Homework 1

Erik Gregg
605.404 Summer 2010

Verison-controlled code for the assignment can be found here:
http://github.com/hank/life/tree/master/school/605.404.OO.Programming.With.Cpp/HW1/

TicTacToe Class

```
// tictactoe.h
// Contains the TicTacToe class and inline functions
#ifndef TICTACTOE_H
#define TICTACTOE_H
#include <stdint.h>
#include <string>
using std::string;

// TicTacToe Class
// Provides an implementation of the game Tic-Tac-Toe
//
// Upon instantiation, X is given the first move. When a valid move is
// completed, the move method changes the player and increments the move
// counter. The main program then must check if the game is finished, which
// calculates if there is a winning board and sets the finished flag and
// winner appropriately. If the game is finished, the main program may
// print the board, print the winner (or a tie in the event of one), or
// reset and play again. If the game is not finished, another move must be
// made. This process is repeated until a finished board is created.
// Functions to check bounds and find if a space is occupied are available.
// Author: Erik Gregg
// Date: Wed Jun 9 22:55:01 EDT 2010
//
// Copyright (c) 2010, Erik Gregg
// All rights reserved.
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions are
// met:
//
// * Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//
// * Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
```

```
//
// * Neither the name of Erik Gregg nor the names of its contributors
// may be used to endorse or promote products derived from this software
// without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
// IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
// THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
// PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
// LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
// NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
// SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
class TicTacToe
{
  public:
    // Default Constructor
    // Initializes a fresh game for us.
    TicTacToe()
    {
      // Start with a freshly cleared board
      clearBoard();
    }

    // Destructor
    virtual ~TicTacToe()
    { // Does nothing, but it's virtual!
    }

    // Sets the board to an initial condition
    void clearBoard();
    // Executes a move by the current player
    // Increments the move counter for the current game
    bool move(uint8_t x, uint8_t y);
    // Finds if a space is currently occupied
    bool occupied(uint8_t x, uint8_t y) const;
    // Checks to see if the game is in a finished state (win/tie)
    // Returns true when game is over, sets winner, sets finished flag.
    // Returns false if game needs to continue.
    bool finished();
    // Prints the result of a finished game.
    // Not const because it calls finished, which may permute members
    void printResult();
    // Prints the current state of the board
    void printBoard() const;
```

```
    // Returns a string containing the board for testing.
    const string getStringBoard() const;
    // Returns the current number of moves
    char getMoves() const;
    // Returns the current player
    char getCurrentPlayer() const;
    // Returns the game winner, or 0 if there is none.
    char getWinner() const;
    // Finds if a space is in bounds
    bool isInBounds(uint8_t x, uint8_t y) const;

  private:
    // Changes the current player to the other player
    void changePlayer();
    // Returns the given space on the board
    // Warning: offsets must be between 1 and 3.
    char getBoard(uint8_t x, uint8_t y) const;
    // Set a place on the board to a given player
    // Warning: offsets must be between 1 and 3.
    void setBoard(uint8_t x, uint8_t y, char player);
    // Returns the finished flag - should only be called inside finished()
    bool getFinFlag() const;
    // Set the finished flag
    void setFinished(bool finished);
    // Set the number of moves
    void setMoves(uint8_t moves);
    // Set the winner - also sets the finflag to true
    void setWinner(char winner);
    // Set the winner - also sets the finflag to true
    void setCurrentPlayer(char currentPlayer);

    char board[3][3];
    char currentPlayer;
    char winner;
    uint8_t moves;
    bool finflag;
};

// Inlines

// changePlayer
// Change current player to other player
// ASCII X is 0x58, ASCII O is 0x4F
// XOR with 0x17 to swap.
inline void TicTacToe::changePlayer()
{
  this->currentPlayer ^= 0x17;
}
```

```
// getCurrentPlayer
inline char TicTacToe::getCurrentPlayer() const
{
  return this->currentPlayer;
}

// getMoves
inline char TicTacToe::getMoves() const
{
  return this->moves;
}

// getWinner
inline char TicTacToe::getWinner() const
{
  return this->winner;
}

// getFinFlag
inline bool TicTacToe::getFinFlag() const
{
  return this->finflag;
}

// getBoard
// Converts from 1-indexed to 0-indexed array subscript
// Returns value at given offset
// WARNING: MUST CHECK BOUNDS BEFORE CALLING!
inline char TicTacToe::getBoard(uint8_t x, uint8_t y) const
{
  return this->board[x - 1][y - 1];
}

// setBoard
// Converts from 1-indexed to 0-indexed array subscript
// Sets value at given offset
// WARNING: MUST CHECK BOUNDS BEFORE CALLING!
inline void TicTacToe::setBoard(uint8_t x, uint8_t y, char player)
{
  this->board[x - 1][y - 1] = player;
}

// setMoves
inline void TicTacToe::setMoves(uint8_t moves)
{
  this->moves = moves;
}

// setWinner
```

```
inline void TicTacToe::setWinner(char winner)
{
  this->winner = winner;
  this->finflag = true;
}

// setFinished
inline void TicTacToe::setFinished(bool finished)
{
  this->finflag = finished;
}

// setCurrentPlayer
inline void TicTacToe::setCurrentPlayer(char currentPlayer)
{
  this->currentPlayer = currentPlayer;
}

#endif // TICTACTOE_H
```

