```
/*

Copyright 2017 [redacted]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

*/

#include <string>

#include <iostream>

using namespace std;

namespace ScoutDb {

        class Team {

        public:

                unsigned short int teamIndex,
                        matchesPer,
                        matchesPlayed;

                unsigned short int* allies;
                unsigned short int* opponents;

                unsigned char record[3];

                unsigned char ap,
                        wp;
                int sp;

                int allyScoreSum,
                        oppScoreSum;

                double avgScored,
                        avgScoredOn,
                        scoreRatio,
                        opr,
                        dpr,
                        ccwm;

                void init(unsigned short index, unsigned short matchesPerTeam) { //Constructor
                        matchesPer = matchesPerTeam;
                        allies = (unsigned short int*)(malloc(matchesPerTeam * sizeof(unsigned
short int)));
                        opponents = (unsigned short int*)(malloc(2 * matchesPerTeam *
sizeof(unsigned short int)));
                        teamIndex = index;
                        matchesPlayed = 0;
                        sp = 0;
                        ap = 0;
                        allyScoreSum = 0;
                        oppScoreSum = 0;
                        avgScored = 0;
                        avgScoredOn = 0;
                        scoreRatio = 0;
                        for(int i = 0; i < sizeof(record); i++)
                            record[i] = 0;

                        for(int i = 0; i < matchesPerTeam; i++) {
                            allies[i] = 0;
                            opponents[2 * i] = 0;
                            opponents[2 * i + 1] = 0;
                        }
                }

                void upStats(unsigned short int ally, unsigned short *opps, short score, short
oppScore) {
                        if (matchesPlayed < matchesPer) { //For the cases where a team has an
extra match that doesn't count
                            allies[matchesPlayed] = ally;
                            opponents[2 * matchesPlayed] = opps[0];
```

```
                    opponents[2 * matchesPlayed + 1] = opps[1];
                    if (score > oppScore) { //If a win
                        record[0]++;
                        sp += oppScore;
                    }
                    else if (score < oppScore) { //If a loss
                        record[1]++;
                        sp += score;
                    }
                    else { //If a tie
                        record[2]++;
                        sp += score;
                    }
                    allyScoreSum += score;
                    oppScoreSum += oppScore;
                    avgScored = allyScoreSum / (matchesPlayed + 1);
                    avgScoredOn = oppScoreSum / (matchesPlayed + 1);
                    if (avgScoredOn != 0)
                        scoreRatio = avgScored / avgScoredOn;
                    else
                        scoreRatio = DBL_MAX;
                    wp = (2 * record[0]) + record[2]; // 2 WP for win, 1 WP for tie

                    matchesPlayed++;

                }

                return;
            }

        };

}

/*
 * tourneyStats.hpp
 *
 *  Created on: Apr 5, 2017
 *      Author: Lentil
 */

using namespace std;
```

```
#ifndef TOURNEYSTATS_HPP_
#define TOURNEYSTATS_HPP_

#include <string>
#include <iostream>
#include "teamStatTracker.hpp"

namespace ScoutDb {

	class Tournament {
	public: Team *Teams,
		**pTeams;
		unsigned char matchesPerTeam;
		unsigned short int matchCt,
			teamCt;

		string input;


		Tournament(unsigned int matchCount, unsigned char teamCount) {
			matchCt = matchCount;
			teamCt = teamCount;
			matchesPerTeam = floor(4 * matchCt / teamCt);
			Teams = (Team*)(calloc(teamCt, sizeof(Team*)));
			pTeams = (Team**)(calloc(teamCt, sizeof(Team**)));
			for (int i = 0; i < teamCt; i++) {
				Teams[i].init(i, matchesPerTeam);
				cout << "Team Index (Save for referencing team later): " << i <<
endl;
			}
			for (int i = 0; i < teamCt; i++) {
				pTeams[i] = &Teams[i];
			}
		}
	};
}




#endif /* TOURNEYSTATS_HPP_ */
```

```
#include <cassert>
#include <cmath>
#include <cstring>

#include <algorithm>
#include <iomanip>
#include <iostream>

#define MM_EPSILON 1e-10

#define MM_FZERO(f) (fabs((f)) < MM_EPSILON)
#define MM_FEQUAL(f, g) MM_FZERO(f - g)
#define MM_FIXZERO(e, f)                                        \
  {                                                             \
    if (MM_FZERO(e)) f = 0.;                                    \
  }

#ifdef MM_DEBUG
#define MM_TRACEMTX printMtx(arr, rows, cols)
#define MM_TRACE
#else
#define MM_TRACEMTX
#endif

namespace MatrixMath {
	inline void printMtx(double *arr, size_t rows, size_t cols) {
		std::ios fmt(nullptr);
		fmt.copyfmt(std::cout);

		for (size_t r = 0; r < rows; ++r) {
			for (size_t c = 0; c < cols; ++c) {
				if (c) std::cout << (c == rows ? " | " : " ");

				double val = arr[r * cols + c], aval = fabs(val);

				std::cout << std::setw(10) << std::setfill(' ')
					<< (aval >= 1e3 || aval <= 1e-2 && aval != 0. ?
