cisc1110, Spring 2012, homework 4

This is the assignment for unit VI, "functions." You are expected to complete the assignment in the C++ language and submit your ".cpp" file. You must complete and submit the assignment on or before the due date of March 28.

This assignment is worth 5 points, plus 1 point of extra credit.

Submission instructions:

- Submit your assignment to me as a .cpp file via the course website: http://www.sci.brooklyn.cuny.edu/~kenny/cisc1110/
- Your document should be titled <lastName>.<firstInitial>_HW4.cpp

Overview

For this assignment, you will create a program that implements a simple yet elegant two-player game called "Iterated Prisoner's Dilemma" or IPD. IPD is a classic well-studied model from the field of Game Theory. It models an interaction between two players who make moves simultaneously and have the choice of either making a "selfish" move that helps themselves or making an unselfish move that helps both players. The model has been studied widely in many disciplines including Economics, Politics, Computer Science, and Philosophy. The basic game is explained by Robert Alexrod in his book The Evolution of Cooperation (1984, p.7-8):

In the Prisoner's Dilemma game, there are two players. Each has two choices, namely cooperate or defect. Each must make the choice without knowing what the other will do. No matter what the other does, defection yields a higher [individual] payoff than cooperation. The dilemma is that if both defect, both do worse than if both had cooperated.

The scoring mechanism for the game, also called a payoff matrix, is shown below:

PD PAYOFF MATRIX	P2's move = Cooperate	P2's move = Defect
P1's move = Cooperate	P1 score = 3 P2 score = 3	P1 score = 0 P2 score = 5
P1's move = Defect	P1 score = 5 P2 score = 0	P1 score = 1 P2 score = 1

In Iterated Prisoner's Dilemma, the above "PD" game is played iteratively (repeatedly).

If you want to read more about Prisoner's Dilemma, go to:

http://plato.stanford.edu/entries/prisoner-dilemma/

For this assignment, your job will be to fill in the program skeleton shown on the next page. This includes completing the main() function, as well as writing 4 functions (detailed below). For 1 point of extra credit, you can create one additional function (described below).

```
#include <iostream>
#include <cctype>
using namespace std;

// function prototypes
void update_score( char p1_move, char p2_move, int &p1_score, int &p2_score );
char all_d();
char all_c();
char tit_for_tat( char opponentsLastMove );

int main() {
  bool more = true; while ( more ) {
    cout << "enter C to cooperate, D to defect, Q to quit\n"; // read user's move (step1)
    // determine computer player's move (step 1)
    // update score for both players (step 2)
    // display both players' moves (step 1) and updated scores (step 2)
  } // end of while()
  cout << "game over!\n";

  return 0;
} // end of main()
```

step 1 (1 point)

- Write the function all_d(), based on the function prototype listed above. This function doesn't do anything except return the character 'D'.
- In the main(), read the user's move and store it in a character variable. Valid moves are 'C' (to Cooperate), 'D' (to Defect) and 'Q' (to quit the game).
- Call the all_d() function that you just wrote in order to determine the computer player's move.
- Display both players' moves.
- Compile, run and test your program. Go back and fix it if it doesn't work properly.

step 2 (1 point)

- Modify your program by writing the function update_score, based on the function prototype listed above.
- Assume that the human user is "player 1" (p1) and the computer is "player 2" (p2).
- The function takes two value parameters: p1_move and p2_move, representing the moves made by player 1 and player 2, respectively. The function takes two reference parameters: p1_score and p2_score, representing the scores of player 1 and player 2, respectively.
- The function should update the two players' scores according to the payoff matrix shown earlier.
- Modify the main() to call this function after both players have moved.
- Modify the main() to display both players' scores after the players' moves have been displayed.
- Compile, run and test your program. Go back and fix it if it doesn't work properly.
  At this point, you should be able to play the game against the computer. The computer's strategy is "All D" (as implemented in the all_d() function). This means that no matter what move you

make, the computer will always Defect. Check out my sample output (shown at the end of the assignment). Yours should match.

step 3 (1 point)

- Modify your program by writing the function all_c(), based on the function prototype listed above.
- The function doesn't do anything except return the character 'C'.
- Modify the main() to call all_c() instead of all_d().
- Compile, run and test your program. Go back and fix it if it doesn't work properly.
- When it does work properly, try playing a few rounds and compare how the scores are different from when the computer was playing "All D". The all_c() function is implementing a strategy called "All C".

step 4 (1 point)

- Modify your program by writing the function tit_for_tat(), based on the function prototype listed above.
- The "Tit for Tat" strategy starts by cooperating on its first move, and then, on all subsequent moves, it plays the same move made by its opponent on the last round. For example, I play 'D' on my first move and the computer plays 'C'. On the second move, I play 'D' again, and the computer plays 'D' (because I played 'D' on my first move). On the third move, I play 'C', and the computer plays 'D' (because I played 'D' on my second move). On the fourth move, I play 'C' again, and the computer plays 'C' (because I played 'C' on my third move). And so on.
- Modify the main() to call tit_for_tat() instead of all_c(). Notice that the function takes one value parameter, which is the opponent's last move. In our case, this will be the move that the human played on the previous round.
- Hint: Create a variable called player1_last_move and initialize it to ' '. When you call tit_for_tat() the first time, it will receive blank as its opponent's last move and that will be the indication that it should cooperate. After the first move, update player1_last_move to contain the move that the human player just entered; that will be passed to (and returned by) tit_for_tat() on subsequent moves.
- Compile, run and test your program. Go back and fix it if it doesn't work properly.
- When it does work properly, try playing a few rounds and compare how the scores are different from when the computer was playing "All D" and "All C".

step 5 (1 point)

- Modify your main() program so that the game starts by asking the human player which computer strategy to play against.
- Then, in the section of the code where you determine the computer player's move, use a switch statement to select and invoke the appropriate function (all_d(), all_c() or tit_for_tat()), depending on which strategy the user selected.
- Compile, run and test your program. Go back and fix it if it doesn't work properly. extra credit step (1 extra credit point)
- Modify your program by creating a fourth strategy and adding it to the list of strategies to choose from.

- This will involve: (1) inventing a strategy and writing a new function to implement it, and (2) modifying the choices in step 5.
- Compile, run and test your program. Go back and fix it if it doesn't work properly.

sample output

Below is sample output from my solution to the assignment:

```
welcome to iterated prisonner's dilemma! enter C to cooperate, D to defect, Q
to quit c
your move = C computer move = D
your score = 0 computer score = 5
enter C to cooperate, D to defect, Q to quit d
your move = D computer move = D
your score = 1 computer score = 6
enter C to cooperate, D to defect, Q to quit d
your move = D computer move = D
your score = 2 computer score = 7
enter C to cooperate, D to defect, Q to quit q
your move = D computer move = D
your score = 2 computer score = 7
game over!
```