Implementation

```
// TicTacToe implementation
// Implements functions of the TicTacToe class.
#include <iostream>
#include <string>
using std::string;
#include "tictactoe.h"

// clearBoard
// Clears the board
// Sets all the board spaces the the space character.
// Resets the winner to 0
// Sets the starting player to X
// Resets move counter and finished flag
void TicTacToe::clearBoard()
{
  // Clear board
  for(uint8_t i = 1; i <= 3; i++)
  {
    for(uint8_t j = 1; j <= 3; j++)
    {
      setBoard(i, j, ' ');
    }
  }
  // Clear winner
  setWinner('\0');
```

```
  // Player 1 always starts
  setCurrentPlayer('X');
  // Reset move counter
  setMoves(0);
  setFinished(false);
}

// isInBounds
// Checks to see if the given offsets are inside the board
// Must be used for safety before dereferencing sections of the board.
bool TicTacToe::isInBounds(uint8_t x, uint8_t y) const
{
  // Check bounds
  if(x == 0 || y == 0 || x > 3 || y > 3)
  {
    return false;
  }
  return true;
}

// Occupied
// Finds if a given space is occupied on the board.
bool TicTacToe::occupied(uint8_t x, uint8_t y) const
{
  if(!isInBounds(x, y)) return false;
  // If that succeeds, check the space.
  return (' ' != getBoard(x, y));
}

// Move
// Returns true if move was successful
// Returns false if move was invalid - out of bounds or occupied
bool TicTacToe::move(uint8_t x, uint8_t y)
{
  // Check bounds
  if(!isInBounds(x, y)) return false;

  // Check current state of board space
  if(occupied(x, y))
  {
    // If already occupied, get out
    // Do not increment move counter
    return false;
  }
  // Set the space as owned by X or O
  setBoard(x, y, getCurrentPlayer());
  // Increment move counter
  setMoves(this->moves + 1);
  // Change the player
```

```
  changePlayer();
  return true;
}

bool TicTacToe::finished()
{
  // Maybe we've already set the finished flag
  if(true == getFinFlag()) return true;

  // Check to see if board is a win
  // Check rows
  for(uint8_t i = 1; i <= 3; i++) {
    if(' ' != getBoard(i, 1) &&
              getBoard(i, 1) == getBoard(i, 2) &&
              getBoard(i, 1) == getBoard(i, 3)
      )
    {
      // Row is a win!
      setWinner(getBoard(i, 1));
    }
  }
  // Check columns
  for(uint8_t i = 1; i <= 3; i++) {
    if(' ' != getBoard(1, i) &&
              getBoard(1, i) == getBoard(2, i) &&
              getBoard(1, i) == getBoard(3, i)
      )
    {
      // Column is a win!
      setWinner(getBoard(1, i));
    }
  }
  // Check cross
  if(' ' != getBoard(2, 2) &&
      ( (getBoard(1, 1) == getBoard(2, 2) && getBoard(2, 2) == getBoard(3,
3)) ||
        (getBoard(1, 3) == getBoard(2, 2) && getBoard(2, 2) == getBoard(3,
1))
      )
    )
  {
    // Cross is a win!
    setWinner(getBoard(2, 2));
  }

  // Check for a tie
  if(9 == getMoves()) {
    setFinished(true);
```

```
  }

  // Return true only if we've finished
  return getFinFlag();
}

// printResult()
// Prints a one-line result of the current game.
// Not const because it calls finished()
void TicTacToe::printResult()
{
  // Make sure game is finished
  if(!finished())
  {
    std::cout << "The game is not finished. It's " << getCurrentPlayer()
              << "'s move.\n";
  }
  else
  {
    if(getWinner())
    {
      std::cout << "The winner is " << getWinner() << "!\n";
    }
    else
    {
      std::cout << "Game was a tie!\n";
    }
  }
}

// printBoard()
// Prints the board to the screen in a nice square layout
void TicTacToe::printBoard() const
{
  std::cout << " 1 2 3" << std::endl;
  for(int i = 1; i <= 3; i++)
  {
    std::cout << i << " |";
    for(int j = 1; j <= 3; j++)
    {
      std::cout << getBoard(i, j) << "|";
    }
    std::cout << "\n";
  }
}

// getStringBoard()
// Used for unit testing purposes
// Converts the board array to a std::string
```

```
const string TicTacToe::getStringBoard() const
{
  string s;
  for(int i = 1; i <= 3; i++)
  {
    for(int j = 1; j <= 3; j++)
    {
      s += getBoard(i, j);
    }
  }
  return s;
}
```