						std::scientific :
						std::fixed)
					<< std::setprecision(3) << arr[r * cols + c];
			}

			std::cout << std::endl;
		}

		std::cout.copyfmt(fmt);
	}

	inline void printMtx(double *arr, size_t dim) { printMtx(arr, dim, dim); }

	class MatrixInverter {
		size_t  rows, cols;
		double *arr;

	public:
		MatrixInverter(double *mat, size_t matSize)
			: rows(matSize), cols(matSize * 2) {
			arr = new double[rows * cols];

			for (size_t row = 0; row < matSize; ++row) {
				memcpy(arr + row * cols, mat + row * matSize, matSize *
sizeof(double));

				for (size_t col = 0; col < matSize; ++col) {
					cell(row, rows + col) = col == row ? 1. : 0.;
				}
			}
		}

		~MatrixInverter() { delete[] arr; }

	private:
		inline double &cell(size_t row, size_t col) {
			assert(row < rows);
			assert(col < cols);
			return arr[row * cols + col];
		}

		void rowMult(size_t row, double fac) {
#ifdef MM_TRACE
			std::cout << "[Inverter] Multiplying row " << row << " by " << fac
				<< std::endl;
#endif

#ifdef MM_DEBUG
			if (!MM_FEQUAL(fac, 1.)) {
```

```
#else
			if (MM_FEQUAL(fac, 1.)) return;
#endif

			for (size_t i = 0; i < cols; ++i)
				MM_FIXZERO(cell(row, i) *= fac, cell(row, i))

#ifdef MM_DEBUG
			}
#endif

		MM_TRACEMTX;
		}

	void rowAdd(size_t from, size_t to, double fac) {
#ifdef MM_TRACE
		std::cout << "[Inverter] Adding " << fac << " times row " << from
			<< " to row " << to << std::endl;
#endif

#ifdef MM_DEBUG
		if (!MM_FZERO(fac)) {
#else
		if (MM_FZERO(fac)) return;
#endif

		for (size_t i = 0; i < cols; ++i)
			MM_FIXZERO(cell(to, i) += cell(from, i) * fac, cell(to, i))

#ifdef MM_DEBUG
		}
#endif

	MM_TRACEMTX;
	}

void rowSwap(size_t a, size_t b) {
#ifdef MM_TRACE
	std::cout << "[Inverter] Swapping rows " << a << " and " << b << std::endl;
#endif

	for (size_t i = 0; i < cols; ++i) std::swap(cell(a, i), cell(b, i));
```

```
		MM_TRACEMTX;
	}

public:
		// Returns the determinant of the input matrix.  You're welcome.
		double gaussElim() {
			double det = 1;

			MM_TRACEMTX;

			for (size_t i = 0; i < rows; ++i) {
				if (MM_FZERO(cell(i, i))) { // To avoid a divide-by-zero
					size_t j = i + 1;

					while (j < rows && MM_FZERO(cell(j, i))) ++j;

					// Everything below us is zero.  Our work here is done.
					if (j == rows) {
#ifdef MM_DEBUG
						std::cout << "[Inverter] Skipping row/column " << i
							<< " due to zeroes." << std::endl;

						det = 0.;
						continue; // You could technically return here, since we know
the
								// matrix isn't invertible now, but we don't to
get the full
								// debug output.
#else
#ifdef MM_TRACE
						std::cout << "[Inverter] No pivot found in row/column " << i
							<< "; stopping" << std::endl;
#endif

						return 0.;	// The matrix isn't invertible.
#endif
					}

					det = -det;
					rowSwap(i, j);
				}

				det *= cell(i, i);
				rowMult(i, 1. / cell(i, i));

				for (size_t j = i + 1; j < rows; ++j) rowAdd(i, j, -cell(j, i));
			}

			for (size_t i = 0; i < rows; ++i) {
				size_t j = 0;

				while (j < rows && MM_FZERO(cell(i, j))) ++j;

#ifdef MM_DEBUG
				if (j == rows) {
					std::cout << "[Inverter] No pivot in row " << i << std::endl;

					continue;
				}
#else
				assert(j < rows); // We should've already stopped if a pivot is missing.
