CSC8501 Coursework 1 – 2017
The Prisoner's Dilemma
Due 27th October 2017 at 10am

Set by Graham.Morgan@ncl.ac.uk

Specification (what you need to do): You will build a computer program in C++ to demonstrate strategies for The Iterated Prisoner's Dilemma Problem.

Clarifying the subject matter (the problem and suggested approach)

- *Prisoner's Dilemma*:
    - Two prisoners are questioned independently of each other
    - Under questioning a prisoner may take one of two actions: (1) betray the other prisoner; (2) remain silent
    - Sentencing outcomes:
        - Both stay silent: 2 years for each prisoner
        - Prisoner A stays silent whereas prisoner B betrays prisoner A: Prisoner A receives 5 years and prisoner B receives 0 years
        - Prisoner B stays silent whereas prisoner A betrays prisoner B: Prisoner B receives 5 years and prisoner A receives 0 years
        - Both prisoners betray each other: 4 years for each prisoner
- *Tournament*:
    - A game lasts for 200 iterations
    - A prisoner may adopt a defined strategy for playing a game
    - A tournament requires all strategies to compete against each other
    - A tournament determines the most successful strategy

- *Your program:*
    - Your program will create 10 strategies that are encoded using the prisoner strategy interpretation language (provided) and stored in separate files
    - Your program should read in two strategy files (one for each prisoner)
        - Strategy files should be interpreted and 200 iterations of the game played with these two strategies
        - After each iteration all variables relied upon by the strategies in play are updated (so the strategies can make use of them)
        - After 200 iterations your program will display all the variable values (for each prisoner) and identify which strategy has won and the scores achieved
    - Your program should provide a tournament to ensure all strategies are tested against each other exactly once in a full (200 iteration) game
        - After a tournament the overall winning strategy is displayed complete with cumulative score
    - Your program should be able to read in valid strategy files from any of your colleagues in the class

Prisoner strategy interpretation language

Keywords and variables:

- You have five keywords: **IF**, **GOTO**, **BETRAY**, **SILENCE**, **RANDOM**
- You have five operators: **+**, **-**, **>**, **<**, **=**
- You have these variables only:
    - **LASTOUTCOME** (gives the outcome of the last game). This can be one of the following values: **W** (I stayed silent and the other prisoner stayed silent); **X** (I stayed silent but the other prisoner betrayed me); **Y** (I betrayed the other prisoner while they stayed silent); **Z** (I betrayed the other prisoner and the other prisoner betrayed me).
    - **ALLOUTCOMES_W** (how many times has W been the outcome previously for me during the current game?)
    - **ALLOUTCOMES_X** (how many times has X been the outcome previously for me during the current game?)
    - **ALLOUTCOMES_Y** (how many times has Y been the outcome previously for me during the current game?)
    - **ALLOUTCOMES_Z** (how many times has Z been the outcome previously for me during the current game?)
    - **ITERATIONS** (how many iterations of this game has there been?)
    - **MYSCORE** (what is my cumulative score during the current game?/how many years have I built up in sentencing?)

Some structuring rules

- Each statement is on a newline
- Lines are numbered in order using integers (used by **GOTO** statement)
- The **IF** statement is constructed in the manner displayed by this example:

```
10 IF ALLOUTCOMES_W < ALLOUTCOMES_Z GOTO 30
20 BETRAY
30 IF LASTOUTCOME = X GOTO 50
40 RANDOM
50 SILENCE
```

All variables are maintained and updated by your program (i.e., the prisoner strategy interpretation language does not have to keep track of updating its own variables).

RANDOM is a special case variable and is substituted with **SILENCE** or **BETRAY** at runtime (randomly).

Deliverables (what we want to see submitted):

- C++ source code authored by the student
- Executable file containing solution
- The 10 output files produced that describe prisoner strategies
- Output file that shows statistics for a tournament including the highlighting of the most successful strategy

Demonstration (your chance to explain and show your solution):
On Friday 27th October from 10am onwards students will demonstrate their solutions

Learning Outcomes (what we expect you to demonstrate in a general way)

- Be capable of designing and creating programs
- Realise inappropriate/appropriate usage of programming languages
- Understand how to manage memory
- To be able to create and use data structures
- To be able to use condition statements, loops and functions

Marking Scheme (what is worth what):
Marks are out of 25 and are awarded as follows:

- 10 Marks for achieving correct output
- 5 Marks for appropriate file input and output
- 5 Marks for user interface design
- 5 Marks for adherence to the 7 rules of programming (see lecture 1)

Additional direction (making the task clearer):

- **Task 1**: Construct a computer program that will generate 10 valid strategies.
    - **It may be easier to create some by hand first to get an understanding of how they should be structured**
- **Task 2**: Extend your computer program to allow an iterative game of The Iterated Prisoners Dilemma Problem to be played by reading in two strategies. Make sure you print out the desired results to screen and to file
    - **At this point you should test to see if your colleagues' files could be read in (i.e., you and your colleagues' strategy files are interchangeable)**
- **Task 3**: Extend your program to allow a tournament to occur and allow ten strategies to compete against each other
    - **At this point you should be able to determine how well your strategies are doing, especially compared with your colleagues'**
- **Task 4**: Use your program to generate thousands of strategies, compare them against each other in tournaments and then pick the top ten
    - **Using randomness in a restrictive way we should be able to find strategies that are more optimum than our manually derived ones (this is commonly termed a Monte Carlo approach)**
- **Task 5**: Find your best strategy and give it to other members of the class
    - **Who has the best strategy?**