Game Main Program

```
// TicTacToe 1-player game
// A version of TicTacToe with a computer opponent.
#include <iostream>
#include <sstream>
#include <string>
#include <stdlib.h>
#include <stdint.h>
#include "tictactoe.h"

// BOOST Random headers
#include <boost/random/mersenne_twister.hpp>
#include <boost/random/uniform_int.hpp>
#include <boost/random/variate_generator.hpp>

// Other RNG stuff
#include <sys/time.h>

int main()
{
  TicTacToe t;
  // Movement variables
  unsigned int x, y;
  std::stringstream ss;
  std::string input;

  // BOOST Random generation support
  // Seed with current microseconds
  struct timeval tv;
  gettimeofday(&tv, NULL);
  boost::mt19937 gen(tv.tv_usec);
  boost::uniform_int<> dist(1, 3);
  boost::variate_generator<boost::mt19937&,
                           boost::uniform_int<> > roll(gen, dist);
```

```
while(1) {
  // Until we finish a game
  std::cout << "To move, put row (1-3) then column (1-3) like so: 1 3"
            << std::endl;
  std::cout << "Then press Enter." << std::endl;
  while(false == t.finished())
  {
    t.printBoard();
    do
    {
      if('X' == t.getCurrentPlayer())
      {
        std::cout << t.getCurrentPlayer() << "'s move: ";
        // Read X and Y from command line
        while(getline(std::cin, input))
        {
          // Clear error state of string stream
          ss.clear();
          // Clear the old string
          ss.str("");
          // Parse the x and y params from the line
          ss << input;
          if(ss >> x >> y)
          {
            // Proper input
            if(!t.isInBounds(x, y))
            {
              std::cout << "Space was out of bounds. Try again."
                        << std::endl
                        << t.getCurrentPlayer() << "'s move: ";
              continue;
            }
            if(t.occupied(x, y))
            {
              std::cout << "Space was already occupied. Try again."
                        << std::endl
                        << t.getCurrentPlayer() << "'s move: ";
              continue;
            }
            else
            {
              // Wonderful. Make the move.
              break;
            }
          }
          else
          {
```

```
            // Bad input
            std::cout << "Bad input. Try again." << std::endl;
            std::cout << t.getCurrentPlayer() << "'s move: ";
          }
        }
        // Make sure we haven't received an EOF
        if(std::cin.eof())
        {
          // Exit
          std::cout << "\nExiting..." << std::endl;
          exit(EXIT_SUCCESS);
        }
        // Make a move
      }
      else
      {
        std::cout << t.getCurrentPlayer() << "'s move" << std::endl;
        // Query PRNG until we get an unoccupied space
        do
        {
          x = roll();
          y = roll();
        }
        while(t.occupied(x,y));
        // Make a move
      }
    }
    while(!t.move(x,y));
    // Successful move!
    // Loop until finished!
  }
  // Game is finished. Who won?
  t.printResult();
  t.printBoard();
  // Continue?
  std::cout << "Press Enter to play again!\n";
  // Get a single character from the keyboard
  // If it's not a newline, break out of the while loop
  if(std::cin.get() != '\n') break;
  // Otherwise, clear the board and continue to the next game
  t.clearBoard();
 }
 return EXIT_SUCCESS;
}
```