#endif

				assert(j >= i);
				assert(MM_FEQUAL(cell(i, j), 1.));

				if (j == rows) continue;

				for (size_t k = 0; k < i; ++k) { rowAdd(i, k, -cell(k, j)); }
			}

			return det;
		}

		void outputIdent(double *output) {
			for (size_t i = 0; i < rows; ++i) {
				for (size_t j = 0; j < rows; ++j) {
					output[i * rows + j] = arr[i * cols + j];
				}
			}
		}

		void outputInverse(double *output) {
			for (size_t i = 0; i < rows; ++i) {
				for (size_t j = 0; j < rows; ++j) {
					output[i * rows + j] = arr[i * cols + rows + j];
				}
			}
		}

		void printAll(double *output) {
			memcpy(output, arr, rows * cols * sizeof(double));
		}
		};
}

#include <cstdio>
#include <cstring>

#include "tourneyStats.hpp"

#include "matMath.hpp"

using namespace std;

using namespace ScoutDb;

using namespace MatrixMath;

void bubbleSort(Tournament* tournament, int* outArr) {
		int swapBuf;
		bool bIsOrdered;

		for (int i = 0; i < tournament->teamCt; i++) {
			outArr[i] = i;
		}

		do {
			bIsOrdered = true;
			for (int i = 1; i < tournament->teamCt; i++) {
				if (tournament->Teams[outArr[i]].opr < tournament->Teams[outArr[i -
1]].opr) {
					bIsOrdered = false;
					swapBuf = outArr[i];
					outArr[i] = outArr[i - 1];
					outArr[i - 1] = swapBuf;
				}
			}
			for (int i = 0; i < tournament->teamCt; i++) {
```

```
                cout << outArr[i] << endl;
            }
        } while (!bIsOrdered);
}

void updateTeams(Tournament *tourney, unsigned short teamCt, unsigned short *teamInds, short
*scores) {
        for (int i = 0; i < teamCt; i++) {
            if(tourney->Teams[i].teamIndex == teamInds[0]) {
                tourney->Teams[i].upStats(teamInds[1], &teamInds[2], scores[0],
scores[1]);
            }
            else if (tourney->Teams[i].teamIndex == teamInds[1]) {
                tourney->Teams[i].upStats(teamInds[0], &teamInds[2], scores[0],
scores[1]);
            }
            else if (tourney->Teams[i].teamIndex == teamInds[2]) {
                tourney->Teams[i].upStats(teamInds[3], &teamInds[0], scores[1],
scores[0]);
            }
            else if (tourney->Teams[i].teamIndex == teamInds[3]) {
                tourney->Teams[i].upStats(teamInds[2], &teamInds[0], scores[1],
scores[0]);
            }
        }
}

int main(){
        string input;
        unsigned short matchCt,
            teamCt,
            teamIndices[4];
        short scores[2];

        cout << "How many matches are there?" << endl;
        cin >> input;
        matchCt = stoi(input, NULL, 10);
        cout << "How many teams are there?" << endl;
        cin >> input;
        teamCt = stoi(input, NULL, 10);
        Tournament* Tourney = new Tournament(matchCt, teamCt);
        int *sortedTeamList = (int*)(calloc(teamCt, sizeof(int)));
        for (int i = 0; i < matchCt; i++) {
            cout << "What is the index of the first red team?" << endl;
            cin >> input;
            teamIndices[0] = stoi(input, NULL, 10);
            cout << "What is the index of the second red team?" << endl;
            cin >> input;
            teamIndices[1] = stoi(input, NULL, 10);
            cout << "What is the index of the first blue team?" << endl;
            cin >> input;
            teamIndices[2] = stoi(input, NULL, 10);
            cout << "What is the index of the second blue team?" << endl;
            cin >> input;
            teamIndices[3] = stoi(input, NULL, 10);
            cout << "What is the red score for the match?" << endl;
            cin >> input;
            scores[0] = stoi(input, NULL, 10);
            cout << "What is the blue score for the match?" << endl;
            cin >> input;
            scores[1] = stoi(input, NULL, 10);
            updateTeams(Tourney, teamCt, teamIndices, scores);
        }
        bubbleSort(Tourney, sortedTeamList);
        cout << "Rankings by OPR" << endl;
        for (int i = 0; i < teamCt; i++) {
            cout << i + 1 << ": " << sortedTeamList[i] << endl;
        }
}
```