.mgs` is

```
3 by 3 game
+0.000000 -1.000000 +1.000000
+1.000000 +0.000000 -1.000000
-1.000000 +1.000000 +0.000000
value for row player p1: 0
strategy p1: 0.333333 0.333333 0.333333
strategy p2: 0.333333 0.333333 0.333333
```