Output

```
$ ./tic_1player
```

```
To move, put row (1-3) then column (1-3) like so: 1 3
Then press Enter.
   1 2 3
1 | | | |
2 | | | |
3 | | | |
X's move: 1 1
   1 2 3
1 |X| | |
2 | | | |
3 | | | |
O's move
   1 2 3
1 |X| | |
2 | | | |
3 | | |O|
X's move: 3 2
   1 2 3
1 |X| | |
2 | | | |
3 | |X|O|
O's move
   1 2 3
1 |X| | |
2 | |O| |
3 | |X|O|
X's move: 2 1
   1 2 3
1 |X| | |
2 |X|O| |
3 | |X|O|
O's move
   1 2 3
1 |X|O| |
2 |X|O| |
3 | |X|O|
X's move: 3 1
The winner is X!
   1 2 3
1 |X|O| |
2 |X|O| |
3 |X|X|O|
Press Enter to play again!
```

Unit Tests (Requires GoogleTest)

```
#include <gtest/gtest.h>
```

```cpp
#include <iostream>
#include <string>
using std::string;
using std::cout;
#include "tictactoe.cpp"

TEST(TicTacToe, Move)
{
  TicTacToe t;
  t.move(1,1);
  t.move(1,3);
  t.move(3,1);
  string s = t.getStringBoard();
  ASSERT_STREQ("X O X ", s.c_str());
}

TEST(TicTacToe, TakenSpace)
{
  TicTacToe t;
  ASSERT_TRUE(t.move(1,1));
  ASSERT_FALSE(t.move(1,1));
}

TEST(TicTacToe, ChangePlayer)
{
  TicTacToe t;
  char p = t.getCurrentPlayer();
  ASSERT_EQ('X', p);
  t.move(1, 1);
  p = t.getCurrentPlayer();
  ASSERT_EQ('O', p);
}

TEST(TicTacToe, BoundsCheck)
{
  TicTacToe t;
  ASSERT_FALSE(t.isInBounds(10,35));
  ASSERT_FALSE(t.isInBounds(0,2));
  ASSERT_TRUE(t.isInBounds(1,3));
  ASSERT_FALSE(t.isInBounds(2,4));
}

TEST(TicTacToe, Finished)
{
  TicTacToe t;
  t.move(1, 1); // X
  t.move(3, 2); // O
  t.move(2, 1); // X
  t.move(3, 3); // O
```

```
  t.move(3, 1); // X
  ASSERT_TRUE(t.finished());
}

TEST(TicTacToe, Winner)
{
  TicTacToe t;
  t.move(1, 1); // X
  t.move(3, 2); // O
  t.move(2, 1); // X
  t.move(3, 3); // O
  t.move(3, 1); // X
  ASSERT_TRUE(t.finished());
  ASSERT_EQ(t.getWinner(), 'X');
}
```

Unit Test Output

```
Running main() from gtest_main.cc
[==========] Running 6 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 6 tests from TicTacToe
[ RUN      ] TicTacToe.Move
[       OK ] TicTacToe.Move (0 ms)
[ RUN      ] TicTacToe.TakenSpace
[       OK ] TicTacToe.TakenSpace (0 ms)
[ RUN      ] TicTacToe.ChangePlayer
[       OK ] TicTacToe.ChangePlayer (0 ms)
[ RUN      ] TicTacToe.BoundsCheck
[       OK ] TicTacToe.BoundsCheck (0 ms)
[ RUN      ] TicTacToe.Finished
[       OK ] TicTacToe.Finished (0 ms)
[ RUN      ] TicTacToe.Winner
[       OK ] TicTacToe.Winner (0 ms)
[----------] 6 tests from TicTacToe (0 ms total)

[----------] Global test environment tear-down
[==========] 6 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 6 tests.
```